Solstice Enterprise Manager Administration Guide

Release 2.0

2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

Contents

Preface . . . . . **xxi**

1. Planning for Network Management **1-1**

1.1 The Agent / Manager Model 1-2

1.2 Client / Server Architecture 1-3

1.3 Access Control 1-5

1.4 Distributed Management. 1-5

1.4.1 Using Cooperative Consoles to Link SunNet Manager to EM 1-9

1.4.2 SunNet Manager Application Support. 1-11

1.5 Network Management Protocol Support 1-11

1.5.1 RPC Support 1-11

1.5.2 SNMP Support. 1-15

1.5.3 CMIP Support 1-19

1.5.4 Other Network Management Protocols 1-21

1.6 Adding New Object Classes and Event Types 1-22

2. Populating the MIS. 2-1

2.1 Starting Discover from the Application Launcher 2-2

2.2 Using the Viewer to Create a View. 2-6

2.2.1 Discovering the Routers 2-6

2.2.2 Creating a Routers View. 2-7

2.3 Creating an Object Instance in the Viewer 2-9

2.3.1 Using the Viewer's Object Menu. 2-9

2.3.2 Using the Object Palette 2-11

2.4 Updating the Views of Your Network 2-11

2.5 What's Next? 2-15

3. Managing Devices 3-1

3.1 Fault Management 3-2

3.1.1 Color-Coding of Fault Status in the Viewer. 3-2

3.1.2 Receiving Event Information 3-5

3.1.3 Event Notifications 3-8

3.1.4 Preparing for Fault Management 3-17

3.1.5 For more information... 3-18

3.2 Monitoring Fault Status of Devices 3-19

3.2.1 Launching Requests 3-20

3.2.2 Obtaining a Picture of an RPC Device Through the Data Viewer. 3-26

3.2.3 Obtaining a Picture of an SNMP Device Through the SNMP Browser 3-28

3.3 Performance Management 3-29

4. Device Management Using RPC Agents 4-1

4.1 Overview 4-1

4.2 Managing Network Resources with RPC Agents. 4-4

5. Using Cooperative Consoles with Enterprise Manager 5-1

5.1 Overview 5-1

5.2 Filtering Criteria for Information Forwarding 5-3

5.3 Cooperative Consoles Configuration and Operation. . . . 5-4

5.4 Receiving SunNet Manager Alarms. 5-7

6. SunNet Manager Application Support 6-1

6.1 Overview 6-1

6.2 SNM Applications' Access to Solstice EM Features. 6-4

6.3 Adding an SNM Application to Solstice EM. 6-6

6.4 Importing an SNM Database into EM 6-9

6.5 SNM Applications' Access to SNM Agents (Over Solstice EM) 6-10

6.6 Solstice EM Applications' Access to SNM Agents 6-12

6.6.1 Configuration 6-14

6.6.2 Agent Support 6-14

6.6.3 Support for SNM Proxy Agents 6-14

7. SunNet Manager SNMP Proxy Agents 7-1

7.1 Overview 7-1

7.2 SNMP Proxy Agent Operation 7-4

7.2.1 SNMP Trap Daemon (`em_snmp-trap`) Operation. 7-8

7.3 Schema Files 7-8

7.4 SNMP Version 2 Support 7-11

7.4.1 SNMPv2 Enhancements 7-11

7.4.2 SNMPv2 Files 7-13

7.4.3 Using the v2mib2schema Program 7-13

8. Mapping SNMP Traps to CMIP Event Notifications 8-1

8.1 Trap Daemon Operation 8-1

8.2 The Structure of SNMP Traps 8-4

8.3 Default Trap Mapping 8-6

8.3.1 Default Method for Specifying the Source of the Alarm 8-7

8.3.2 Default `perceivedSeverity` Values 8-7

8.3.3 Default probableCause Values. 8-9

8.3.4 Default `additionalText` Information. 8-9

8.3.5 Default Event Notification Type 8-10

8.3.6 Default Location of Information from Trap Variable Bindings 8-11

8.4 Trap Daemon Behavior When no Mapping Is Provided . 8-11

8.5 Customizing the Mapping of SNMP Traps 8-11

8.5.1 Overview 8-12

8.5.2 Enterprise Mapping Blocks 8-12

8.5.3 Mapping Records 8-13

8.5.4 How to Customize SNMP Trap Mapping 8-15

8.6 Format of Trap Mapping Records. 8-17

8.6.1 Using FDN Templates to Specify the Source of a Trap 8-21
8.6.2 Mapping Restrictions 8-26
8.7 Distributed Trap Handling 8-26
8.7.1 Forwarding SNMP Traps to Other Managers 8-27
9. Nerve Center Overview 9-1
9.1 Overview 9-1
9.2 Guide to Nerve Center Documentation 9-2
9.3 Request Terminology 9-3
9.4 Nerve Center Operation 9-6
9.4.1 How a Request Gets Information 9-6
9.4.2 Variables and Attributes in a Request 9-9
9.4.3 Where and When a Condition is Evaluated 9-11
9.4.4 Action at a Transition 9-11
9.4.5 Specifying the Objects to be Polled 9-12
9.4.6 Alarm Logging and the Alarm Service 9-15
10. Building Request Templates 10-1
10.1 Overview 10-1
10.1.1 Building Blocks of Requests: States, Transitions, and Conditions 10-3
10.1.2 State Machine Diagrams 10-4
10.1.3 Sample Request Template 10-8
10.1.4 Controlling Fault Status Color in the Viewer 10-13
10.2 Designing Request Templates 10-16
10.3 Requests Based on Polling 10-18

10.3.1 Adding States 10-21
10.3.2 Adding Conditions 10-22
10.3.3 Adding Transitions 10-23
10.4 Polling RPC Agents 10-27
10.4.1 Targeting the RPC `ping-reach` Group 10-29
10.4.2 Correlating Information from Multiple Polls 10-31
10.5 Requests Based on Event Subscription 10-33
10.5.1 Example: Subscribing for Enterprise-Specific SNMP Traps 10-34
10.6 Debugging Your Templates 10-40
10.6.1 Nerve Center Debugging Agents 10-41
10.6.2 Turning Off Debug Agents 10-43

11. Building Templates for SunNet Manager Event Requests 11-1

11.1 Overview 11-1
11.2 Nerve Center's SNM Event Request Capability 11-4
11.3 SNM Alarms 11-6
11.4 Building SNM Event Request Templates 11-7
11.4.1 Subscribing for SNM Events 11-10
11.4.2 Sending an SNM `ping` Event Request 11-11
11.4.3 Waiting for a Response to the Event Request 11-13

12. Request Designer 12-1

12.1 Overview 12-1
12.2 Starting the Request Designer 12-3
12.3 Using the Request Designer 12-3

12.3.1 Main Window 12-3
12.3.2 File Menu 12-4
12.3.3 Edit Menu. 12-8
12.3.4 View Menu. 12-8
12.4 Edit States Window 12-9
12.5 Transitions Window 12-10
12.6 Conditions Window 12-16
12.7 Poll Rates Window 12-17
12.8 Severities Window 12-18
12.9 Graphical State Diagram Display 12-18
12.9.1 Creating a Template Through the State Diagram Display 12-20
12.9.2 Other Tasks in the Graphical Display. 12-21
12.10 `em_ncimport` and `em_ncexport` Utilities 12-22
13. Request Condition Language 13-1
13.1 Types of Operands 13-2
13.2 Constants. 13-3
13.3 Variables in a Condition 13-3
13.3.1 Variable Names 13-4
13.3.2 Scope of Variables 13-4
13.4 Data Types. 13-5
13.5 System Variables. 13-6
13.5.1 $pollFdnSet 13-6
13.5.2 $pollfdn 13-7

13.5.3 $eventOI 13-9
13.5.4 $eventInfo 13-9
13.5.5 $eventType 13-10
13.6 Message Types 13-12
13.7 Attributes 13-13
13.7.1 Syntax of Attribute Names 13-14
13.8 Operators 13-15
13.8.1 Logical Operators 13-17
13.8.2 Bitwise Operators 13-17
13.8.3 Precedence and Associativity 13-18
13.9 Control Structures 13-18
13.9.1 IF Constructs 13-19
13.9.2 IF ELSE Constructs 13-19
13.9.3 WHILE Constructs 13-20
13.9.4 FOREACH Constructs 13-21
13.9.5 Nested Constructs 13-22
13.10 Timestamp Arithmetic 13-23
13.11 Error Checking 13-24
14. RCL Functions 14-1
14.1 Summary of RCL Built-in Functions 14-1
14.2 AddressStrToAddress 14-3
14.3 Alarm 14-3
14.3.1 Alarm Logging and Viewer Fault Status 14-4
14.4 AlarmOi 14-6

14.5 AlarmStr 14-7
14.6 AnyStr 14-8
14.7 AppendRdn 14-9
14.8 AsnToStr 14-13
14.9 Defined 14-13
14.10 Extract 14-14
14.11 FinalStr 14-15
14.12 FirstStr 14-15
14.13 GetTimeStamp 14-16
14.14 Include 14-16
14.15 InitialStr 14-17
14.16 IsChoice 14-17
14.17 IsList 14-18
14.18 Mail 14-18
14.19 NameToAddress 14-19
14.20 NameToOid 14-19
14.21 NumElements 14-19
14.22 OiNameToOi 14-20
14.23 OiToOiName 14-20
14.24 Print 14-21
14.25 SendAction 14-21
14.26 SendEvent 14-22
14.27 SendTrap 14-23
14.28 Set 14-24

14.29 SnmEventRequest. 14-25
14.30 SnmKillRequest 14-30
14.31 StrToAsn 14-30
14.32 StrCat 14-31
14.33 Strstr 14-31
14.34 Strstrplus 14-32
14.35 Subscribe 14-32
14.36 SubscribeFilter 14-33
14.37 SubscribeOi 14-36
14.38 TrapGenericType. 14-37
14.39 TrapSpecificType. 14-38
14.40 Undefine 14-38
14.41 Unixcmd 14-39
14.42 UnSubscribe 14-40
15. Adding New Event Types 15-1
16. Adding a Managed Object Class to the MIS. 16-1
16.1 GDMO and ASN.1 Used in the Examples. 16-3
17. Adding a MIB to the MIS 17-1
18. Adding an Object Class Based on an SNM Schema File to the MIS 18-1
19. Configuring Communication with CMIP Agents 19-1
19.1 Overview of CMIP Configuration Tasks 19-1
19.2 Preparing the System for CMIP Configuration. 19-4
19.2.1 Determining the Distribution Model 19-4

19.2.2 Installing the Required SunLink Products 19-4
19.2.3 Gathering Your Configuration Information 19-6
19.3 Compile and Load CMIP Agent Object Types into MIS . 19-6
19.4 Starting and Configuring SunLink OSI 8.1 19-6
19.5 Starting and Configuring SunLink CMIP 8.2 19-8
19.6 Starting and Configuring the CMIP MPA 19-9
19.6.1 Configuring Multiple MPAs on One System 19-14
Index Index-1

Figures

Figure 1-1 Agent / Manager Communication in Solstice EM Environment 1-2

Figure 1-2 EM Application Launcher 1-4

Figure 1-3 A Sample Configuration Using MIS-to-MIS Communication 1-7

Figure 1-4 Topology Tree as Seen by Viewer Connected to MIS A 1-8

Figure 1-5 Topology Tree as Seen by Viewer Connected to MIS Net_B . 1-8

Figure 1-6 Topology as Seen in Viewer Connected to MIS Net_D 1-9

Figure 1-7 MIS-to-MIS Connection from MIS A to MIS Net_B 1-9

Figure 1-8 Forwarding of Information to Central Management Station . 1-10

Figure 1-9 Polling RPC Agents. 1-12

Figure 1-10 Using SNM Event Requests with Solstice EM. 1-14

Figure 1-11 MIS Communication with SNMP Agents 1-15

Figure 1-12 SNMP Trap Daemon Operation 1-17

Figure 1-13 Viewing Trap Notifications in the Alarm Manager 1-18

Figure 1-14 SNMP Proxy Agent Operation 1-19

Figure 1-15 CMIP MPAs in Distributed Configuration 1-20

Figure 1-16 TMN Q3 Connection to EM 1-21

Figure 2-1 Discover Main Window 2-3

Figure 2-2 Discover Properties Window 2-5

Figure 2-3 Gather Window 2-8

Figure 2-4 Save Results Window 2-9

Figure 2-5 Monitor Properties Window 2-13

Figure 3-1 Selecting a Severity for communicationsAlarm Generated by Monitor 3-7

Figure 3-2 CMIP Management of a Cellular Network 3-9

Figure 3-3 Viewing Trap Notifications in the Alarm Manager 3-10

Figure 3-4 Solstice EM Processing of SNMP Traps 3-11

Figure 3-5 Example: SNMP Trap Handling Using SnmpLinkUpDownTrap Request 3-13

Figure 3-6 AlarmLog Discriminator Construct with enterpriseSpecificTraps Excluded 3-15

Figure 3-7 Creating a New Log for enterpriseSpecificTraps 3-16

Figure 3-8 Interaction of Nerve Center Request and EM Applications . . 3-23

Figure 3-9 Using Data Viewer to Obtain hostperf Data 3-27

Figure 3-10 Invoking the SNMP Browser from the Element Icon Menu . . 3-28

Figure 3-11 Using SNMP Browser to Get Router ifTable 3-29

Figure 3-12 Creating a Data Request 3-31

Figure 4-1 Communication with RCP Agents in Direct Polling Requests 4-2

Figure 4-2 Using SNM Event Requests with Solstice EM. 4-4

Figure 4-3 Selecting RPC Agents to be Configured during Network Discovery 4-7

Figure 5-1 Forwarding of Information to Central Management Station . 5-3

Figure 5-2 Information Forwarding from SNM Console to EM MIS 5-7

Figure 6-1 SNM-Solstice EM Compatibility 6-3

Figure 6-2 SNM Application Accessing Solstice EM Features. 6-5
Figure 6-3 SNM Application Accessing SNM Agents over Solstice EM . 6-10
Figure 6-4 Solstice EM Applications Accessing SNM Agents 6-13
Figure 7-1 MIB, GDMO, and Schema Definitions 7-3
Figure 7-2 SNMP Proxy Agent Operation 7-5
Figure 8-1 em_snmp-trap Operation 8-3
Figure 8-2 SNMP Trap PDU Structure 8-5
Figure 8-3 Trap Mapping Record Format 8-17
Figure 8-4 Sample FDN for cmipsnmpProxyAgent Object Instance 8-21
Figure 8-5 FDN Format 8-22
Figure 8-6 Sample FDN for internetSystem Group Object Instance 8-22
Figure 8-7 Sample ifTable FDN 8-23
Figure 8-8 Sample FDN Template 8-25
Figure 10-1 Request Example with Poll Rates and Severities 10-5
Figure 10-2 Request Example with Poll Rates and Severities 10-6
Figure 10-3 Request Example with Conditions 10-7
Figure 10-4 IsSnmpSystemUp Sample Request Template 10-12
Figure 10-5 Using AlarmOi to Log a Minor Alarm 10-14
Figure 10-6 State Diagram of IsSnmpSystemEverDown Template 10-21
Figure 10-7 Entering Condition Code in the Request Designer 10-22
Figure 10-8 Order of Transitions in IsSnmpSystemEverDown 10-24
Figure 10-9 isSnmpSystemEverDown Template 10-26
Figure 10-10 State Diagram of SnmpPingBackoffReachable Request 10-28
Figure 10-11 `get_rpc_mo` Condition 10-29
Figure 10-12 SnmpTrapSubscription Condition 10-36

Figure 10-13 IsSubscriptionError Condition 10-36
Figure 10-14 receivedTrap Condition 10-37
Figure 10-15 State Diagram for `IsEnterpriseSpecificTrap` Template 10-38
Figure 10-16 examineTrap Condition 10-39
Figure 10-17 SNMP Trap Subscription Template 10-40
Figure 11-1 Using SNM Event Requests with Solstice EM..... 11-3
Figure 11-2 State Machine Diagram for `DeviceReachablePing` Template 11-8
Figure 11-3 get_rpcAgent_name Condition 11-9
Figure 11-4 `subscribe_snmAlarmEvent` Condition 11-10
Figure 11-5 `send_ping_reach` Condition 11-11
Figure 11-6 another_event Condition 11-14
Figure 11-7 wakeup_count Condition..... 11-14
Figure 12-1 Import Window 12-6
Figure 12-2 Export Customized Window 12-7
Figure 12-3 Order Transitions Window 12-12
Figure 12-4 Example: Adding AlarmClearedOi as an Action at Up-to-Up Transition 12-14
Figure 12-5 Graphical State Diagram Display 12-19
Figure 14-1 Sample SetInternetSystem Condition..... 14-11
Figure 14-2 IsSystemDesc Sample Condition..... 14-14
Figure 14-3 IsSystemDescr Sample Condition 14-39
Figure 14-4 UndefineSystemDescr Sample Condition 14-39
Figure 19-1 Configuring EM for Communication with CMIP Agents 19-3
Figure 19-2 OCT CMIP Configuration Tool Window..... 19-12

Tables

Table 3-1	Default Color-Coding of Severities	3-3
Table 3-2	Default SNMP Trap Notifications and Severities	3-10
Table 3-3	Mapping of SNM Console Fault Indications to perceivedSeverity Values	3-17
Table 3-4	Request Templates Shipped with Solstice EM	3-20
Table 4-1	RPC Request Templates Shipped with EM	4-9
Table 5-1	Mapping of SNM Console Fault Indications to perceivedSeverity Values	5-8
Table 8-1	Standard SNMP Trap Types	8-6
Table 8-2	Default Color-Coding of Severities	8-8
Table 8-3	Default IP Management Trap Event Types	8-10
Table 8-4	Example: Mapping an SNMP `linkDown` Trap	8-18
Table 10-1	Enterprise Specific Traps Example	10-34
Table 11-1	Mapping of SNM Event Severities	11-6
Table 12-1	Action Menu Items	12-13
Table 12-2	Poll Rates	12-17
Table 12-3	Severities	12-18

Table 13-1 Types of Request Condition Language Constants 13-3
Table 13-2 System Variables Available to a Condition 13-6
Table 13-3 perceivedSeverity Values 13-10
Table 13-4 Values of $messType. 13-13
Table 13-5 Precedence of Operators. 13-18
Table 14-1 Valid Alarm Severities 14-4
Table 14-2 Arguments in <EventRequest> 14-26
Table 14-3 Relational Operators in SNM Request Thresholds. 14-27
Table 14-4 Data Types for Threshold Operands 14-28
Table 14-5 Mapping of SNM Event Severities 14-29
Table 14-6 Standard SNMP Trap Types 14-37
Table 15-1 Default Notification to Event log Record Object Class Mapping 15-2
Table 19-1 `em_oct` Parameters. 19-11

Preface

The *Solstice Enterprise Manager Administration Guide* provides procedures, guidelines, and examples for setting up, customizing, and using Solstice Enterprise Manager™, hereafter referred to as Solstice EM™, to accomplish your network management objectives. This guide also provides reference information on the Solstice EM SNMP trap daemon, Nerve Center, and the Request Designer application.

Our goal in writing this document was to anticipate what you, our customers, would want to do using Solstice EM. Inevitably, we will not have thought of everything. Our hope is that we have described enough tasks, of enough variety, that you can extrapolate from what we have provided to figure out how to perform those tasks that we had not covered.

Who Should Use This Book

This document is intended for network administrators who are responsible for customizing, setting up, and maintaining a Solstice EM network management installation. Users who want reference information on Solstice EM applications (other than Request Designer) should consult the *Solstice Enterprise Manager Reference Manual*.

Before You Read This Book

If you have just acquired the Solstice EM product, you should read Chapter 1, “Planning for Network Management” in this guide for an overview of the Solstice EM architecture and possible scenarios for deploying EM. The *“Overview” chapter in the Solstice Enterprise Manager Reference Manual* also provides an overview of the Solstice EM product functions, features, and components. You should also read the *Solstice Enterprise Manager 2.0 Release Notes* for information on installing and starting, compatibility and minimum machine and software requirements, known problems, an inventory of the product components, and late breaking information about the Solstice EM product. It will be useful to you to at least browse the *Solstice Enterprise Manager Reference Manual*.

How This Book Is Organized

This document is organized as follows:

Chapter 1, “Planning for Network Management,” provides guidelines and examples on how to set up and use Solstice EM to meet network management goals.

Chapter 2, “Populating the MIS,” provides an example on how to use the Discover application to find elements on your network and add them to the Management Information Server (MIS).

Chapter 3, “Managing Devices,” provides procedures and examples for using Solstice EM to do fault and performance management of your network.

Chapter 4, “Device Management Using RPC Agents,” provides procedures and examples for using SunNet Manager Remote Procedure Call (RPC) agents with Solstice EM.

Chapter 5, “Using Cooperative Consoles with Enterprise Manager,” describes the use of Cooperative Consoles to forward management information from Site/SunNet/Domain Manager Consoles to the EM MIS.

Chapter 6, “SunNet Manager Application Support,” describes Solstice EM support for SunNet Manager applications.

Chapter 7, “SunNet Manager SNMP Proxy Agents,” describes the configuration and operation of Solstice Site/SunNet/Domain Manager SNMP proxy agents for managing SNMP devices with Solstice EM.

Chapter 8, “Mapping SNMP Traps to CMIP Event Notifications,” describes the SNMP trap handling capabilities of the Solstice EM SNMP trap daemon, `em_dmuxd`, and the procedure for customizing conversion of SNMP traps to CMIP event notifications.

Chapter 9, “Nerve Center Overview,” provides an overview of the operation of the Solstice EM Nerve Center and an introduction to Nerve Center request terminology.

Chapter 10, “Building Request Templates,” provides guidance on using the Request Designer application and Request Condition Language to build Nerve Center request templates.

Chapter 11, “Building Templates for SunNet Manager Event Requests,” provides information and examples for using the Nerve Center’s SunNet Manager event request capability.

Chapter 12, “Request Designer,” provides information on using the Request Designer application, which is used to create and modify Nerve Center request templates.

Chapter 13, “Request Condition Language,” provides information on the Request Condition Language (RCL) used in building conditions used in Nerve Center request templates.

Chapter 14, “RCL Functions,” describes the built-in functions that can be used to construct conditions that are used as components in Nerve Center request templates.

Chapter 15, “Adding New Event Types,” provides information on adding new event notification types to Solstice EM.

Chapter 16, “Adding a Managed Object Class to the MIS,” provides examples on how to add a GDMO object class to the MIS.

Chapter 17, “Adding a MIB to the MIS,” provides an example on how to add a Sun defined MIB to the MIS.

Chapter 18, “Adding an Object Class Based on an SNM Schema File to the MIS,” provides an example on how to convert a SunNet Manager schema file to a GDMO document and, then, add a GDMO object class for that schema to the MIS.

Chapter 19, “Configuring Communication with CMIP Agents,” provides information on setting up communication between Solstice EM and CMIP Agents. It includes examples on how to configure a CMIP MPA, SunLink OSI, and SunLink CMIP.

Conventions Used in This Book

This section describes the conventions used in this book.

What Typographic Changes and Symbols Mean

The following table describes the type changes and symbols used in this book.

Table P-1 Typographic Conventions

Typeface or Symbol	**Meaning**	**Example**
`AaBbCc123`	The names of commands, files, and directories; on-screen computer output	Edit your `.login` file. Use `ls -a` to list all files. `system% You have mail.`
`AaBbCc123`	What you type, contrasted with on-screen computer output	`system% su` `Password:`
<AaBbCc123>	Command-line placeholder: replace with a real name or value	To delete a file, type `rm` *<filename>*`.`
AaBbCc123	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in the *Solstice Enterprise Manager Administration Guide*. These are called *class* options. You *must* be root to do this.

Shell Prompts in Command Examples

All command line examples in this guide use the C-shell environment. If you use either the Bourne or Korn shells, refer to `sh`(1) and `ksh`(1) man pages for command equivalents to the C-shell. The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

Table P-2 Shell Prompts

Shell	**Prompt**
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	#

User Interface Conventions

The following subsections discuss conventions that apply to the descriptions of the Solstice EM applications.

Mouse/Menu Interactions

We have pursued a minimalist approach in describing a user's interactions with the graphical-based applications in Solstice EM. That is, rather than write:

> To exit, press the right mouse button on the File icon. In the pull-down menu that you receive, move the mouse pointer down to Exit and release the right mouse button.

We write:

> To exit, select File➤Exit.

The symbol ➤ indicates moving down a level, from a button or icon to a menu, or from one menu to another.

The interface to the Solstice EM applications is, with the exception of the Object Editor, standard Motif. Selections are made in the identical way they are made for Motif applications that run on Sun and non-Sun machines.

The following table compares the exhaustive description of a user interaction with the way we have chosen to describe that interaction in this manual:

Table P-3 User Interaction Equivalents

Complete Description	**As Described in this Document**
Select an item by clicking once with the left mouse button.	Select an item.
Activate an item by double-clicking with the left mouse button.	Activate an item.
Press left on the slider in the scrollbar move the slider so that the item comes into view.	Scroll until the item comes into view.
Press right on the icon to obtain the icon pulldown menu. Move the mouse pointer over the item in the menu and release the mouse button.	Select icon➤item. *or* Invoke icon➤item.
Press and hold middle mouse button on the icon. Move the mouse pointer to the target location and release the mouse button.	Drag and drop.

Tear-off Menus

The top-level menus in the Solstice EM applications—those applications accessible through the Application Launcher, plus others—have a type of menu known as a “tear-off” menu. When you select a button, you receive a menu with a dotted line at the top. If you click left on that dotted line, the menu “tears off,” like a sheet of paper from a tablet, and positions itself in a separate window. If you are running the application in a Motif environment, the title displays as “<*menu title*>—Tear-off.” If you are not in a Motif environment, the title displays as “No Name.”

To dismiss a tear-off menu, select the menu title bar to obtain a menu of options. In that menu, select Dismiss. Alternatively, you can simply press Esc while your mouse pointer is in the tear-off menu window.

Connecting to an MIS

All Solstice EM applications rely on a connection with the product kernel—called the Management Information Server or MIS—for their data. All of the Solstice EM applications can connect to an MIS on a local or remote machine. When reading the descriptions of the applications in this manual, keep these facts in mind:

- An application must have a connection to a running MIS.
- An MIS can be on a local or remote machine.

When connecting to a remote MIS, you can use either of two methods. These two methods are described below, using the Application Launcher, invoked from the `em` command, as an example.

- Invoke the application with the `-host` option. For example:

```
% em -host <remote MIS machine> &
```

- Set the `$EM_SERVER` environment variable to the name of the remote MIS machine. For example:

```
% setenv EM_SERVER <remote MIS machine>
% em &
```

Note that we use C-shell syntax for specifying the environment variable. Use the syntax appropriate for your shell.

If you invoke an application without the `-host` option and with the `EM_SERVER` variable not set, the application attempts to connect to an MIS on the local machine.

Planning for Network Management 1

The Agent/Manager Model	*page 1-2*
Client/Server Architecture	*page 1-3*
Access Control	*page 1-5*
Distributed Management	*page 1-5*
Network Management Protocol Support	*page 1-11*
Adding New Object Classes and Event Types	*page 1-22*

Network management is the ability to monitor and control network resources. A network management system should allow you to do the following:

- Detect and correct network problems
- Monitor and evaluate network activity
- Monitor, analyze, and change network configurations

Solstice Enterprise Manager (EM) is a distributed, multi-user management platform, and a set of user applications, that allows you to accomplish these network management goals.

Solstice EM offers various installation options so that you can tailor the product to meet your network management needs. This overview discusses some of the considerations that will affect your decisions about how to deploy components shipped with Solstice EM.

1

1.1 The Agent/Manager Model

Solstice EM is based upon the manager/agent model described in the International Organization for Standardization (ISO) network management standards. Solstice EM can exchange monitoring and control information about network resources with software processes called “agents.” Any network resource that is manageable through this exchange of information is a “managed resource.” This could be an NFS server or a hub or a cellular base station or a WAN link, or components such as a circuit or a router interface, or software entities such as an application or a printer queue. Agents access the managed resource and collect data on behalf of managers.

Agents provide information in response to requests from managers. In addition, agents typically have the ability to issue reports — called *event notifications* — to managers on their own initiative when they detect predefined thresholds or events on a managed resource. Manager/agent communication is illustrated in Figure 1-1.

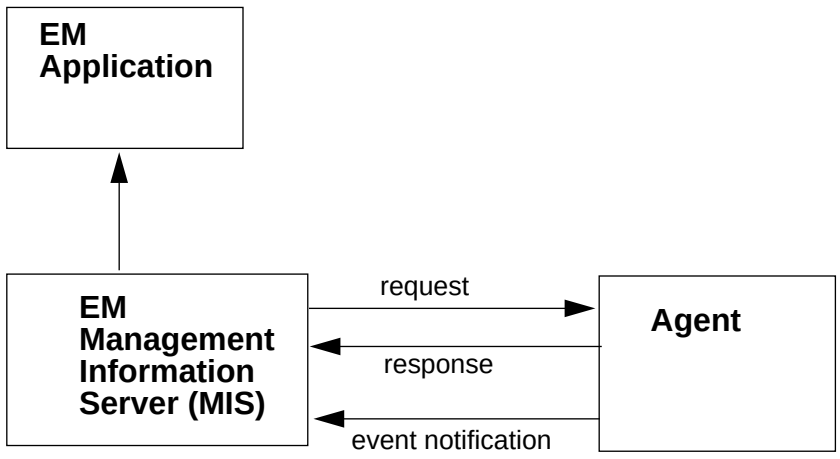


Figure 1-1 Agent/Manager Communication in Solstice EM Environment

A manager relies upon a database of definitions and information about the properties of managed resources and the services that agents support. In Solstice EM this information resides in the Management Information Server (MIS).

1.2 Client/Server Architecture

The management functionality of EM is based on a client/server architecture. Solstice EM is shipped with a set of applications to carry out network management tasks. For example, the EM Viewer provides a graphical, dynamically updated display of your network topology. Coloring of icons is used to represent the fault status of devices displayed in the Viewer. The Viewer's Request tool allows users to launch Nerve Center requests to monitor devices for the occurrence of critical events. In addition to one-at-a-time launching of requests from the Viewer, EM also includes an Auto Manager daemon, which you can activate to automatically launch requests to manage routers, links, or to check hosts for reachability. The Auto Manager is the most efficient method for checking thresholds on large numbers of devices. (The Auto Manager is discussed in the "Automatic Management" chapter in the *Solstice Enterprise Manager Reference Manual*.)

Another key application is the Alarm Manager, which allows users to view and sort incoming alarms and acknowledge or clear them. (The Solstice EM applications are documented in the *Solstice Enterprise Manager Reference Manual*.)

These and other user applications may be installed on machines remote from the machine that runs the MIS. Multiple users, running EM applications on one or more workstations, may be connected to the same MIS. The user's point of entry to most of the EM applications is the EM Application Launcher, shown in Figure 1-2. You can configure the Launcher to include other applications in addition to those shipped with EM.

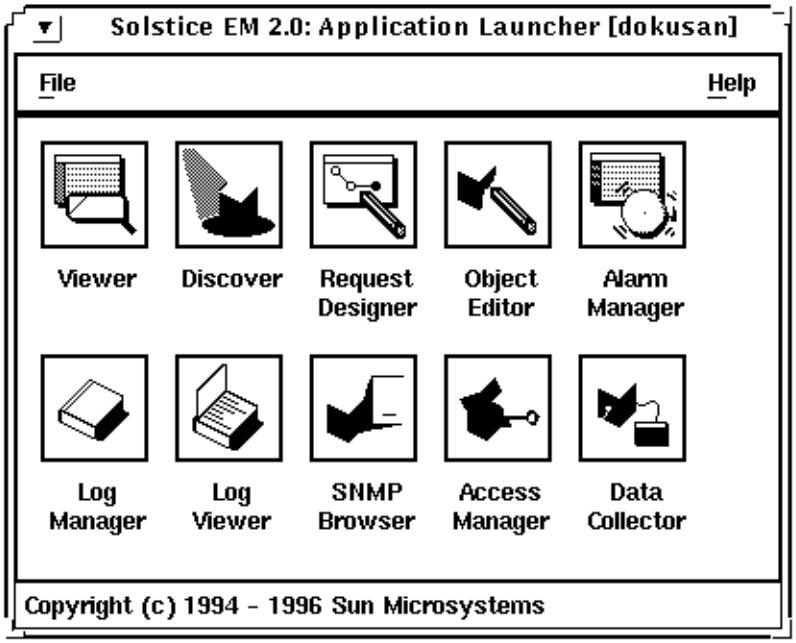


Figure 1-2 EM Application Launcher

EM applications that are installed on the same machine as the MIS can be displayed remotely by means of an X windows session. This is different from the situation where you install the EM applications on a remote machine. In the latter case, EM applications connect to the MIS using a reliable Common Management Information Protocol (CMIP) over TCP / IP connection. In general, applications running on a remote machine consume far less network bandwidth than applications that are run on the MIS machine and displayed remotely.

The multi-user capabilities of EM are based on EM's ability to provide consistent management information to components of the network management solution — operators, applications, other management stations. This enables management tasks to be divided across geography and organization with confidence that all users will see the same view of management data. This is particularly useful in fault management scenarios where cooperation among staff members leads to quicker resolution of problems.

1.3 Access Control

Solstice EM provides the ability to define levels of user access privileges. Access control allows you the flexibility to grant or deny users access to applications or to specific features within applications. The privileges for users are defined through their membership in groups.

The EM Access Manager application allows a system administrator to create groups which define a level of access. Three groups are shipped with EM: full-access, operator, and view-only. If you use the default installation settings, access control is activated. A first step in setting up EM is using Access Manager to define user profiles to the system.

Access control is an optional feature that can be either enabled or disabled. Access control is described in the "Access Manager" chapter in the *Solstice Enterprise Manager Reference Manual*.

1.4 Distributed Management

A particularly powerful aspect of the Solstice EM platform is the ability to distribute the management information base to multiple Management Information Servers while allowing transparent access for users to management data irrespective of whether it resides in the local MIS to which their applications are connected or in a remote MIS in another geographical locale.

The EM MIS Manager application is used to set up and take down such connections. Setting up a connection from one MIS to another is analogous to using NFS to mount a file system from one workstation to another. When a connection is initiated from MIS A to MIS B, the internal Management Information Tree (MIT) of MIS B is "mounted" into MIS A — and becomes visible in the Navigator of a Viewer connected to MIS A.

The user running the Viewer connected to MIS A then has access to the views and devices represented in MIS B. These devices become manageable from the local MIS. For example, the user could launch Nerve Center requests targeted at a device in the topology "tree" of the remote MIS, and this request will execute on the remote MIS. Whether the request is running on the local MIS or a remote MIS is transparent to the user.

Many of the applications shipped with EM have this ability to access managed resources via MIS-to-MIS connections. However, there are some applications (such as Request Designer, Access Manager, and Discover) that only access data in the local MIS.

The example in Figure 1-3 illustrates a possible configuration using MIS-to-MIS communication. In this example, MIS A is a central office “manager of managers” connected to three regional MISs on Net_B, Net_C, and Net_D.

The Viewer connected to MIS A will see a topology like that shown in Figure 1-4. The topology tree for the Viewer connected to MIS Net_B is shown in Figure 1-5. The Viewer connected to the MIS on Net_D, however, sees only the local MIS, as illustrated in Figure 1-6. If the user running the Viewer on mom were to select the Bldg_1_Subnet view, under the MIS Net_B root, the Viewer accesses the data on MIS Net_B and the user sees the same view as the of this subnet as a user running the Viewer connected to MIS Net_B.

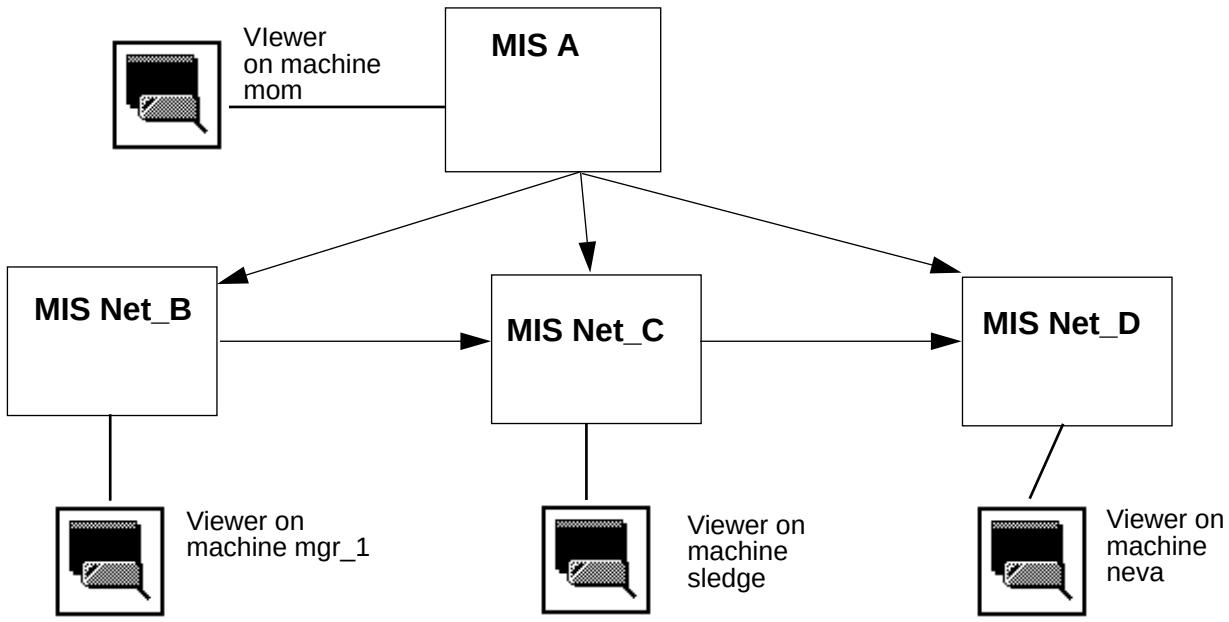


Figure 1-3 A Sample Configuration Using MIS-to-MIS Communication

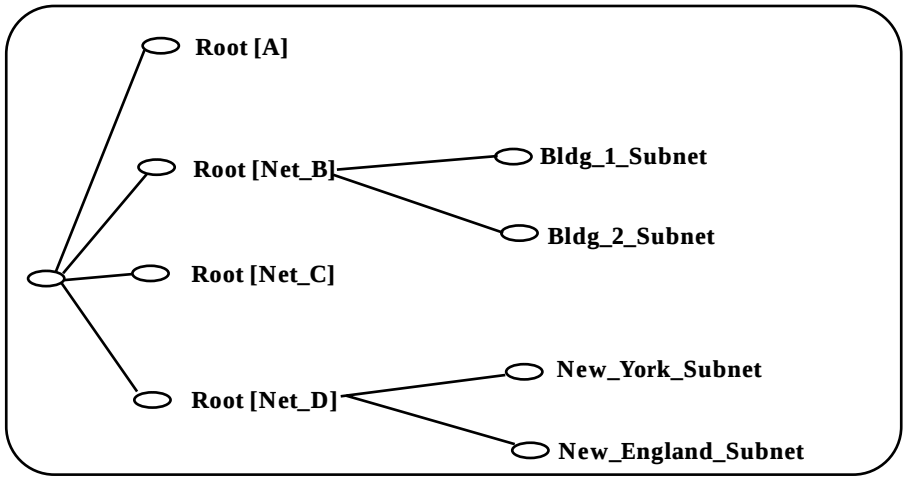


Figure 1-4 Topology Tree as Seen by Viewer Connected to MIS A

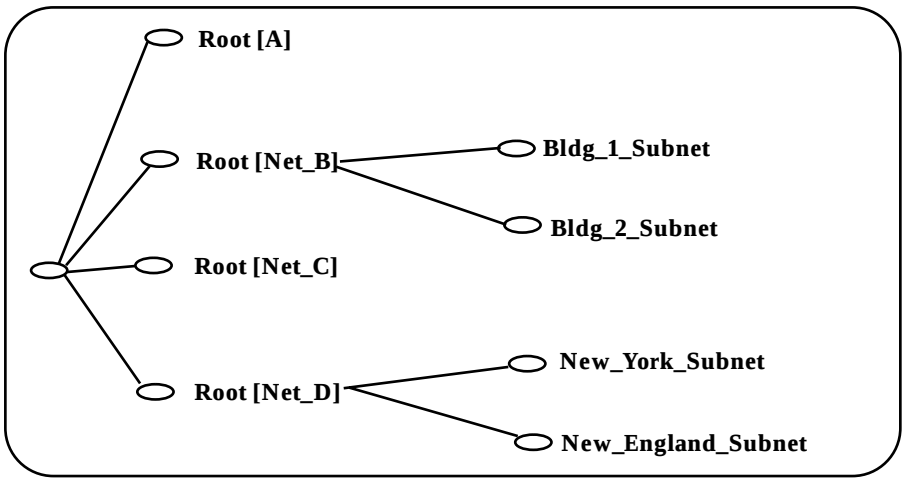


Figure 1-5 Topology Tree as Seen by Viewer Connected to MIS Net_B

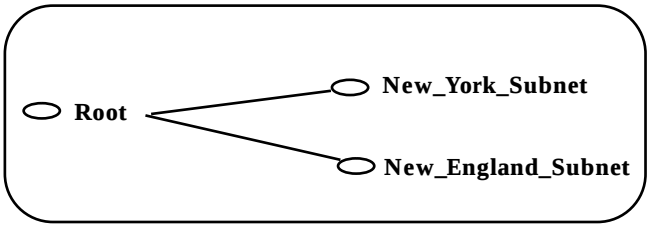


Figure 1-6 Topology as Seen in Viewer Connected to MIS Net_D

When a connection is established from MIS A to MIS Net_B, MIS A takes on the “manager role” in MIS-to-MIS communication, as the initiator of requests for data, and MIS Net_B plays the role of an agent, responding to requests initiated by MIS A. This is illustrated in Figure 1-7. For information about setting up MIS-to-MIS connections, refer to the “MIS-to-MIS Communication” chapter in the *Solstice Enterprise Manager Reference Manual*.

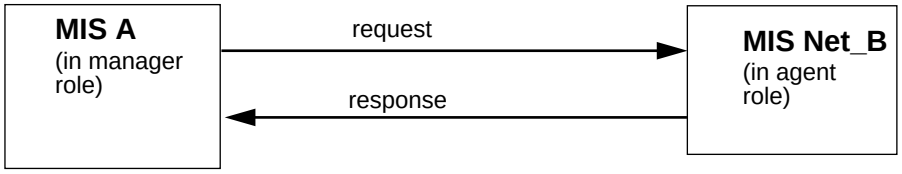


Figure 1-7 MIS-to-MIS Connection from MIS A to MIS Net_B

1.4.1 Using Cooperative Consoles to Link SunNet Manager to EM

Another aspect to Solstice EM’s support for distributed management is the ability to implement forwarding of event and topology information about selected changes in the state of critical network resources or changes in selected aspects of network topology from SunNet Manager or Solstice Domain Manager Consoles to one or more Solstice EM Management Information Servers.

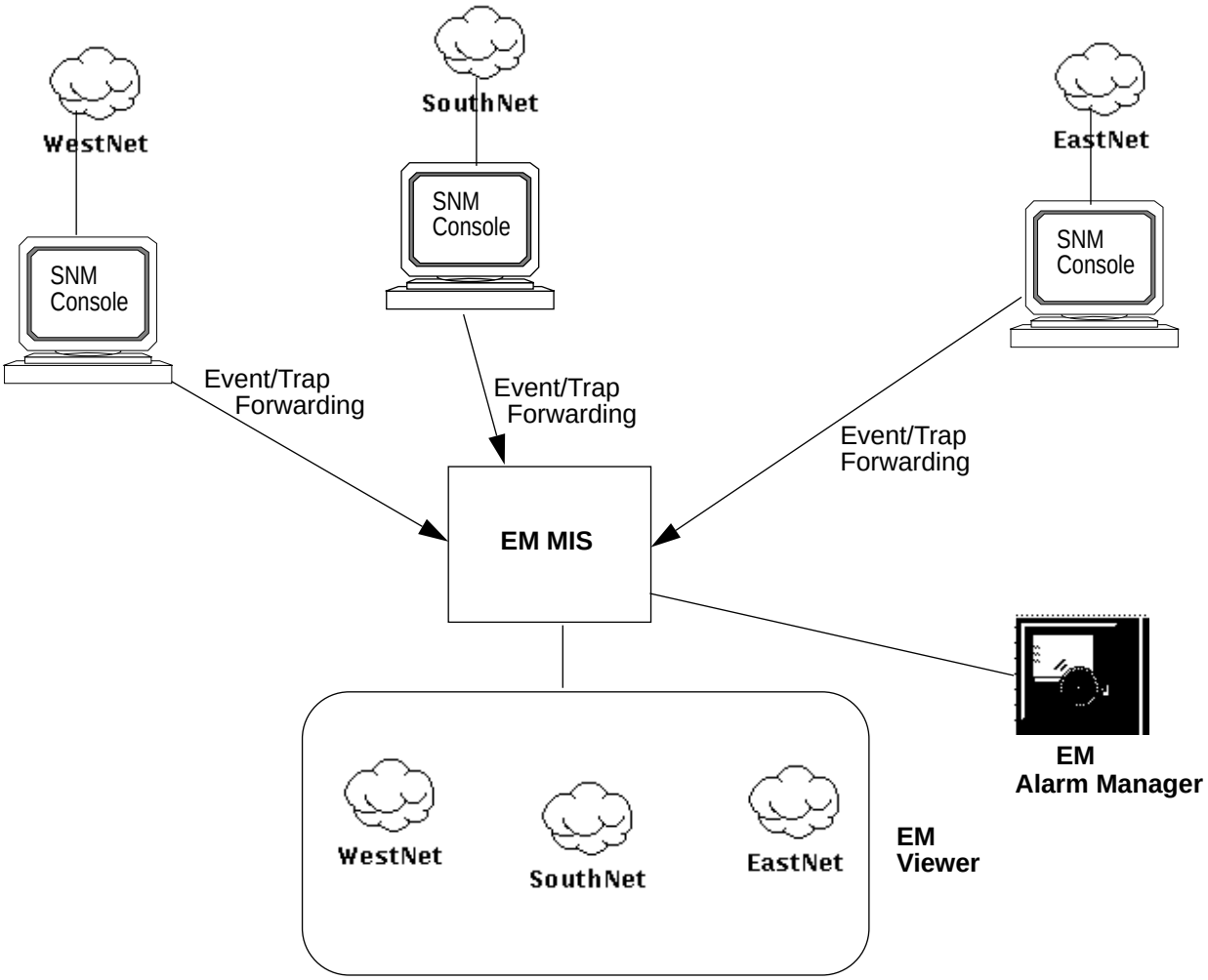


Figure 1-8 Forwarding of Information to Central Management Station

If you already using SunNet Manager (SNM) to manage segments of your network, the Cooperative Consoles Receiver application can be used on an MIS machine to implement one-way forwarding of topology and event information from the SNM Consoles to the MIS. This creates a periphery-to-center configuration in which the MIS functions as a central “manager of managers.”

This configuration is illustrated in Figure 1-8. For a more detailed discussion of Cooperative Consoles, refer to Chapter 5, “Using Cooperative Consoles with Enterprise Manager.”

1.4.2 SunNet Manager Application Support

EM’s ability to interoperate with Cooperative Consoles is an illustration of EM’s support for applications that have been developed for use with SunNet Manager. There are numerous third-party applications developed for SNM that can also be used with Solstice EM. For more information, refer to Chapter 6, “SunNet Manager Application Support.”

1.5 Network Management Protocol Support

A network management protocol defines the types of messages, encoding rules, and how messages are exchanged in communication between a manager and agent. As shipped to you, Solstice EM offers support for three network management protocols:

- Simple Network Management Protocol (SNMP)
- Common Management Information Protocol (CMIP)
- Remote Procedure Call (RPC) protocol (as used by SunNet Manager)

SNMP and RPC are network management protocols used to manage resources in the context of an Internet (IP) network environment. When you install EM, you are asked whether you want support for IP management, CMIP management, or both. Your choice will be dictated by the types of devices used in your network, and the network management protocols that they support.

1.5.1 RPC Support

Solstice Enterprise Manager is shipped with a suite of agents developed for the SunNet Manager network management platform. These agents communicate with a network manager, such as Solstice EM, using Remote Procedure Call (RPC) protocol. When deployed on systems in your network, these RPC agents can be used by Solstice EM as part of your strategy for managing network resources. The resource may be a machine, a component in a machine (such as a router interface card), or some other resource. The RPC agent may be local to or remote from that resource.

As illustrated in Figure 1-9, SNM agents use Remote Procedure Call (RPC) protocol to communicate with the MIS. However, an SNM agent may act as a “proxy” for the management station, using a different management protocol in gathering information from other agents. The RPC Protocol Driver Module (PDM) in the MIS translates requests from management applications, such as Nerve Center requests, into appropriate SNM RPC messages, which it forwards to the RPC agent. RPC responses from the agent are in turn translated from SNM RPC format into the CMIP format used for messages internal to the MIS.

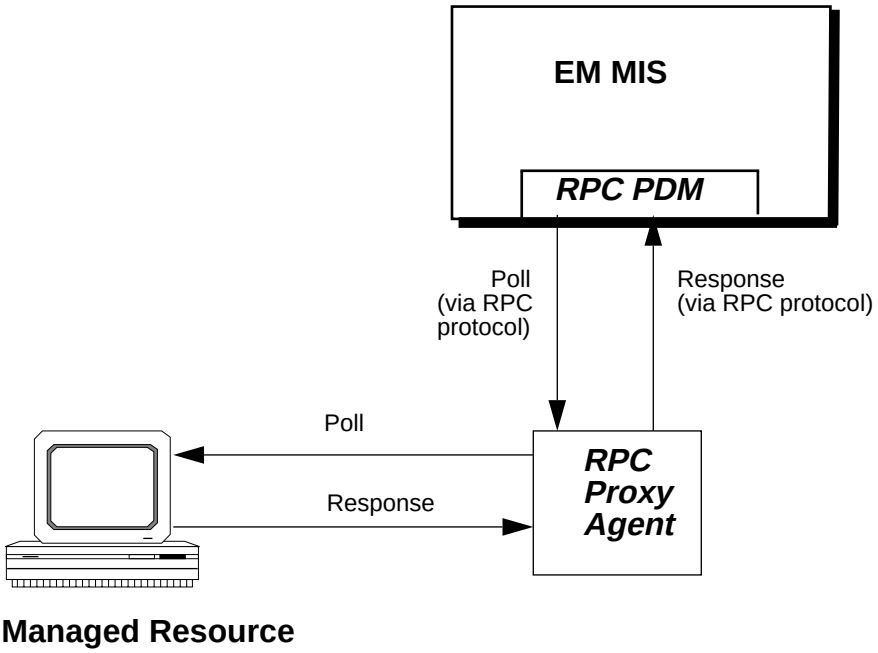


Figure 1-9 Polling RPC Agents

Step by step guidance in using RPC agents as part of your network management solution is provided in Chapter 4, “Device Management Using RPC Agents.”

An important aspect of EM's RPC support is the ability of the EM MIS to offload threshold-checking activity to RPC proxy agents, which may be distributed to various sites around your network.

SunNet Manager RPC agents have the ability to poll managed resources to check for user-configurable thresholds and send an event notification — called an *SNM event* — to a specified management station. This polling activity is initiated by a one-shot message from a management station — called an *SNM event request*. The SNM event request defines the threshold and polling interval for the agent's polling activity. The flow of information using EM's SNM event request capability is illustrated in Figure 1-10. Figure 1-10 illustrates a configuration where the RPC proxy agent is distributed to a machine other than the MIS. The RPC proxy agent may also be located on the MIS machine.

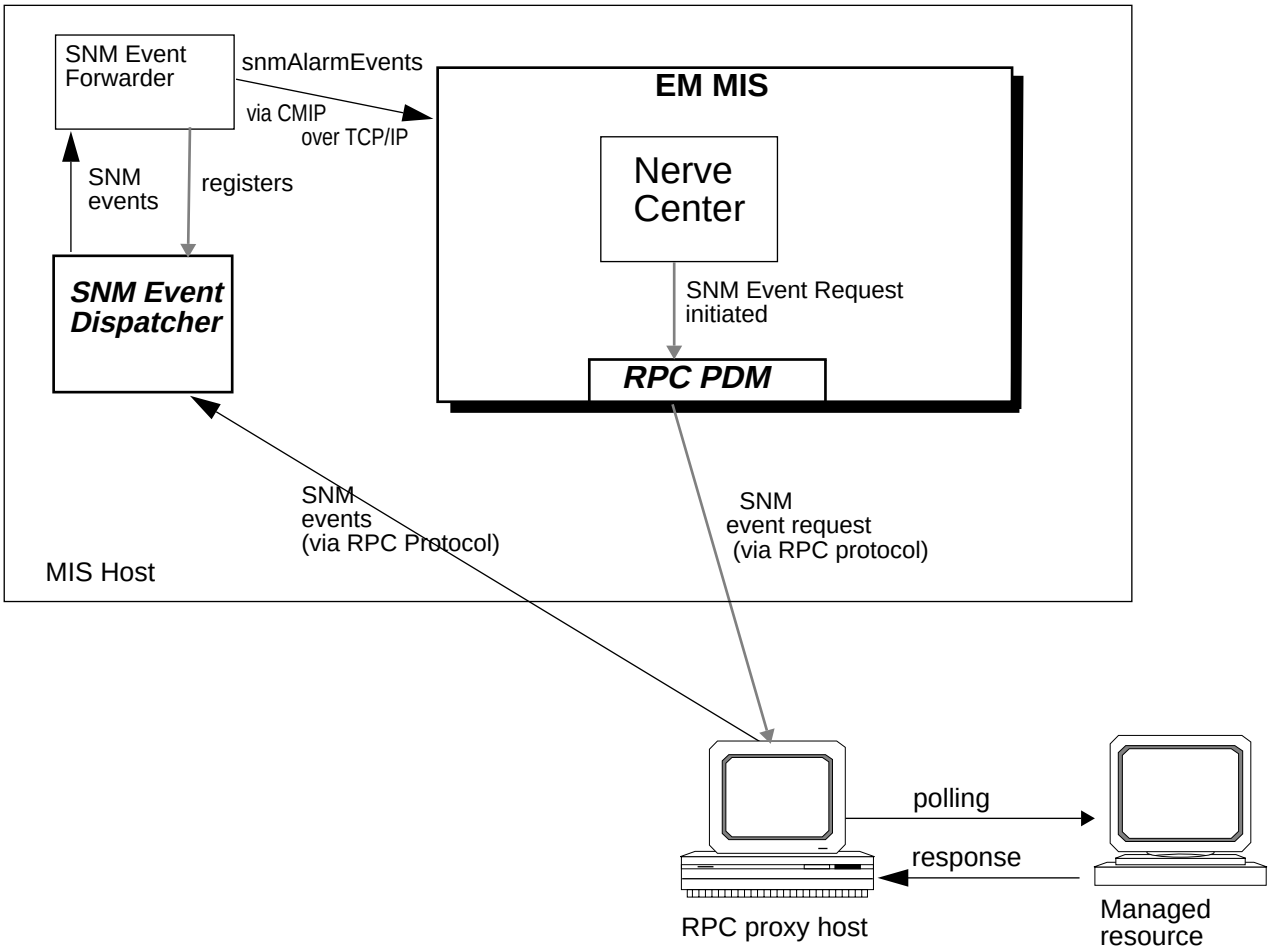


Figure 1-10 Using SNM Event Requests with Solstice EM

The RPC proxy agents, SunNet Manager Event Dispatcher and SNM Event Forwarder are installed on the MIS machine if you select the IP management option (or both CMIP and IP management) during installation. Solstice EM's Request Condition Language (RCL) — a script language used in building Nerve Center request templates — has built-in support for SNM event requests. This capability is described in Chapter 11, "Building Templates for SunNet Manager Event Requests" and Chapter 9, "Nerve Center Overview."

1.5.2 SNMP Support

A key component of EM's SNMP support is the SNMP Protocol Driver Module (PDM) in the MIS. The SNMP PDM translates management requests into an appropriate SNMP message and translates messages from SNMP agents into the internal CMIP format used by the MIS. This is illustrated in Figure 1-11.

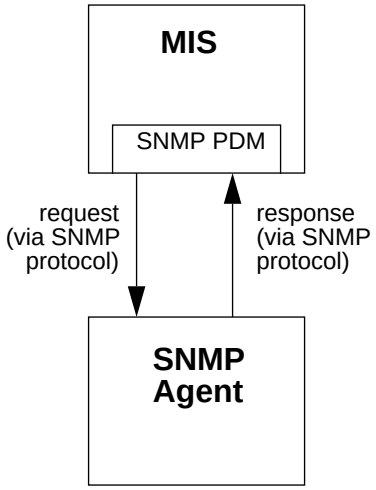


Figure 1-11 MIS Communication with SNMP Agents

For example, if you select a device in the Viewer that is manageable via SNMP, and invoke EM's SNMP Browser, you can retrieve the current values of SNMP attributes or poll for selected attributes. The SNMP Browser, which connects to the MIS, sends requests for data which are translated by the MIS into SNMP requests via the SNMP PDM.

A second important aspect of EM's SNMP support is the EM SNMP trap daemon, which can be distributed to various sites in your network. Simple Network Management Protocol (SNMP) agents have the ability to generate event notifications on their own initiative when certain conditions are detected; these notifications are called *traps*. The EM trap daemon listens for incoming

SNMP traps and converts them to CMIP event notifications for forwarding to one or more MIS. Like other EM applications, the trap daemon uses a reliable CMIP over TCP/IP connection to the MIS.

The trap daemon also has the ability to forward SNMP traps to Site/SunNet/Domain Manager Consoles or other managers. Trap daemon operation is illustrated in Figure 1-12.

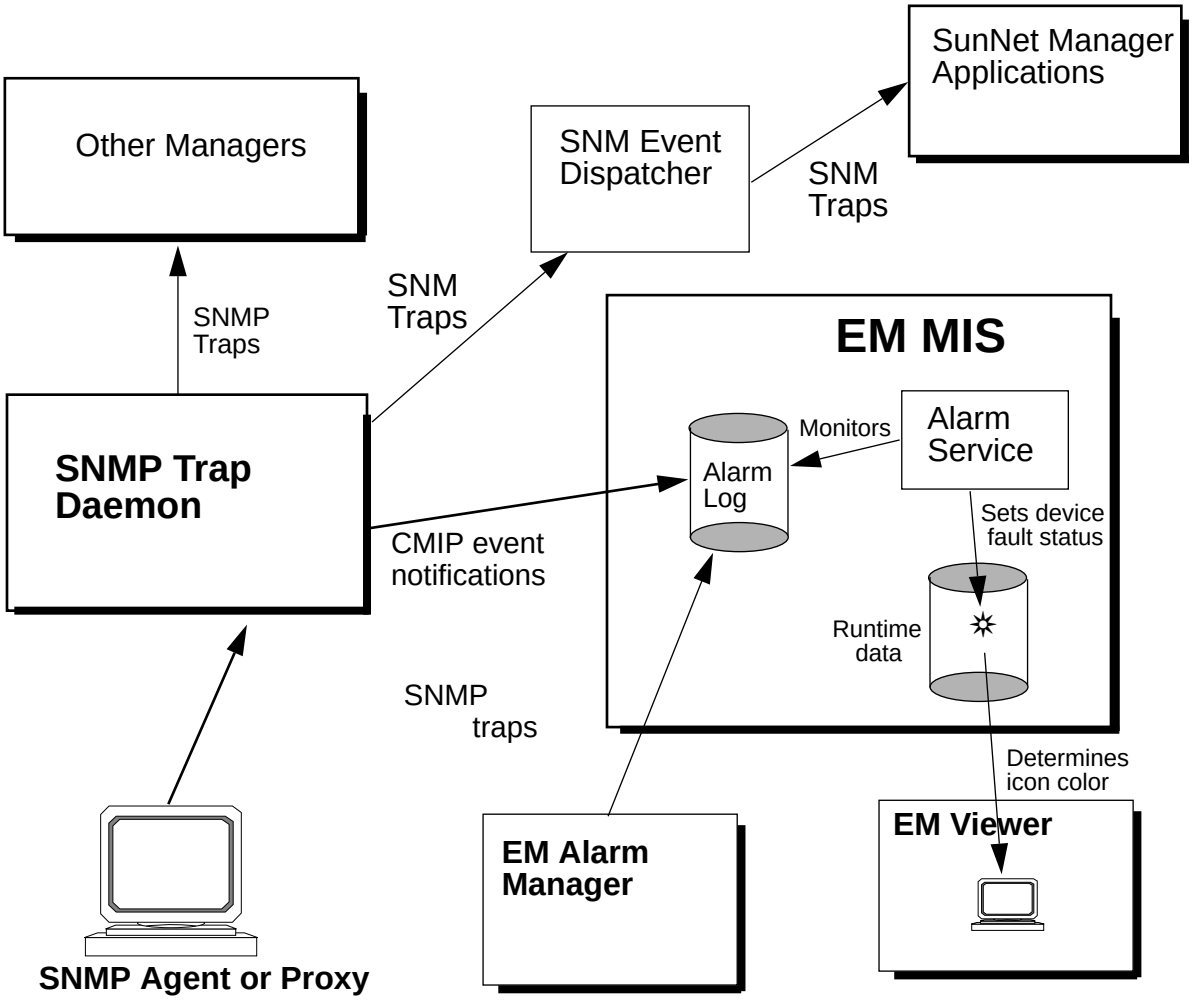


Figure 1-12 SNMP Trap Daemon Operation

The trap daemon has a flexible, user-configurable trap-mapping capability which allows you to customize the conversion of incoming SNMP traps to event notifications to create more meaningful alarms tailored to your network

management needs. How to customize the trap daemon's mapping of SNMP traps is described in Chapter 8, "Mapping SNMP Traps to CMIP Event Notifications."

A default mapping is provided when you install the trap daemon. With this default mapping, a user who invokes the Alarm Manager to examine the alarm log can tell at a glance the types of traps that have been logged against devices in their network, as shown in Figure 1-13.

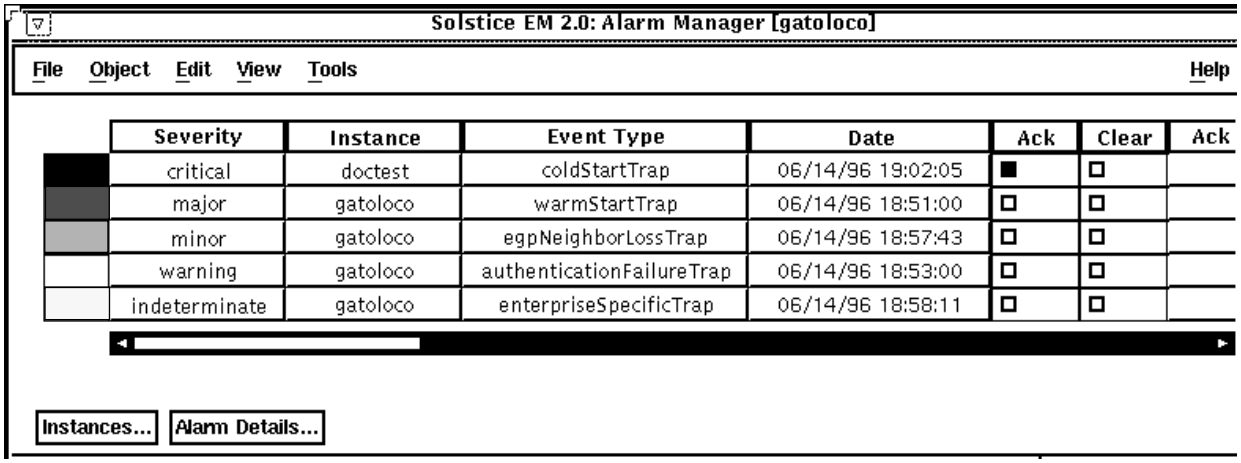


Figure 1-13 Viewing Trap Notifications in the Alarm Manager

The SunNet Manager SNMP proxy agent, shipped with Solstice EM, provides an additional element of SNMP support. Polling of SNMP devices can be offloaded from the MIS to the SNMP proxy agent, using the EM Nerve Center's SNM event request capability. Using its RPC PDM, the MIS communicates with the SNMP proxy agent via RPC protocol (over UDP/IP), and the proxy agent talks to SNMP devices. Figure 1-14 illustrates the use of the SNMP proxy agent for offloaded polling of SNMP devices.

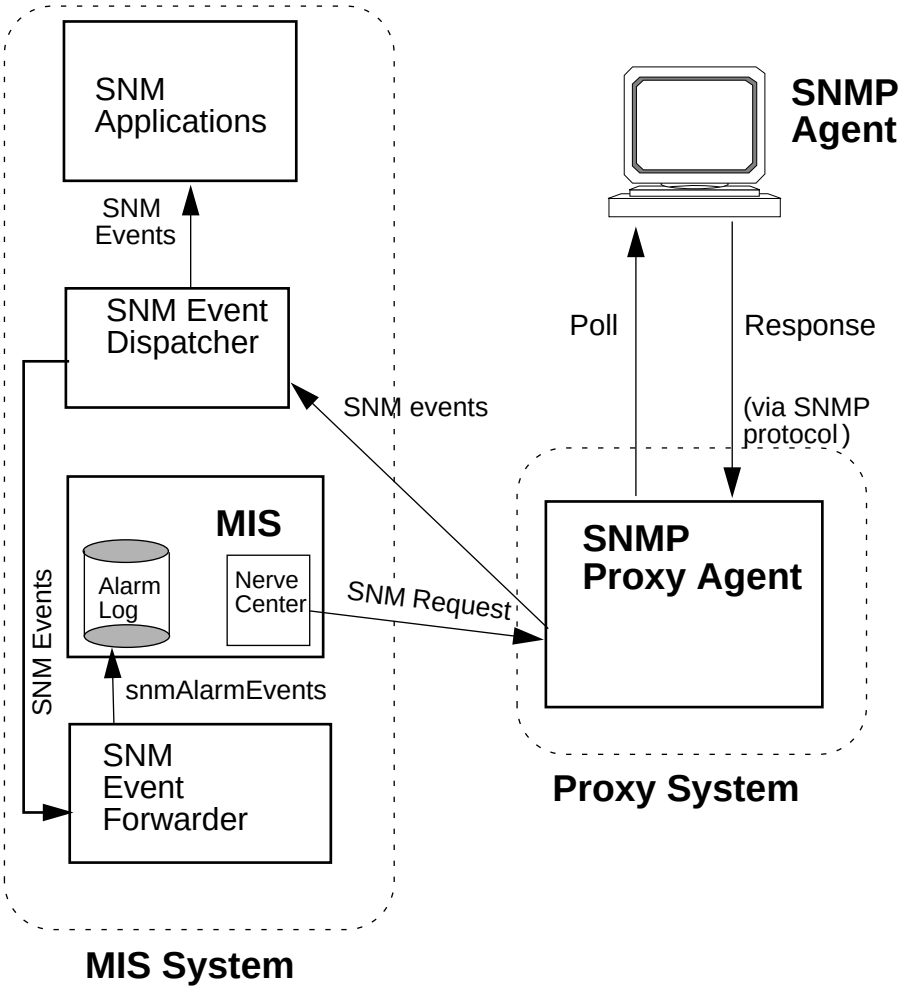


Figure 1-14 SNMP Proxy Agent Operation

1.5.3 CMIP Support

The Solstice EM CMIP Management Protocol Adaptor (MPA) supports communication between the Solstice EM MIS and CMIP agents. The CMIP MPA is installed if you select the CMIP management option (or mixed IP and

CMIP management) during installation. The CMIP MPA may be installed on the same machine as the MIS or it can be distributed to multiple sites. This distributed scenario is illustrated in Figure 1-15. Alternatively, if the MIS is installed on a more powerful server machine, multiple MPAs could be installed on the MIS machine to “fan out” the message-handling load in communications with large numbers of CMIP agents.

The machine on which the MPA is installed must be running SunLink CMIP 8.2. The MPA can be used with SunLink CMIP 8.2 using RFC 1006 (over TCP/IP) or SunLink CMIP 8.2 over SunLink OSI 8.1. This enables communication with conformant CMIP management entities. SunLink CMIP and SunLink OSI are not shipped with Solstice EM.

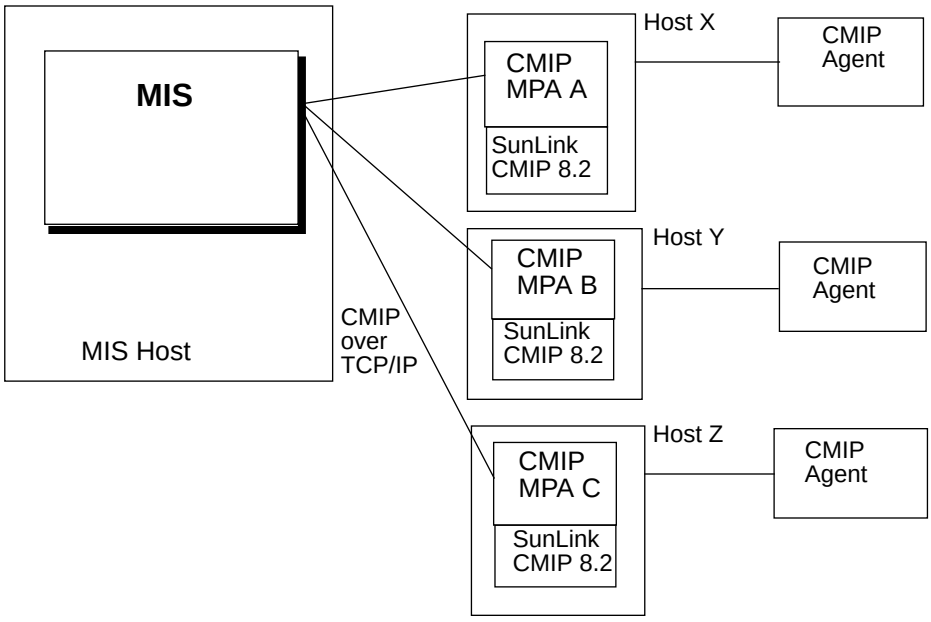


Figure 1-15 CMIP MPAs in Distributed Configuration

For information about configuring CMIP support, refer to Chapter 19, “Configuring Communication with CMIP Agents.”

1.5.3.1 *Telecommunications Management Network*

Solstice EM complies with the Telecommunications Management Network (TMN) standard, an extension of the Open Systems Interconnection (OSI) standards developed through the International Telecommunications Union-Telecommunications Standardization Sector (ITU-T, formerly the CCITT). A Telecommunications Management Network is a network providing surveillance and control over another network. As illustrated in Figure 1-16, EM's CMIP Management Protocol Adaptor (MPA), installed on the MIS machine, can support a TMN Q3 connection to a CMIP agent, which provides access to the managed resources.

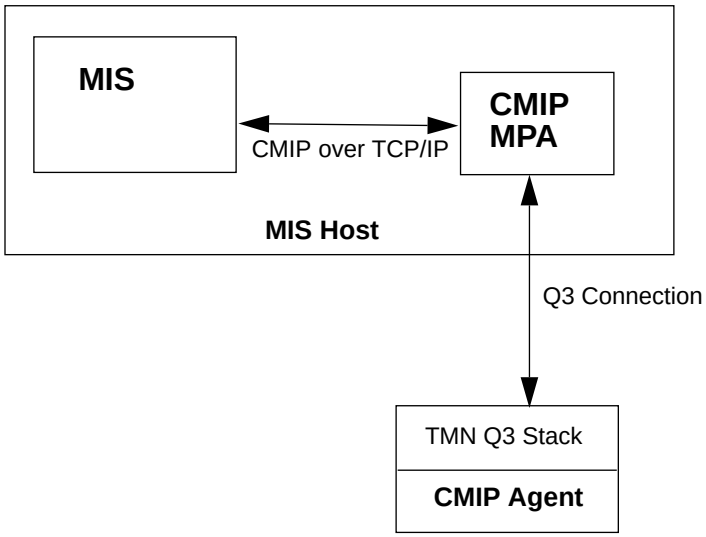


Figure 1-16 TMN Q3 Connection to EM

1.5.4 *Other Network Management Protocols*

Legacy or proprietary network management protocols can be supported by EM through the development of a custom Management Protocol Adaptor (MPA). Third-party developers interested in creating such custom MPAs should consult the "Management Protocol Adaptors" chapter in the *Solstice Enterprise Manager Application Development Guide*.

1.6 Adding New Object Classes and Event Types

The definition language used to represent management information internally in the MIS is the Guidelines for the Definition of Managed Objects (GDMO), outlined in the ITU ISO/IEC 10165-4 standard. This provides the EM management platform with an integrated, standards-based view of all managed resources.

Solstice EM is shipped with a variety of GDMO-defined object classes and event notification types that allow you to perform OSI, SNMP, and RPC network management for most common network elements and topologies. However, the system can be easily extended through the addition of new object classes and event types. All object classes and event types are defined in GDMO documents that have been loaded into the MIS. Solstice EM allows you to create your own GDMO definitions, or to add new GDMO definitions that you have obtained from third-party vendors. Also, Solstice EM is shipped with tools that enable you to convert third-party SNMP MIBs and SNM schemas to GDMO documents.

For more information...

- On adding new event types — See Chapter 15, “Adding New Event Types.”
- On adding new GDMO object class definitions — See Chapter 16, “Adding a Managed Object Class to the MIS.”
- On converting an SNMP Concise MIB to a GDMO document — See Chapter 17, “Adding a MIB to the MIS.”
- On converting an SNM schema to a GDMO document — See Chapter 18, “Adding an Object Class Based on an SNM Schema File to the MIS.”

Populating the MIS 2

Starting Discover from the Application Launcher	*page 2-2*
Using the Viewer to Create a View	*page 2-6*
Creating an Object Instance in the Viewer	*page 2-9*
Updating the Views of Your Network	*page 2-11*
What's Next?	*page 2-15*

This chapter provides an example on how to populate the Management Information Server (MIS) with managed objects, using applications shipped with Solstice EM.

As a Solstice EM user, you add objects to the MIS by:

- Invoking Discover (the primary means)
- Invoking the Object Configuration Tool from the Viewer. For more information, see the "Object Configuration Tool" chapter in the *Solstice Enterprise Manager Reference Manual*, or Section 2.3, "Creating an Object Instance in the Viewer."
- Invoking the Viewer's Object Palette tool. For more information, see the "Viewer" chapter in the *Solstice Enterprise Manager Reference Manual*.

This chapter describes how to populate the MIS by using the Discover application. Specifically, you will proceed through a scenario in which you will:

- Discover all routers up to one hop away
- Place them in a single view for ease of management
- Update the view so that if new routers are found, they are added to the MIS

2.1 Starting Discover from the Application Launcher

Start the Application Launcher with the command shown below.

```
hostname% em &
```

If you need to connect to a remote MIS, specify the name of the host where the MIS is running with the command shown below.

```
hostname% em -host <hostname> &
```

In the Application Launcher, click on the Discover icon. This starts Discover and brings up the main Discover window (Figure 2-1).)

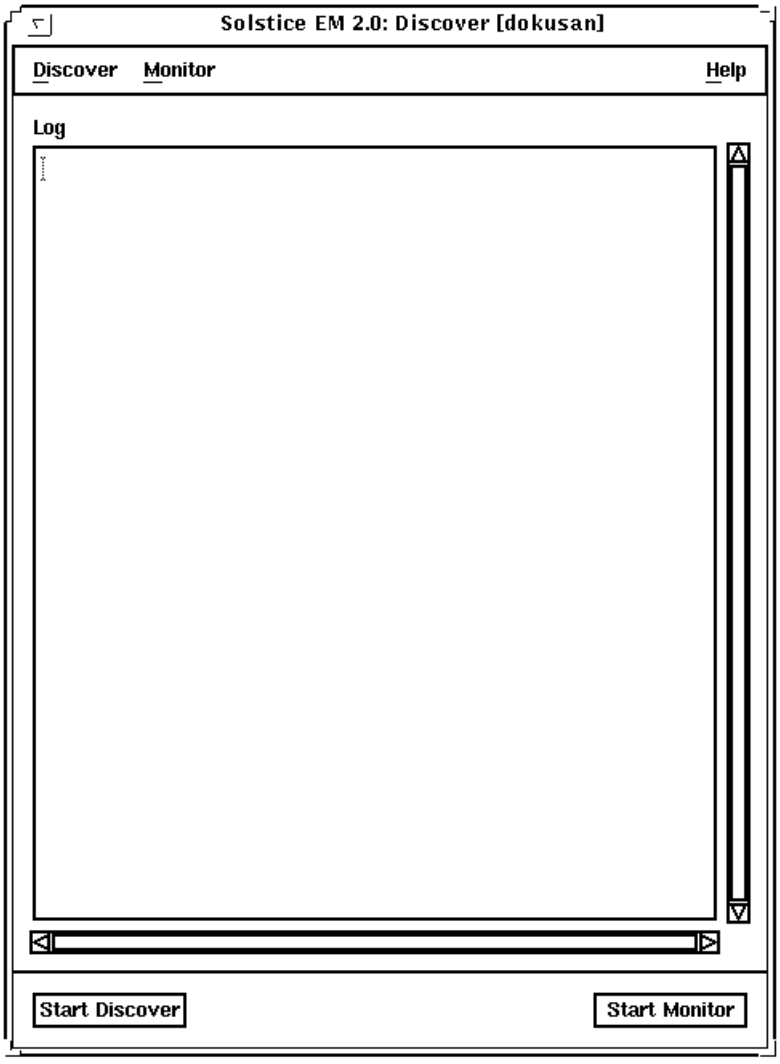


Figure 2-1 Discover Main Window

Starting from the machine on which it is run, (which might be local to or remote from the MIS machine), Discover finds hosts, routers, networks, subnetworks, links, and Simple Network Management Protocol (SNMP) devices. Upon finding one of these network elements, Discover creates an object for that element in the MIS.

When you click on the **Start Discover** button from the Discover main window, Discover probes your network in a way that is determined by the parameters specified in the Discover Properties window (Figure 2-2). This mode of discovery can be as limited or extended as your time and machine resources allow.

By default, Discover finds all objects on the local subnetwork (0 hops away). See the “Discover” chapter in the *Solstice Enterprise Manager Reference Manual* for a discussion of configuration options and of the possible consequences of performing a multiple-hop discover.

Discover configuration options may be set in the Properties window. Select Discover ➤ Properties from the Discover main window to display the Properties window, shown in Figure 2-2.

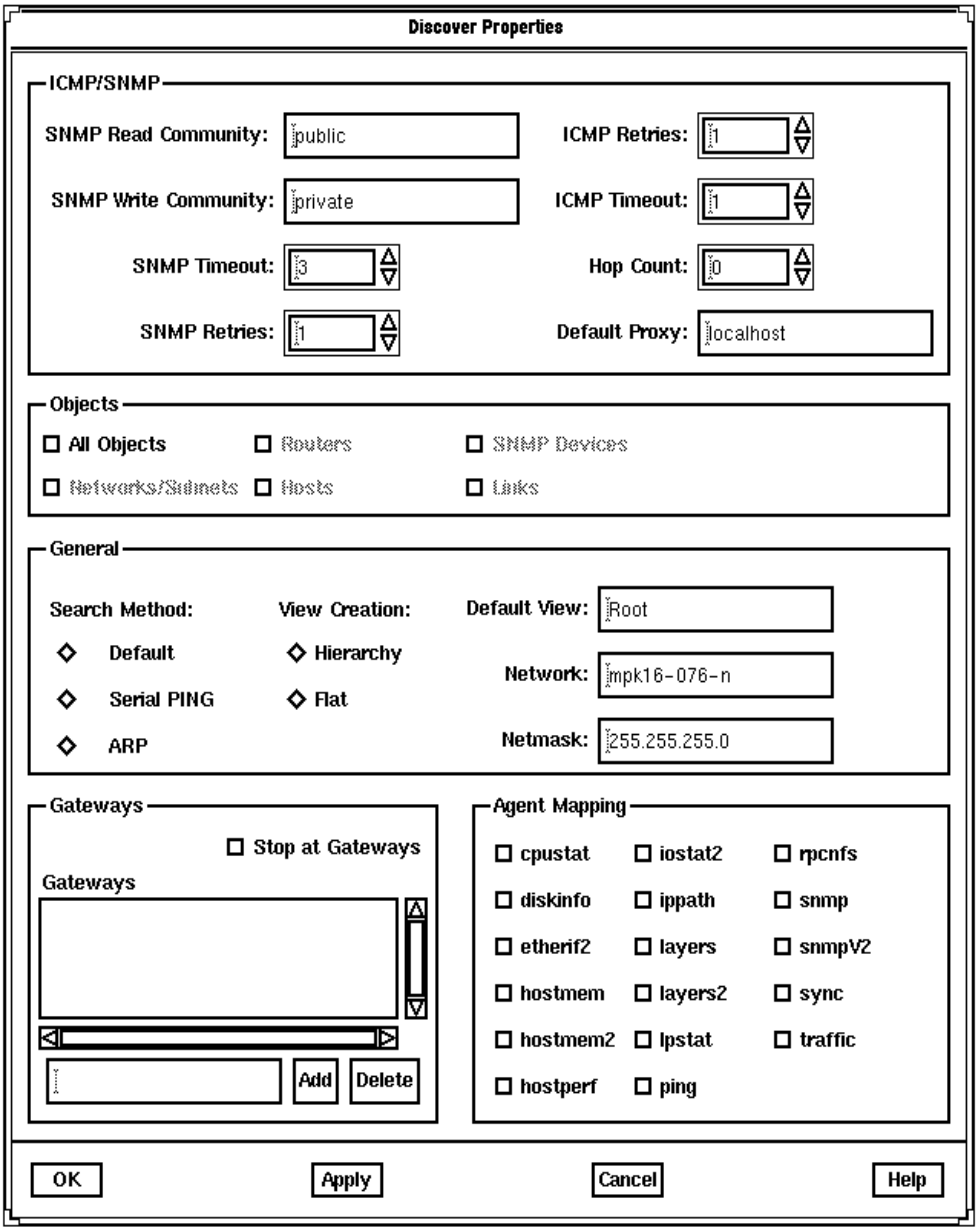


Figure 2-2 Discover Properties Window

The following example illustrates the steps required to perform a discover in which you find all routers no more than one hop from the network on which Discover is running. You then create a view in which you place all the discovered routers. This example represents just one of a number of possibilities for performing a discover. This may not be the discover operation that you want or need to perform for your network.

2.2 Using the Viewer to Create a View

2.2.1 Discovering the Routers

1. **In Discover's Properties window, shown in Figure 2-2, click on the up arrow in the Hop Count window to increase the hop count to 1.**

2. **Click on the All Objects toggle button in the Objects field to deselect that option and activate the other options, then click on the Routers toggle button.**
   Note that selecting Routers automatically activates the Networks/Subnets and Links options.

3. **In the Agent Mapping field, select the desired SunNet Manager (SNM) RPC-based agents from the list.**
   Notice that when you selected Routers, the `ping` agent was automatically selected. If there are other specific SNM RPC-based agents installed on routers in your network, and you want objects to be configured for these agents when they are discovered, select them from this list.

   For more information, see the "Device Management Using RPC Agents" chapter in the *Solstice Enterprise Manager Administration Guide.*

4. **If you want, specify the name of a proxy host in the Default Proxy field.**
   The Default Proxy field allows you to specify the name of a machine you want to act as a proxy and perform the actual network polling. Any machine with the desired RPC-based SunNet Manager (SNM) agent(s) can be specified. The default value for this field is *localhost*, meaning that the polling will be done by the agent on the local machine.

   For more information, see the "Device Management Using RPC Agents" chapter in the *Solstice Enterprise Manager Administration Guide.*

5. **Click on the OK button.**

6. **In the Discover main window, click on the Start Discover button to start the discovery process.**
   Discover reports its progress in the main window (called the Discover Log window). The application first finds the subnetworks and networks within one hop of the machine on which Discover is running, then it finds routers.

To stop the Discover process prior to completion, click on the **Stop Discover** button. Any machines already discovered remain in the MIS.

2.2.2 Creating a Routers View

For ease of management, you can create a view in which all the routers are placed.

1. **Select the Viewer icon in the Application Launcher to start that application.**
   Select the bottom right corner of the Viewer and pull right and down to increase the size of the Viewer window. Use the sash, the small box at the base of the divider between the view navigator window and the Viewer canvas, to adjust the proportions of those two subwindows.

   In the view navigator window, you see subnetworks found by Discover. Activate a subnetwork icon to switch to the view represented by that icon.

2. **In the Viewer window, select Tools ➤ Find.**
   You will receive the Find window (Figure 2-3).

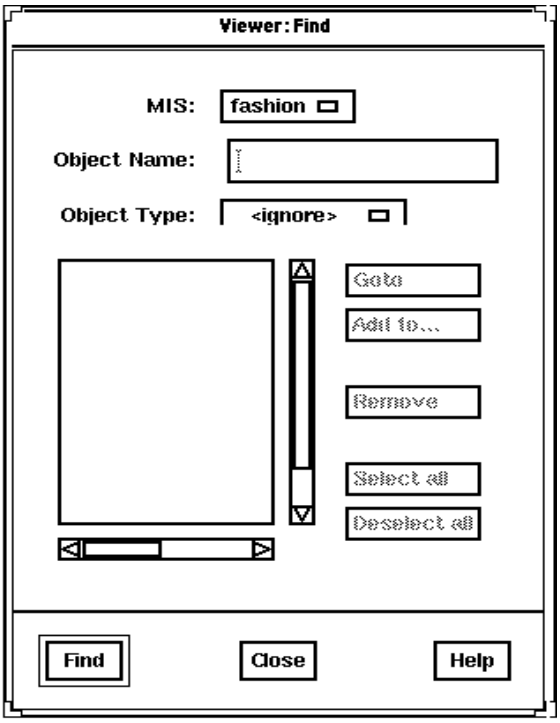


Figure 2-3 Gather Window

3. **In the Find window, click on the button in the Object Type field and select “Router” from the pop-up menu, then click on the Find button.**
   The tool displays all Router objects in the window beneath the Object Type field.

4. **Click on the Add to button.**
   You will receive the Save Results window, as shown in Figure 2-4.

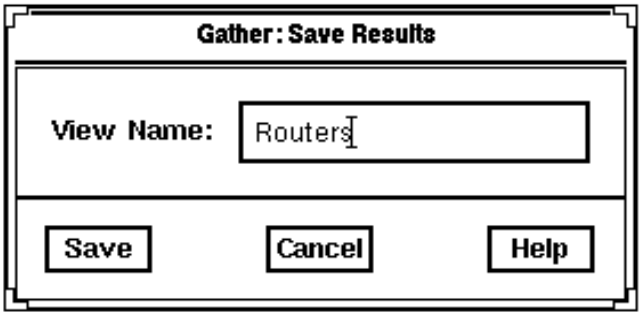


Figure 2-4 Save Results Window

5. **In the Save Results window, enter “Routers”, then click on the Save button.**
   The Viewer gathers the just-found routers into a new logical view called “Routers”, which appears in the view navigator window.

6. **In the view navigator window, click on the new Routers icon to switch to that view.**
   In the Viewer canvas, you will see the router icons you originally saw in the Find window.

2.3 Creating an Object Instance in the Viewer

The Viewer offers you several different methods for adding object instances and, thus, further populating your MIS. One method is to use the options available in the Viewer’s Object pull-down menu, while another is through the use of the Object Palette, which you invoke using the Viewer’s Tool pull-down menu.

There is also a third creation method, which applies only to the creation of container objects, such as subnetworks or networks. An example of this method was shown in Section 2.2.2, “Creating a Routers View,” when you selected Tools ➤ Find from the Viewer and created a new instance of a container object called “Routers.”

2.3.1 Using the Viewer’s Object Menu

The Viewer’s Object pull-down menu contains the following options:

New Container Option

Select Object ➤ New Container to configure the following objects:

- Container
- Subnetwork
- Network
- Universe

New Monitor Option

Select Object ➤ New Monitor to configure the following objects:

- Hexagon
- Hexagon120
- OmniSector
- Circle

New Device Option

Select Object ➤ New Device to configure the following objects:

- Device
- Bus
- Router
- Bridge
- Hub
- Host
- Server
- Interface
- Pc
- Sunws
- Printer

New Link Option

Select Object ➤ New Link to configure the following object:

- Link

Selecting any item under the New Container, New Device, or New Monitor pull-right menus causes the Object Configuration Tool (OCT) window to be displayed. At the very minimum, the name of the managed object, its type, and

the managed protocol that is to be used with it should be specified. In addition to these three pieces of information, any other optional configuration information may be specified.

If you create any object under the New Container or New Monitor pull-right menus, an icon for that object shows up in the view navigator window and also in the Viewer canvas.

When you select Link from the New Link pull-right menu, you are prompted to click on the object you are linking from and then on the object you are linking to. After doing this, the OCT window appears.

Each icon in the Viewer canvas has an available pop-up menu, which you can see by positioning the mouse cursor over the icon and clicking the third mouse button. To bring up the OCT window, select the Object Properties option from this menu. Alternatively, you can double-click on any icon to bring up the OCT window.

2.3.2 Using the Object Palette

Select Tools ➤ Object Palette from the Viewer's menu to invoke the Object Palette. From the Object Palette window you can drag-and-drop an object to the Viewer canvas (using the middle mouse button). Alternatively, you can select a point within the Viewer canvas, then select an object in the Object Palette window.

After either dragging and dropping or pointing and selecting an object from the Object Palette window to the Viewer canvas, the OCT window will appear. After configuring the new object and clicking on the **OK** button in the Object Configuration Tool, the object will appear in the Viewer canvas.

2.4 Updating the Views of Your Network

While the Discover function provides you with an easy way of adding objects to the MIS, The Monitor function provides a convenient means of regularly *updating* the MIS topology information. Monitor compares the current MIS network topology to network resources that it finds while it is active. Monitor can be used to find devices that have been added to the network since Discover was last run.

In this scenario, you will start the Monitor function to update the Routers view which you just created. Any new routers, networks, subnets, and/or links that are found since the last time Discover was run will be added to the MIS.

1. **In the Discover main window, select Monitor ➤ Properties to display the Monitor Properties window, shown in Figure 2-5.**
   The Monitor Properties window allows you to configure the Monitor function. For more information, see the "Discover" chapter in the *Solstice Enterprise Manager Reference Manual.*

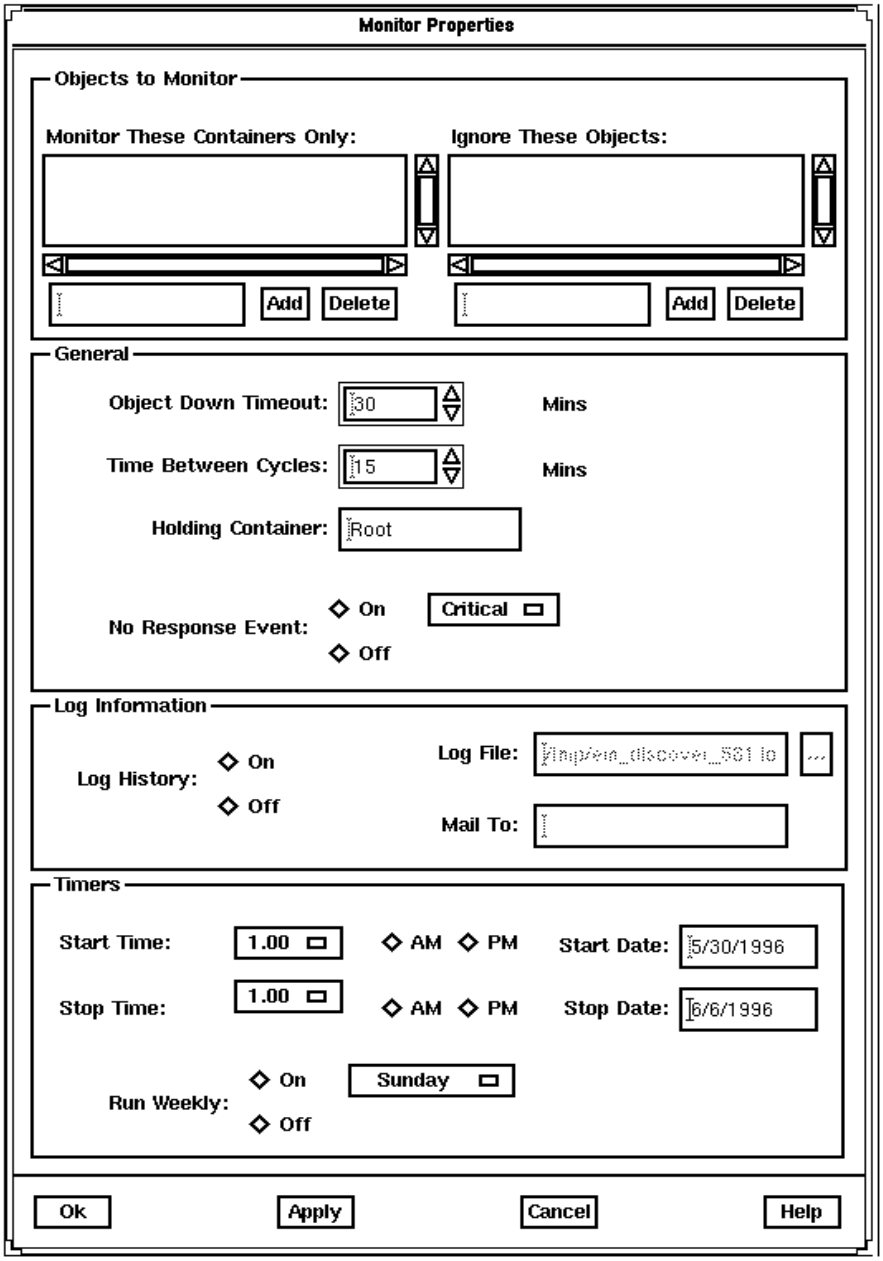


Figure 2-5 Monitor Properties Window

2. **In the Holding Container field, specify the name of the container in which you want to place any newly discovered objects.**
   The default is Root, which means that newly discovered objects are placed in the MIS as they are found, preserving the network topology. If you want to place all newly discovered objects in a specific container, specify the container name in this field.

3. **If you want, turn the Log History feature on and specify a name and electronic mail recipient for the log file.**
   For more information, see the "Discover" chapter in the *Solstice Enterprise Manager Reference Manual.*

4. **Click on the button in the Start Time field, select 12:00 from the resulting pop-up menu, and click on the AM toggle button.**
   This instructs monitor to begin updating your topology database at midnight.

5. **Change the date in the Date field to tomorrow's date.**
   By default, the date in this field is today's date.

6. **Click on the button in the Stop Time field, select 5:00 from the resulting pop-up menu, and click on the AM toggle button.**
   This instructs monitor to stop updating your topology database at 5:00 a.m. tomorrow morning.

7. **Change the date in the Stop Date field to tomorrow's date.**
   By default, the date in this field is one week from today's date.

8. **Click on the OK button.**

9. **In the Discover main window, click on the Start Monitor button.**
   Monitor will reports its progress in the Discover Log window.

If you turned the log history feature on, and specified a mailing address, the log will be mailed electronically to the specified address at 5:00 a.m. tomorrow, when you instructed the Monitor process to stop. To stop the Monitor process prior to completion, click on the **Stop Monitor** button.

2.5 What's Next?

By using Discover or the tools in the Viewer, you can create a database of objects to be managed. To be able to manage the objects in your network the objects in the database must be configured to indicate what management capabilities these objects support — such as CMIP, SNMP, or SunNet Manager agents. There are two ways this can be done:

- If you use Discover to populate your database, Discover will automatically configure objects for SNMP capability if it has uncovered this in the discovery process. You can also select the RPC agents for which you want discovered objects to be configured (`ping` is the default RPC agent).
- If you have added objects to the database through the one-at-a-time methods provided in the Viewer, or if you want to specify SunNet Manager RPC agents, then you must use the Object Configuration Tool to specify CMIP, SNMP, or SunNet Manager management capabilities for these objects. See the "Object Configuration Tool" chapter in the *Solstice Enterprise Manager Reference Manual* for a complete description of the Object Configuration Tool's capabilities.

Following OCT configuration (if needed), you can use Nerve Center requests and related applications (see the example in Chapter 3, "Managing Devices") and the Data Viewer to manage your network.

Managing Devices 3

Fault Management	*page 3-2*
Monitoring Fault Status of Devices	*page 3-19*
Performance Management	*page 3-29*

This chapter is intended to provide you with some ideas about how you can use Solstice Enterprise Manager to accomplish your network management goals. The methods and scenarios described here are not the only ways those goals could be met. The approach that is best suited for a given situation will depend on the particular network configuration, available network management applications, and network management priorities.

This chapter assumes you have already populated your MIS, through a combination of the use of Discover, to add multiple managed objects automatically, and through use of various Viewer-related tools, to add managed objects one-by-one. See Chapter 2, “Populating the MIS,” for examples of how you can add managed objects to your MIS. We also assume that the objects representing your network have been configured according to the network management protocol and agents they support — CMIP, SNMP, or SunNet Manager (SNM) RPC. (This is done either via the Discover process or one-at-a-time using the Object Configuration tool.)

3

3.1 Fault Management

Fault management is the tracking and managing of critical events on your network. For example, if one of your critical network resources — such as a server, link, or key application — becomes inoperative or unavailable to users, you will want to be notified of this immediately.

Three key tools provided by Solstice EM for tracking fault status are the Viewer, Alarm Manager, and Log Manager. Two important Solstice EM components that can provide you with information about critical network events are Nerve Center requests, which are launched from the Viewer, and the Solstice EM SNMP trap daemon, which listens for traps generated by SNMP agents. A number of the options available to you in setting up Solstice EM for tracking fault status of devices on your network are described in this section. The steps in monitoring device status are described below in Section 3.2, “Monitoring Fault Status of Devices.”

3.1.1 Color-Coding of Fault Status in the Viewer

The Viewer provides a window into your network that is continuously updated with the latest fault status information. Fault status is indicated by an icon changing color. The fault status of an object reflects the incoming alarms posted against that object.

Alarms differ in their *severity*. The severity of an event is a rating used to represent the importance or impact of the event. For example, you might regard an event indicating high router network memory usage as less severe than an event indicating that the router is not working at all.

Solstice EM provides six severities; by default, these are color-coded as indicated in Table 3-1.

Table 3-1 Default Color-Coding of Severities

Integer Value	Severity	Default Color
1	Critical	Red
2	Major	Orange
3	Minor	Cyan
4	Warning	Yellow
5	Cleared	No color
0	Indeterminate	Blue

The same color-coding of severities is used in the Alarm Manager — a Solstice EM application that enables you to selectively view, acknowledge, and clear alarms. (You can change the color-coding of severities through the Request Designer application. Refer to Chapter 12, “Request Designer” in this guide. The Alarm Manager is described in the “Alarm Manager” chapter in the *Solstice Enterprise Manager Reference Manual*.)

3.1.1.1 Propagating Changes in Fault Status in the Viewer

The Viewer’s display of the fault status of an object can be “propagated” to objects that “contain” it. (Propagation is off by default.) If you have turned on fault status propagation, a “cloud” icon that represents a subnet changes color to indicate the highest severity fault of any device in that subnet. Similarly, with state change propagation turned on, if you have the Viewer reduced to an icon on your computer screen, the icon representing the Viewer changes color to indicate the highest severity fault status of any device represented in the Viewer. (For information on state change propagation, refer to the “Viewer” chapter in the *Solstice Enterprise Manager Reference Manual*.)

3.1.1.2 The Role of the Alarm Service

The fault status of objects displayed in the Viewer and Alarm Manager is controlled by the Alarm Service, which is a part of the MIS. The Alarm Service monitors incoming alarms posted to the alarm log and updates the fault status of objects to match the highest severity amongst the outstanding (uncleared) alarms posted against that object. If the alarm log receives four minor alarms

and one critical alarm against router `sledge`, `sledge`'s icon is changed to red to reflect the critical alarm. If the critical alarm is cleared, the icon changes to cyan. If all the alarms are cleared or purged, the icon has no status coloring — indicating that the state of the device is "normal."

As shipped to you, the Alarm Services monitors a log called `AlarmLog`. When alarms are logged to this log, they automatically affect the icon color in the Viewer.

3.1.1.3 The Log Manager and Alarm Logging

The Log Manager is the application that you use to create logs to store incoming event notifications, and to define which events are stored in which logs.

The particular event types that are selected for logging to the AlarmLog is determined by a Common Management Information Service (CMIS) filter, called a *discriminator construct*. You use the Log Manager to add or subtract event types to the `AlarmLog` by editing the `AlarmLog`'s discriminator construct. (An example that illustrates how to do this is described in Section 3.1.3.4, "Creating a Separate Log for Enterprise-Specific Trap Notifications.")

Actions of the Alarm Manager also affect fault indication in the Viewer. If a network administrator uses the Alarm Manager to clear all the outstanding alarms against router `sledge`, the Alarm Service changes `sledge`'s fault status to `cleared`, and the Viewer icon changes color accordingly. Thus, the Alarm Service ensures that the Viewer and Alarm Manager have the same picture of the fault status of the network resources you are managing.

The types of events that you will want the Alarm Service to monitor (thus updating the color of Viewer icons automatically) depends upon the types of network events you want to track and the management protocols you are using (as discussed in Chapter 1, "Planning for Network Management").

For example, SunNet Manager RPC agents (shipped with EM) have the ability to poll managed resources to check for predefined thresholds and send an event notification — called an *SNM event* — to the management station. This polling activity can be initiated by a one-shot message — called an *SNM event request*. SNM event requests can be initiated from the MIS by Nerve Center requests. (Using Nerve Center requests to initiate threshold-checking by RPC agents, is described in Chapter 11, "Building Templates for SunNet Manager

Event Requests.") When an RPC agent generates an SNM event in response to threshold-checking initiated by the MIS, this arrives at the MIS as an snmAlarmEvent. There are two ways in which you might use these events:

- The Nerve Center request that initiated the RPC agent threshold-checking could *subscribe* for incoming snmAlarmEvents from the target device and take appropriate action in response, such as logging nerveCenterAlarms. nerveCenterAlarms are alarms created by Nerve Center requests using alarm-logging functions that can be inserted in request templates. Two SNM event request templates shipped with Solstice EM, `CheckCPU` and `DeviceReachablePing`, use this method of handling snmAlarmEvents. To avoid duplication of alarms, therefore, snmAlarmEvents are, by default, not logged to the AlarmLog. You may want to use the Request Designer application to examine these request templates to see how they handle SNM event requests.
- Alternatively, the AlarmLog could be configured to automatically log incoming snmAlarmEvents. If you want to implement this, you can use the Log Manager to remove the entry for snmAlarmEvents from the default discriminator construct for the AlarmLog. The default log discriminator only specifies the types of events that are to be *excluded* from the AlarmLog. Any incoming event not explicitly excluded is logged automatically.

Even if you do not want snmAlarmEvents posted to the AlarmLog, you might create a special log, `SNMLog`, to retain an historical record of incoming `snmAlarmEvents`. You can use the Log Viewer to examine the contents of event logs.

For more information ...

The Viewer, Alarm Service, Alarm Manager, Log Manager, and Log Viewer each have a separate chapter devoted to them in the *Solstice Enterprise Manager Reference Manual*.

3.1.2 Receiving Event Information

Information about changes in network resources are reported by agents. There are two types of event information that agents provide:

- **Responses to polls** — Managers can request attributes of managed objects at periodic intervals; this is called *polling*.

- **Event notifications** — Agents also typically have the ability to generate messages on their own initiative when they detect events on a resource the agent is responsible for; these messages are called *event notifications.*

3.1.2.1 Polling

There are two types of polling. Polling can be done directly by the Nerve Center module in the MIS, or SunNet Manager event requests can be used to offload polling to Remote Procedure Call (RPC) proxy agents. Fault management strategies that rely on event notifications and indirect polling by proxy agents are most efficient than direct polling for managing large numbers of devices because such strategies minimize network traffic and MIS processing load. (Offloading of polling to RPC agents is described in Chapter 11, “Building Templates for SunNet Manager Event Requests.’)

You can deploy fault management strategies based on logging of incoming event notifications, direct polling by the Nerve Center, or threshold-checking by RPC proxy agents; or you can develop strategies that use a combination of these. Fault management scenarios that illustrate some of the possibilities are described below.

Solstice EM is shipped with a number of Nerve Center request templates which you may find helpful in developing your fault management strategy. You may find these templates useful as is, or you might modify them to better fit your network management needs.

Note – If you want to monitor large numbers of devices (for example, more than 500) for reachability, the most efficient way to do this is to activate the Solstice EM Auto Manager. For information on Solstice EM’s automatic management capability, refer to the “Automatic Management” chapter in the *Solstice Enterprise Manager Reference Manual.*

3.1.2.2 Using Discover to Monitor Device Availability

EM’s Discover application provides a form of polling for device status that does not require the use of Nerve Center requests.

The main purpose of Discover’s Monitor function is the updating of the representation of your network’s topology in the MIS. Monitor uses Internet protocols, such as SNMP and Internet Control Message Protocol (ICMP), to

probe for devices that have been added to the network since Discover was last run. Monitor compares the existing topology in the MIS to the results of its searches and adds objects to the MIS if new devices are uncovered.

Monitor can also be configured to query all links and interfaces represented in the MIS and generate CMIP communicationsAlarms if these network resources are not available. You can also select the severity that you want to attach to the alarms that would be generated. CMIP communicationsAlarms are logged to the AlarmLog by default when they arrive. Alarm Service monitors the AlarmLog and sets fault status indication to reflect the highest severity alarm currently posted against an object. For example, if you select critical as the severity for Monitor "No Response" alarms, the device icon changes to red in the Viewer when the alarm is logged to the AlarmLog.

If Monitor finds that a previously downed interface or link has become available, it posts a communicationsAlarm with a severity of cleared against the object. The Alarm Service changes fault status indication to reflect this, and icon color in the Viewer changes accordingly.

By default, Monitor's "No Response" event generation capability is turned off. To activate this capability, do the following:

1. **Invoke the Discover application from the Launcher if it is not currently running. Select the Discover➤Monitor Properties... option to invoke the Monitor properties window.**

2. **Select On for the No Response Event option and select a severity from the pulldown menu (shown in Figure 3-1).**

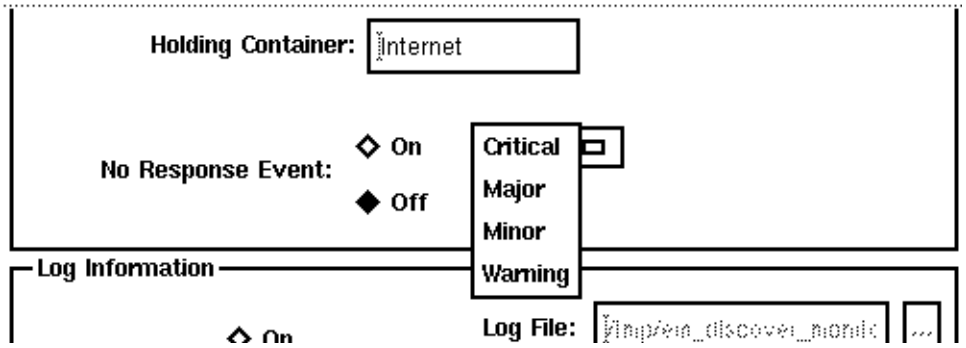


Figure 3-1 Selecting a Severity for communicationsAlarm Generated by Monitor

3. **Select the time of day and days of the week when you want Monitor to be active. Click OK to make your choices take effect.**

The Discover application is described in the “Discover” chapter in the *Solstice Enterprise Manager Reference Manual*

3.1.3 Event Notifications

There are two ways in which event notifications can be used in fault management:

- Automatic monitoring of incoming events by the Alarm Service
- Event correlation and processing by Nerve Center requests

Several types of event notifications are, by default, automatically logged to the AlarmLog when they arrive at the MIS. When these events arrive, icon color in the Viewer is dynamically changed to reflect the severity of the alarms.

3.1.3.1 Example: Monitoring Event Notifications from CMIP Agents

In this scenario XYZ Communications Corp. is using Solstice EM to manage a cellular network. The vendor for their network components has provided AwesomeCell CMIP agents to manage switches and other network elements.The agents can be configured to generate OSI alarms, such as environmentalAlarms and communicationsAlarms, when specified thresholds are crossed. This configuration is illustrated in Figure 3-2.

When, for example, a failure occurs in a relay, the agent generates an `environmentalAlarm` with a severity of critical. The alarm is logged to the AlarmLog and the icon for the device is colored red automatically. No Nerve Center request or polling of the agent was necessary.

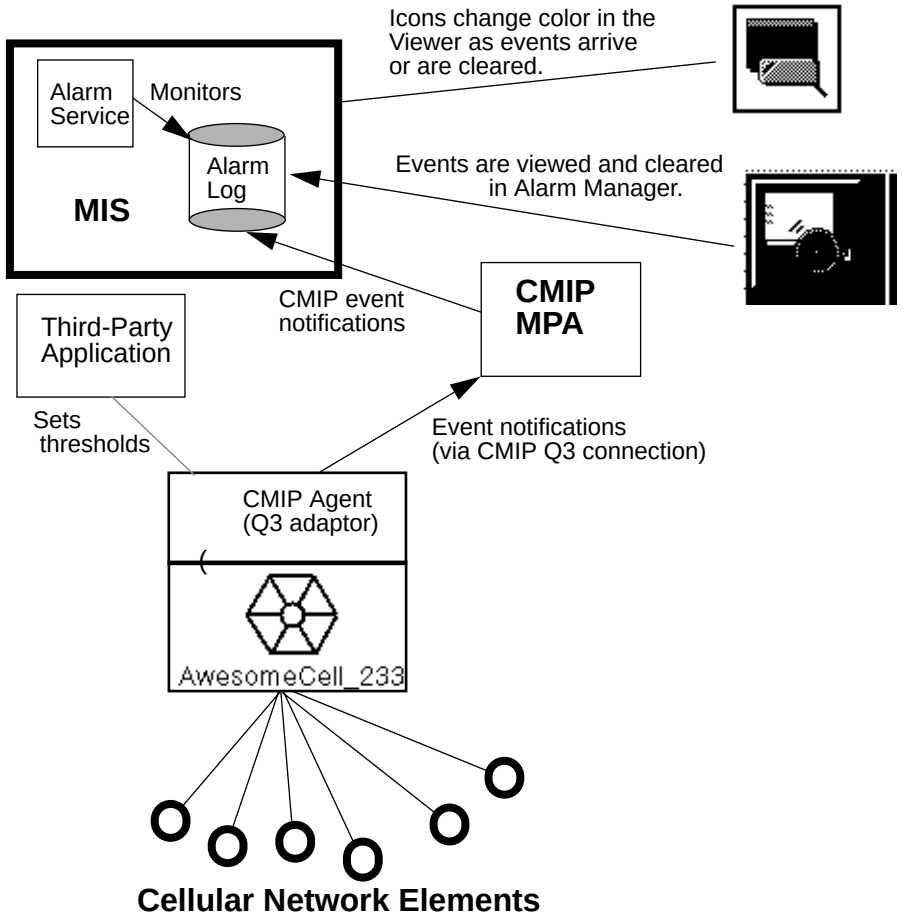


Figure 3-2 CMIP Management of a Cellular Network

3.1.3.2 Using SNMP Traps

Simple Network Management Protocol (SNMP) agents also have the ability to generate event notifications on their own initiative; these messages are called *traps*. CMIP protocol is used by Solstice EM internally to represent all network management event information. Accordingly, Solstice EM's SNMP trap daemon (`em_snmp-trap`) converts incoming SNMP traps to CMIP event notifications and sends them to the MIS.

By default, the trap daemon converts SNMP traps into event notifications as indicated in Table 3-2.

Table 3-2 Default SNMP Trap Notifications and Severities

SNMP Trap	Notification Name	Default Severity
cold Start	coldStartTrap	critical
warmStart	warmStartTrap	major
linkDown	linkDownTrap	major
linkUp	linkUpTrap	clear
authenticationFailure	authenticationFailureTrap	warning
egpNeighborLoss	egpNeighborLossTrap	minor
enterpriseSpecific	enterpriseSpecificTrap	indeterminate

These notifications are, by default, sent to the AlarmLog when they arrive. When you open the Alarm Manager, you can tell at a glance the types of traps that have been logged against devices in your network, as shown in Figure 3-3.

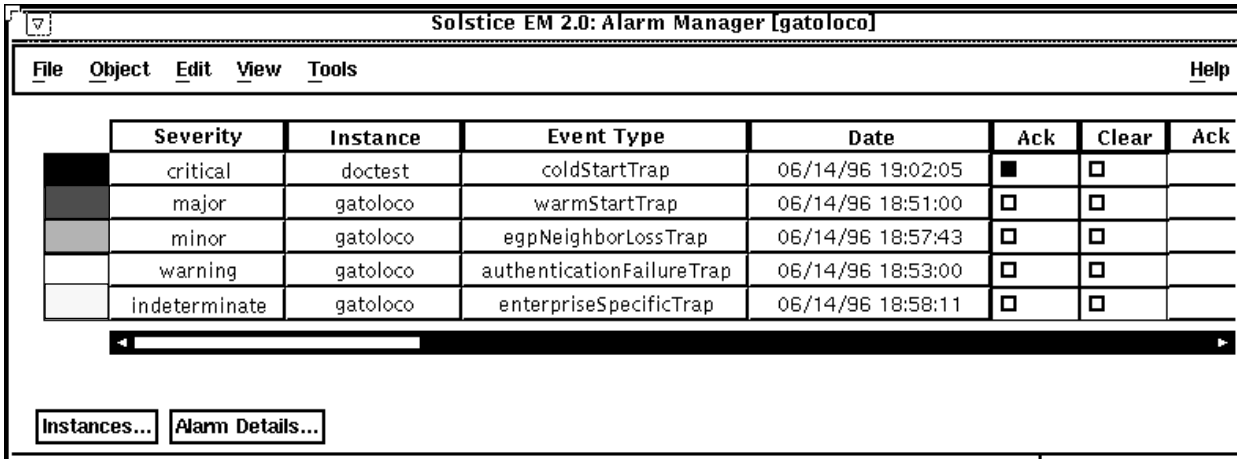


Figure 3-3 Viewing Trap Notifications in the Alarm Manager

SNMP trap daemon operation is illustrated in Figure 3-4. The SNMP trap daemon's mapping of SNMP traps into event notifications can be customized to create alarms that are tailored to your particular network management

needs. For example, you can customize the severities that attach to trap notifications or create custom mappings for enterprise-specific traps based on the enterprise identifier and the specific trap type.

The trap mapping capability also allows you to more finely pinpoint the element that is the source of the alarm. You might want to represent the interface cards in a router with separate icons. You could configure the trap daemon to convert router linkDown and linkUp traps to communicationsAlarms targeted to the responsible interface. The interface icons would change color to pinpoint problems to the level of the individual interface. (Customizing the trap daemon's trap-to-event notification mapping is described in Chapter 8, "Mapping SNMP Traps to CMIP Event Notifications.")

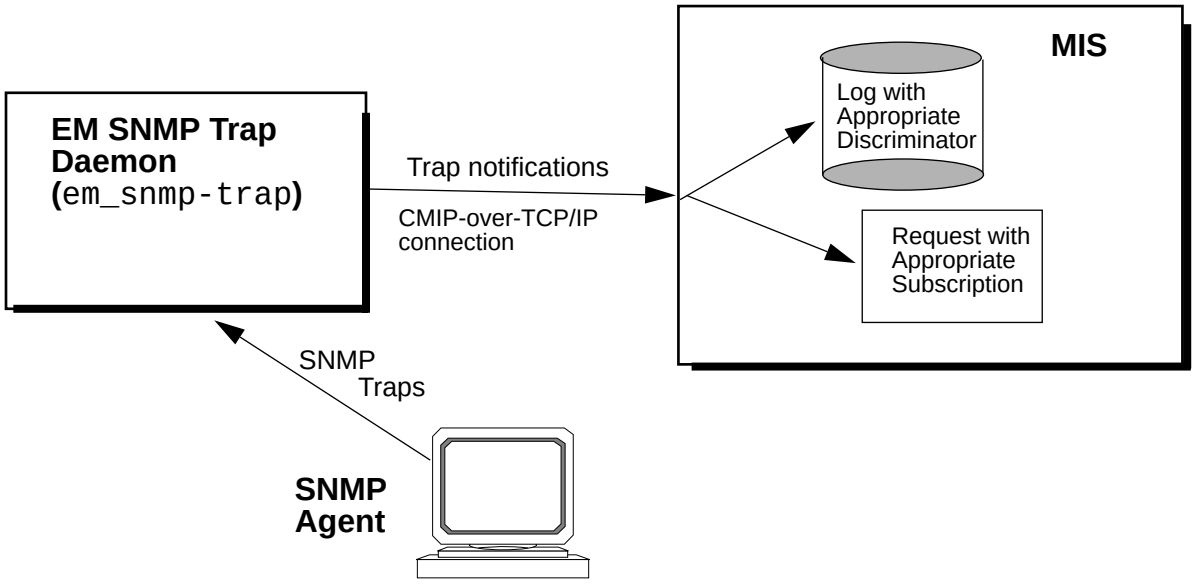


Figure 3-4 Solstice EM Processing of SNMP Traps

3.1.3.3 Monitoring SNMP Traps with Nerve Center Requests

Nerve Center requests can be designed to receive a specified type of event notification, or events from a selected object; this is called *event subscription*. The request enters a subscription with the MIS to receive the specified events as they arrive. A request can subscribe for any type of event notification that has been defined in the MIS.

Event subscription requests can be used to customize your handling of incoming SNMP traps. The sample template `SnmpLinkUpDownTrap`, shipped with Solstice EM, illustrates this possibility. If you launch the SnmpLinkUpDownTrap request at a target router in the Viewer, the request subscribes for incoming linkDown traps from the target device. If a linkDownTrap notification arrives, the request terminates the subscription for linkDown traps and initiates a subscription for linkUp traps. If a matching linkUp trap does not arrive from the target device within a specified polling interval, the request transitions to the Down state and logs a nerveCenterAlarm with a severity of critical. Because the critical alarm is a higher severity than the major severity of the linkDown trap, the Alarm Service sets the fault status of the device to critical and the device icon turns red.

Figure 3-5 shows the flow of information from traps to logs using the SnmpLinkUpDownTrap request.

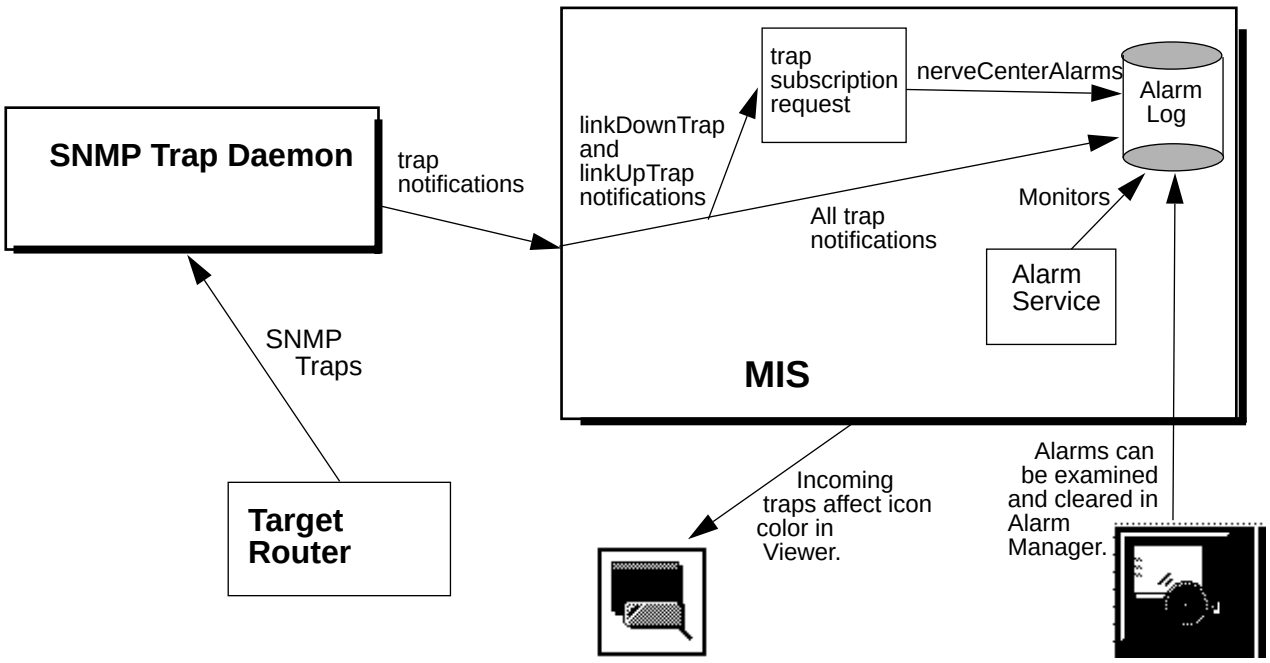


Figure 3-5 Example: SNMP Trap Handling Using SnmpLinkUpDownTrap Request

3.1.3.4 Creating a Separate Log for Enterprise-Specific Trap Notifications

Because enterprise-specific traps may have a variety of possible causes, the default severity of enterpriseSpecificTrap notifications is indeterminate. You may want to create more meaningful alarms by customizing the SNMP trap daemon's mapping of enterprise-specific traps, or by using a Nerve Center request that subscribes for enterpriseSpecificTraps and logs nerveCenterAlarms with severities that match the cause, as indicated by the specific trap type. (An example of a Nerve Center request that subscribes for enterprise-specific traps is described in Chapter 1, "Building Request Templates.")

If you do not want enterpriseSpecificTrap notifications to automatically affect icon color in the Viewer, one way to accomplish this is to edit the discriminator construct for the default AlarmLog to add enterpriseSpecificTraps to the list of excluded event types. However, you may also want to create a separate log to store the enterpriseSpecificTraps for an historical record. To accomplish this change, do the following:

1. **Invoke the Log Manager from the Application Launcher.**
2. **Select the AlarmLog and then select the Edit➤Object Properties... option to invoke the Log Object window.**
   This window displays the CMIS filter that selects the events to be logged to the AlarmLog.
3. **Add a new entry for enterpriseSpecificTraps, as shown in Figure 3-6. Click on OK, of course, to make this change take effect.**
   You should see the change in the “discriminatorConstruct” field for AlarmLog in the main Log Manager window.

discriminatorConstruct:

```
not : or : {
    item : equality : {
        attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
        attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":attributeValueChange
    },
    item : equality : {
        attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
        attributeValue globalForm : "EM MPA":snmAlarmEvent
    },
    item : equality : {
        attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
        attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":objectCreation
    },
    item : equality : {
        attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
        attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":objectDeletion
    },
    item : equality : {
        attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
        attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":stateChange
    },
    item : equality : {
        attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
        attributeValue globalForm : "SNMP Traps":enterpriseSpecificTrap
    }
}
```

Figure 3-6 AlarmLog Discriminator Construct with enterpriseSpecificTraps Excluded

4. **To create a new log for enterpriseSpecificTraps, select the Edit➤Create menu option.**
   This invokes a blank Log Object window.

5. **Enter the name of the new log in the logId field.**
   If you leave the maxLogSize field at 0 (the default), there is no limit on the size. If you enter an integer value in this field, this becomes the maximum log size in bytes. In Figure 3-7 a log name of `SNMPlog` was entered and maxLogSize was set to 5 MB. Select OK to create the new log.

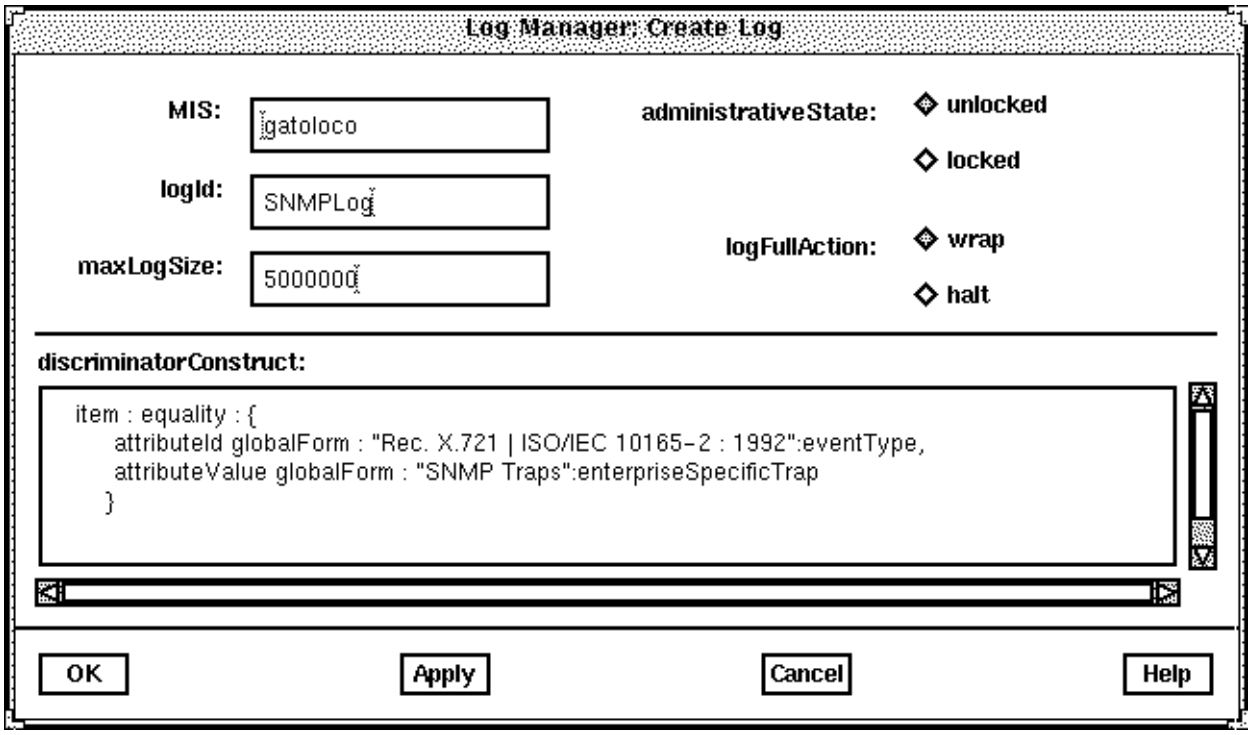


Figure 3-7 Creating a New Log for enterpriseSpecificTraps

3.1.3.5 *Forwarding of Events from SunNet Manager Consoles*

If you have Site/SunNet/Domain Manager Consoles installed in various sites on your network, this can provide an additional source of fault status information for EM. When RPC agents generate event notifications about critical events, in response to threshold-checking initiated from SNM Consoles, Cooperative Consoles can be used to forward these event notifications to the EM MIS. When SNM event notifications are forwarded to EM by Cooperative Consoles, these arrive at the SNM Event Forwarder (`em_snmfwd`) on the MIS

machine. The SNM Event Forwarder translates SNM's fault status indications into EM alarm severities in the manner indicated in Table 3-3. The SNM event notifications are then logged to the AlarmLog as snmAlarmTraps.

Table 3-3 Mapping of SNM Console Fault Indications to perceivedSeverity Values

SNM Event Priority	SNM Fault Status Indicator	snmAlarmTrap perceivedSeverity Value	Default EM Icon Color
Low	color by priority	Minor	Cyan
Medium	color by priority	Major	Orange
High	color by priority	Critical	Red
	blinking	Warning	Yellow
	dim	Indeterminate	Blue
	glyph reset	Cleared	No color

The Alarm Service, which controls the fault status color of icons in the Viewer, monitors the perceivedSeverity of alarms posted against a device, and sets fault status to reflect the highest severity of outstanding (uncleared) alarms against a device. Incoming snmAlarmTraps will thus affect fault status color of icons in the Viewer. (For more information on forwarding of information from SNM Consoles to EM, refer to Chapter 5, "Using Cooperative Consoles with Enterprise Manager.")

3.1.4 Preparing for Fault Management

Looking at the use of Solstice EM from a larger perspective, the steps in preparing for fault management can be summarized as follows:

1. **Decide on the information you need to manage your network.**
   This topic is discussed in Chapter 1, "Planning for Network Management."

2. **Use the Request Designer to create Nerve Center request templates, if needed.**
   Solstice EM is shipped with a number of sample request templates. These may be sufficient for your needs. A request template is a set of commands used to obtain information about network devices, either by direct polling, initiation of a SunNet Manager event request, or by subscribing to receive incoming event notifications, or a combination of these methods.

3. **Use the Log Manager to create logs to store those events for which you want to have an historical record, and to define which events are logged to which logs.**

4. **Configure the `em_log2rdb` daemon to store historical log file data in a relational database, if desired.**
   Solstice EM's relational database logging capability is described in the "Log Management" chapter in the *Solstice Enterprise Manager Reference Manual.*

5. **Chose the logs that you want the Alarm Service to monitor.**
   The event notifications that are logged to these logs are the events that will automatically determine fault indication in the Viewer and Alarm Manager.

6. **Edit the SNMP trap daemon's trap_maps file to customize the mapping of SNMP traps to event notifications.**

7. **If you want to implement forwarding of information from SNM Consoles to EM, use the Cooperative Consoles Configuration Tool to configure the Sender daemons on the SNM machines and the Receiver application on the EM MIS machine.**

3.1.5 For more information…

- On designing Nerve Center request templates — Consult Chapter 10, "Building Request Templates" and Chapter 13, "Request Condition Language."
- On designing Nerve Center requests to handle SunNet Manager event requests — Chapter 11, "Building Templates for SunNet Manager Event Requests."
- On using the Request Designer application — Refer to Chapter 12, "Request Designer."
- On using the Log Manager — Refer to the "Log Manager" chapter in the *Solstice Enterprise Manager Reference Manual.*
- On relational database logging — Refer to the "Log Management" chapter in the *Solstice Enterprise Manager Reference Manual.*
- On customizing the SNMP trap daemon mapping of SNMP traps to event notifications — Refer to Chapter 8, "Mapping SNMP Traps to CMIP Event Notifications."
- On the Alarm Service — Refer to the "Alarm Service" chapter in the *Solstice Enterprise Manager Reference Manual.*
- On configuring Cooperative Consoles for EM — Chapter 5, "Using Cooperative Consoles with Enterprise Manager."

- On event notifications — Refer to Chapter 8, “Mapping SNMP Traps to CMIP Event Notifications” and Chapter 15, “Adding New Event Types.”

3.2 Monitoring Fault Status of Devices

The following is a summary of steps you can take to monitor the fault status of network resources:

1. **Choose the devices you want to be the target of requests.**
2. **Select a request template that will give you the type of information about the device you want and launch the request.**
   You use the Request tool, invoked from the Viewer’s Tools menu, to launch, examine, and stop requests. A request is created when a request template is launched after selecting a particular device. The same template can be launched against multiple devices to create multiple requests.
3. **Monitor the fault status of objects in the Viewer.**
4. **View incoming event notifications, including alarms logged by requests, in the Alarm Manager.**
   The Alarm Manager allows you to sort, acknowledge, clear, and purge alarms. From the Alarms Manager, you can invoke the Grapher, to represent management data in pictorial form.
5. **Use the Log Viewer to browse the historical record of events stored in logs.**
6. **Use the SNMP Browser to obtain a snapshot of SNMP devices.**
7. **Use the Data Viewer to obtain a snapshot of RPC or CMIP devices.**

3.2.1 Launching Requests

Solstice EM is shipped with a number of Nerve Center request templates that you can use for managing devices that support SNMP and RPC protocols. Table 3-4 describes what each template does and the type of device the template can be launched against. A number of these templates are described in detail in the paragraphs that follow.

Table 3-4 Request Templates Shipped with Solstice EM

Name	Description	Target Device Type
IsSnmpSystemUp	Directly polls an SNMP agent for its system description to determine if it is running.	SNMP agent system
AdminOperStatusUp	Directly polls all of the interfaces on a router to determine if ifOperStatus is not up. If not, the request polls for ifAdminStatus. If ifAdminStatus is up but ifOperStatus is not, a critical alarm is logged.	Router
CheckCPU	Uses the hostperf RPC proxy agent to initiate two SNM event requests to check for CPU usage. A minor alarm is generated if CPU usage is greater than 50%. A major alarm is generated if CPU usage is greater than 80%.	UNIX host that is manageable via the hostperf RPC proxy agent.
DeviceReachablePing	Initiates an SNM event request using the RPC ping proxy agent to check for device reachability. Posts a critical alarm if the device is not reachable and a warning alarm when device is up after having been down.	Host that is manageable via the ping RPC proxy agent.
LinkUp	Directly polls agent on each device for ifOperStatus. If no response, logs critical alarm indicating the agent is down. If agent comes back up, a warning is logged. If agent response indicates ifOperStatus is down, the request polls for ifAdminStatus. If ifAdminStatus is up but ifOperStatus is down, a critical “Link Down” alarm is logged against the down interface.	Link

Table 3-4 Request Templates Shipped with Solstice EM

Name	Description	Target Device Type
PingUpOrDown	Direct polling for reachability of device using the RPC ping proxy agent.	Host that is manageable via ping RPC proxy agent.
SnmpLinkUpDownTrap	Subscribes for linkDownTrap notifications and logs a critical alarm if a linkUp trap is not received within one minute of arrival of a linkDown trap.	Routers
SnmpPingBackoffReachable	Directly polls to determine whether the SNMP agent is up, and if there is no response, uses ping to determine whether the host machine is reachable. If there is no response to the poll for reachability, a “Device down” critical alarm is logged.If the device is reachable but the SNMP agent isn’t responding, a major alarm is logged indicating the SNMP agent is down.	SNMP agent system that is also manageable via ping.

3.2.1.1 *Example: Polling for SNMP System Availability*

Polling of managed objects is initiated by launching a *request* from the Viewer. A request is based on a request template. You can have multiple requests, targeted at various objects, based on the same request template. Requests are launched from the Requests tool, invoked from the Viewer’s Tools menu. The Requests tool provides you a menu of all the request templates currently available in the MIS.

Note – You are not restricted to launching requests against devices represented in the local MIS. If the Viewer is connected to an MIS which has established a connection to another MIS, you can also launch requests against managed objects in that remote MIS.

Direct polling can be illustrated using the `IsSnmpSystemUp` template, shipped with Solstice EM. As its name suggests, this template determines whether the Simple Network Management Protocol (SNMP) daemon is running on the target system.

To launch this request, do the following:

1. **Select an appropriate object in the Viewer.**
   This must be an object that supports SNMP protocol. (To check which network management protocols an object is configured to support, or to change this configuration, double-click on the object's icon in the Viewer to invoke the Object Configuration Tool.)

2. **Invoke Tools➤Requests to bring up the Requests tool.**

3. **In the Requests window, select the `IsSnmpSystemUp` request template in the Request Template display area and click Start.**
   The request will be added to the list of running requests.

4. **Selecting Examine brings up a window that provides dynamic tracking of the values of the system and user-defined variables in a running request.**

The `IsSnmpSystemUp` request polls for the value of the system description attribute in the SNMP RFC 1213 `(MIBII) internetSystem` group. If the agent system responds, the request knows it is up. You may want to open this request in the Request Designer to examine how the request is designed to work.

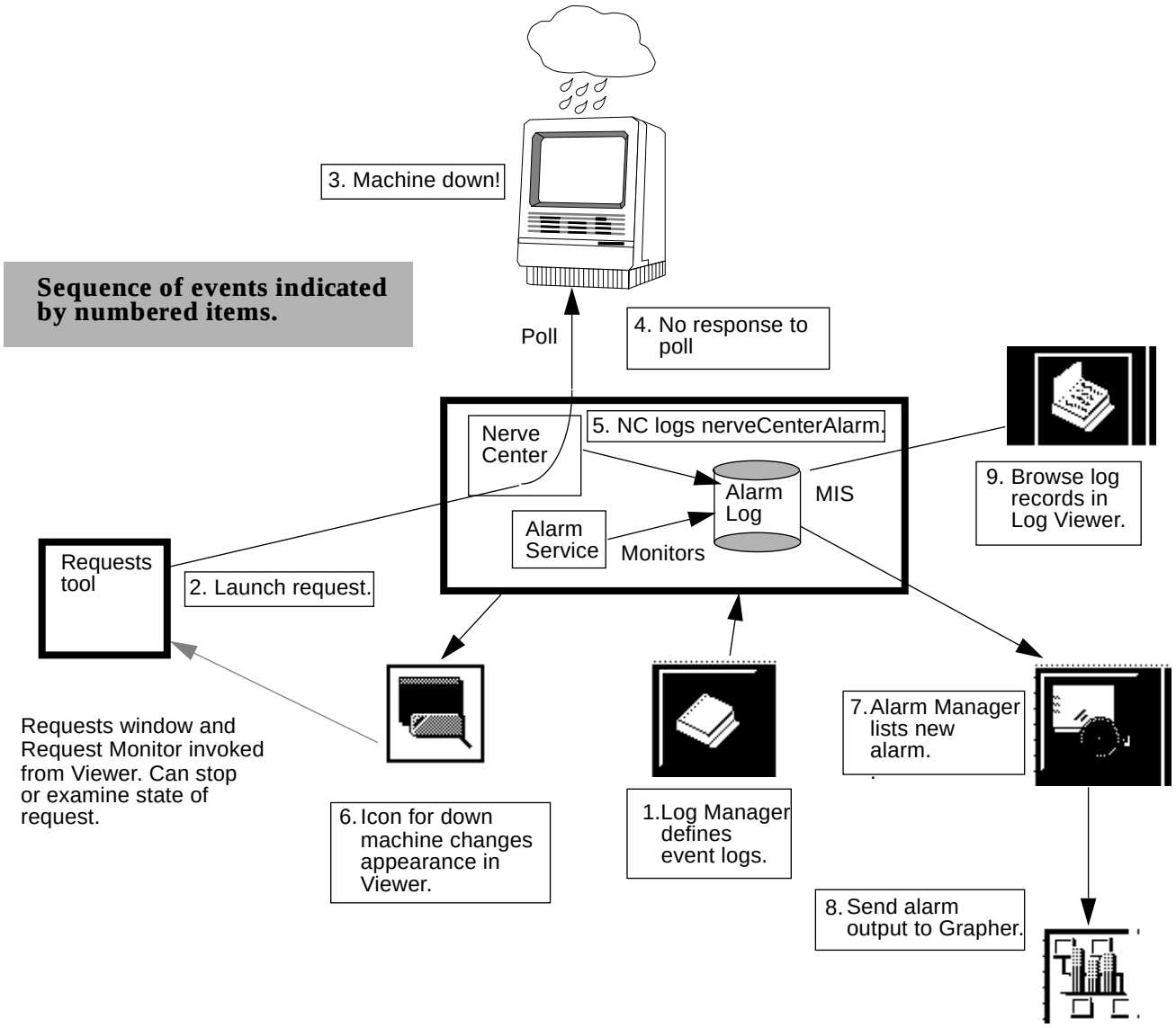


Figure 3-8 Interaction of Nerve Center Request and EM Applications

The result of a request is largely a matter of the action defined for the transition from one state to another in the request template used for that request. The `IsSnmpSystemUp` request calls a Request Condition Language (RCL) alarm-logging function to log nerveCenterAlarms with different severities that indicate the fault status of the object. If the agent system fails to respond, the request logs a nerveCenterAlarm with a severity of critical. The Alarm Service then sets an appropriate fault indication for the object — and its icon in the Viewer turns red. If the agent system becomes available and responds to a poll, an alarm with a severity of cleared is logged. This clears the previous critical alarm, causing the red color to disappear from the icon.

Figure 3-8 shows the various Solstice EM applications you can use to view the results of a request such as `IsSnmpSystemUp`.

3.2.1.2 *Example: Using* `sysUpTime` *to Monitor SNMP Devices*

A limitation of the `IsSnmpSystemUp` template is that it can only tell you whether an SNMP agent system is up on the occasions when it polls the agent. But you might want to be notified if an SNMP system has been down momentarily between polls and then become available again. Another attribute in the SNMP MIB II internetSystem group — `sysUpTime` — could be used as the basis of a request that can do this. The time since the last system restart, in hundredths of a second, is stored in the sysUpTime attribute. Chapter 10, “Building Request Templates,” shows you how to build a template — `IsSnmpSystemEverDown` — which checks for a decrease in the `sysUpTime` value. If this value has decreased since the previous poll, the request knows the system has been down between polls. If the IsSnmpSystemEverDown request determines that sysUpTime has decreased, it logs a nerveCenterAlarm with a severity of minor. This causes the device icon to “decay to cyan” in the Viewer, indicating that it is currently up but was down at some time in the past.

3.2.1.3 *Example: Using both* `sysUpTime` *and* `ping` *to Monitor Devices*

A limitation of the `IsSnmpSystemEverDown` template is that it cannot distinguish between a situation where the lack of response to a poll for `sysUpTime` is due to the SNMP daemon being down and a situation where the lack of response is due to the unavailability of the machine on which the daemon is installed. The `SnmpPingBackoffReachable` sample template

overcomes this limitation by correlating information from two polls — a poll for the SNMP sysUpTime attribute and a poll for reachability, using the `ping` RPC proxy agent.

The SnmpPingBackoffReachable request begins by polling the SNMP system every 30 seconds for the sysUpTime attribute. If there is no response to the poll, the request backs off the poll rate to 60 seconds — possibly the request needs to wait longer for a response. If there is still no response to the poll, the request attempts to poll the device for reachability using the RPC ping proxy agent. If there is a response to the ping, the request knows that the machine is up but the SNMP daemon is down, and a major alarm indicating this is logged.

However, if there is no response to the poll for reachability, the request knows that the machine itself is unavailable and logs a major alarm indicating this. The request continues to poll for reachability. When a response is received, a warning alarm is logged, indicating that the device is up after having been down. The request then transitions back to polling for the SNMP `sysUpTime` attribute.

3.2.1.4 Example: Checking the Status of Router Interfaces

The `AdminOperStatusUp` sample template is an example of another direct polling request that correlates information from multiple polls. This is a template that you can use to check the status of interfaces on routers.

The request checks the number of interfaces configured for the device and sets the target of the poll initially to the first interface in the list and then transitions from the setup state to the Poll state. The request polls to determine the value of ifOperStatus. If ifOperStatus is up, the request transitions back to the setup state to reset the target of polling to the next interface in the list of interfaces for the router. In this way the request cycles through all the interfaces, checking to ensure that each interface is operatiional. If the request finds an interface whose ifOperStatus value is not up, the request polls for the value of ifAdminStatus. If the ifAdminStatus is not set to up — perhaps the interface has been downed due to operator intervention — the request ignores that interface.

If, however, the request finds an interface that is not operational but its ifAdminStatus is up, the request posts a critical alarm against the device in the transition from the Poll state back to the setup state.

3.2.1.5 *Example: Using SNM Event Requests to Test for Reachability*

The DeviceReachablePing request template, shipped with EM, is an example of a Nerve Center request that initiates a SunNet Manager event request.

When a DeviceReachablePing request is launched against a target host, an SNM event request is sent to the `ping` proxy agent with a polling interval of 30 seconds. The ping proxy agent is to poll to determine if the target device is not reachable. A high priority SNM event is generated by the ping proxy agent if it finds the target device not reachable. This event notification is sent to the SNM Event Forwarder on the MIS machine; the Event Forwarder translates the high priority SNM event into an snmAlarmEvent with a `perceivedSeverity` of critical. The `DeviceReachablePing` request listens for incoming snmAlarmEvents from the target device and posts a nerveCenterAlarm with a `perceivedSeverity` of critical if an SNM event is received.

The ping proxy agent uses a “timeout” of 30 seconds when polling the target device. The timeout is the length of time the proxy agent waits for a response to a poll before sending an alarm.

While it is listening for incoming SNM events, the `DeviceReachablePing` request counts the elapsed time since any previous “Device Down” event, and if the elapsed time is greater than the timeout used by the ping proxy agent in polling the device, the DeviceReachablePing request assumes the device is up and posts a minor alarm to indicate the device is up after having been down.

This request template is discussed in detail in Chapter 11, “Building Templates for SunNet Manager Event Requests.”

3.2.2 *Obtaining a Picture of an RPC Device Through the Data Viewer*

The Data Viewer is a Solstice EM application that enables you to examine the data that is collected by the on-going operation of an RPC or CMIP agent. You can invoke the Data Viewer from the pulldown menu for a device in the Viewer canvas (as shown in Figure 3-10). This tool enables you to perform one-time gets of information or to establish polling of attribute values. It also allows you to perform sets.

The Data Viewer can be used to obtain data from RPC agent groups that are supported on target devices. In the example shown in Figure 3-9, a snapshot is obtained of the current values of the `hostperf-data` group on the host

gatoloco. The Inspect Criteria window indicates that lpstat-status, ping-reach, and ping-stats are three additional RPC agent groups supported for this device. Selecting one of these groups would cause the Data Viewer to retrieve the current values of the selected group.

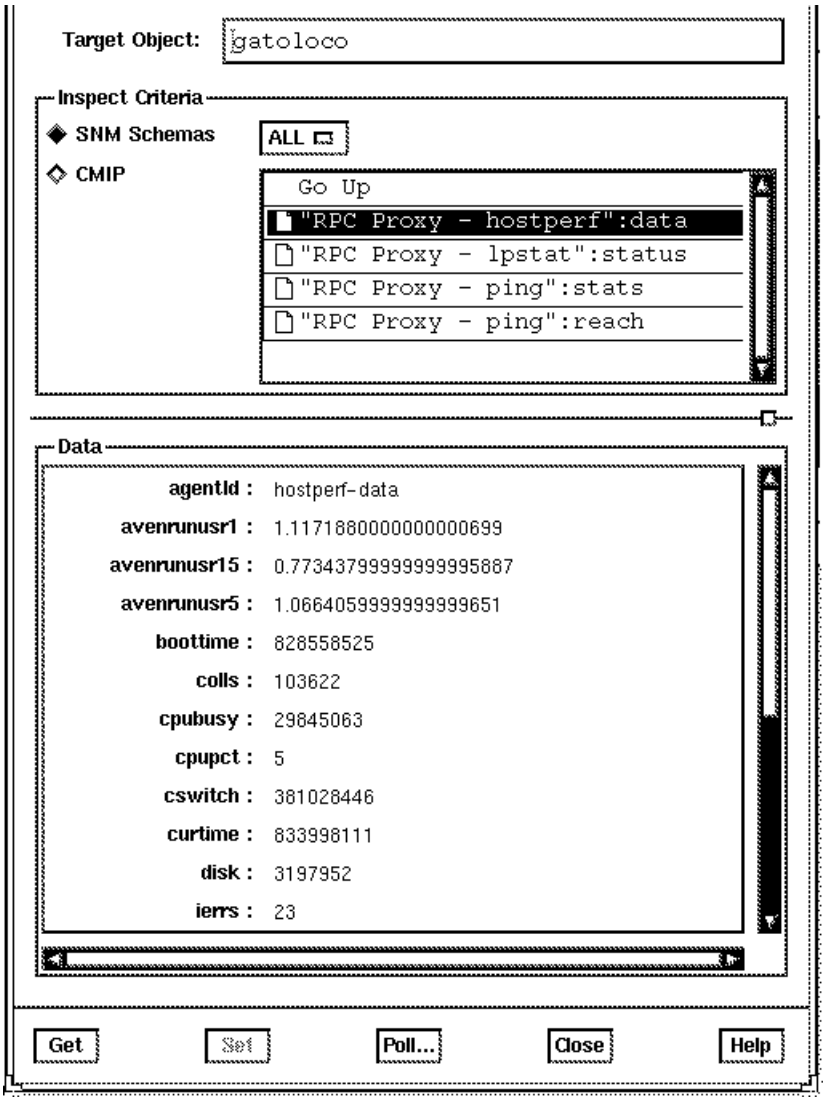


Figure 3-9 Using Data Viewer to Obtain hostperf Data

3.2.3 Obtaining a Picture of an SNMP Device Through the SNMP Browser

The SNMP Browser is a Solstice EM application that enables you to examine the data that is collected by the on-going operation of an SNMP agent for a selected device, or browse the attributes defined in SNMP MIBs known to the MIS. You can either invoke the SNMP Browser from the EM Application Launcher, or from the pulldown menu for a device in the Viewer canvas (as shown in Figure 3-10). This tool enables you to perform one-time gets of information, or you can create tabular views of selected SNMP attributes and then establish polling for the selected attributes. The tool also allows you to perform sets.

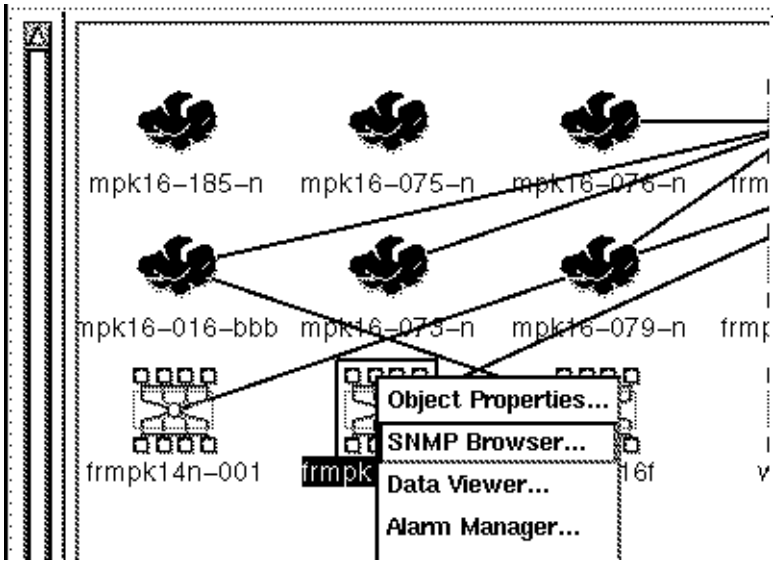
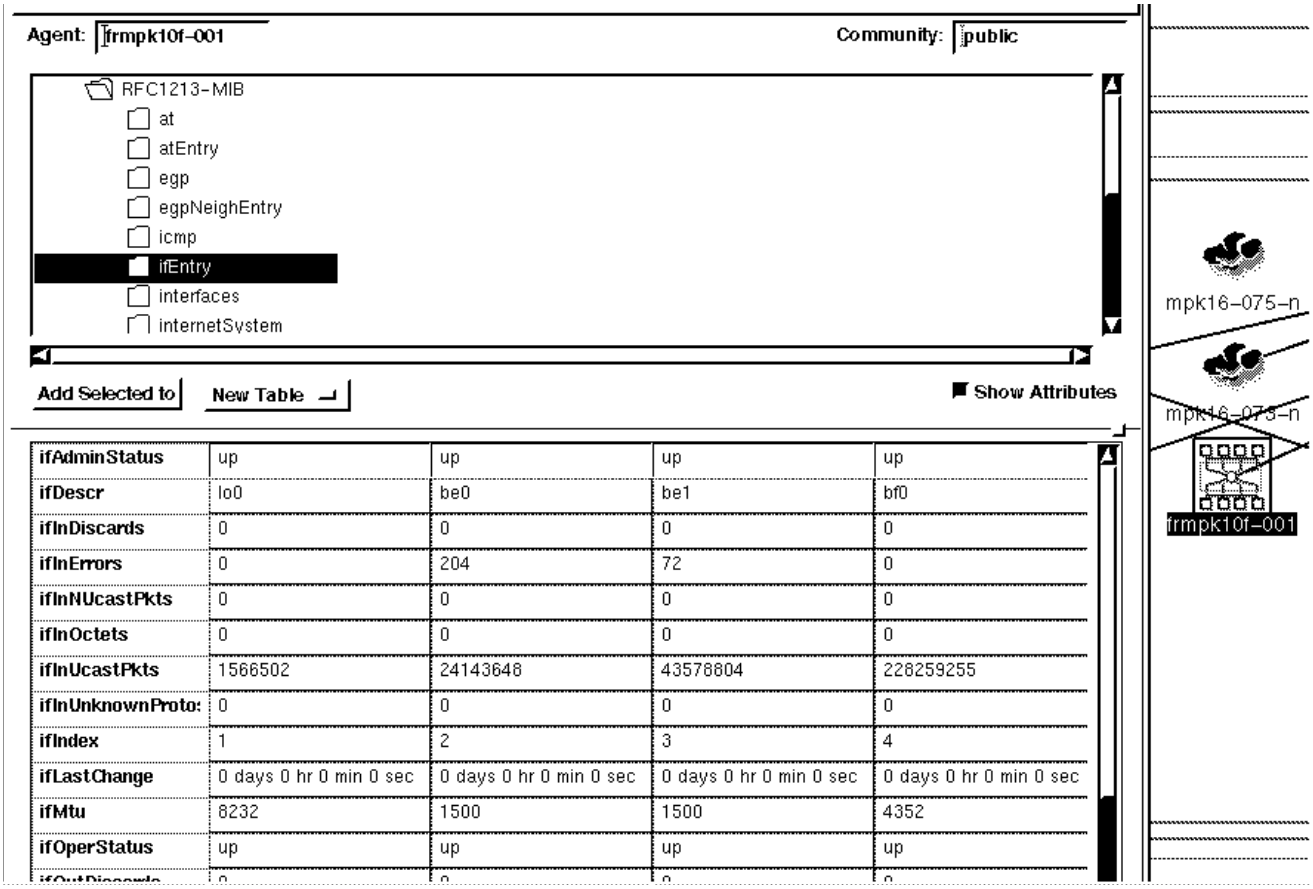


Figure 3-10 Invoking the SNMP Browser from the Element Icon Menu

In the example in Figure 3-10 we have selected a router in the Viewer and invoked the SNMP Browser. In this case, the SNMP Browser automatically comes up with the current values of the router ifTable, as shown in Figure 3-11.



ifAdminStatus	up	up	up	up
ifDescr	lo0	be0	be1	bf0
ifInDiscards	0	0	0	0
ifInErrors	0	204	72	0
ifInNUcastPkts	0	0	0	0
ifInOctets	0	0	0	0
ifInUcastPkts	1566502	24143648	43578804	228259255
ifInUnknownProto:	0	0	0	0
ifIndex	1	2	3	4
ifLastChange	0 days 0 hr 0 min 0 sec	0 days 0 hr 0 min 0 sec	0 days 0 hr 0 min 0 sec	0 days 0 hr 0 min 0 sec
ifMtu	8232	1500	1500	4352
ifOperStatus	up	up	up	up

Figure 3-11 Using SNMP Browser to Get Router ifTable

3.3 Performance Management

Collecting data on the utilization rates of network resources can help you to determine whether particular resources are overburdened. You may want to collect statistical data to compare the performance of your critical nodes, such as servers and gateways, over a period time. Data collection can also provide you with statistics on network traffic levels at various times.

The Data Collector is the Solstice EM tool for gathering data from devices in your network. The Data Collector allows you to create data requests — called *request objects*. The Data Collector allows you to set the polling rate for requests, and to schedule the period of time when the data-gathering is active.

In most cases, you will probably not want continual polling for data as this can itself add an additional burden to the network. If you activate data-gathering periodically, you can develop an historical record that can help you spot trends, and determine “normal” load levels.

If you determine that problems occur on particular devices when utilization rates reach certain levels, you can use that information in fault management — for example, in setting thresholds that trigger alarms in Nerve Center requests.

A request object is essentially a definition of the type of data that is being gathered, polling frequency, and which managed resources (represented on local or remote MISs) to poll for the data. There is no limit, other than system resources, on the number of concurrent data requests.

In the example ilustrated in Figure 3-12, we have invoked the Data Collector’s Create window to define a new data request. The target of this data request is the `ifEntry` object under the MIB II interfaces group for the router `frmpk16c-76`. Clicking on the Object Attributes... button invokes the list of attributes supported by the target object. We’ve selected the ifInErrors attribute for this data request. The polling rate is set at 15 minutes to minimize network traffic load.

Two formats are available for log output. In this example we have selected SunNet Manager log file format. Various third-party applications that support SNM log file format can be used for presentation and analysis of the data. For more information on the EM Data Collector, consult the “Data Collector” chapter in the *Solstice Enterprise Manager Reference Manual*.

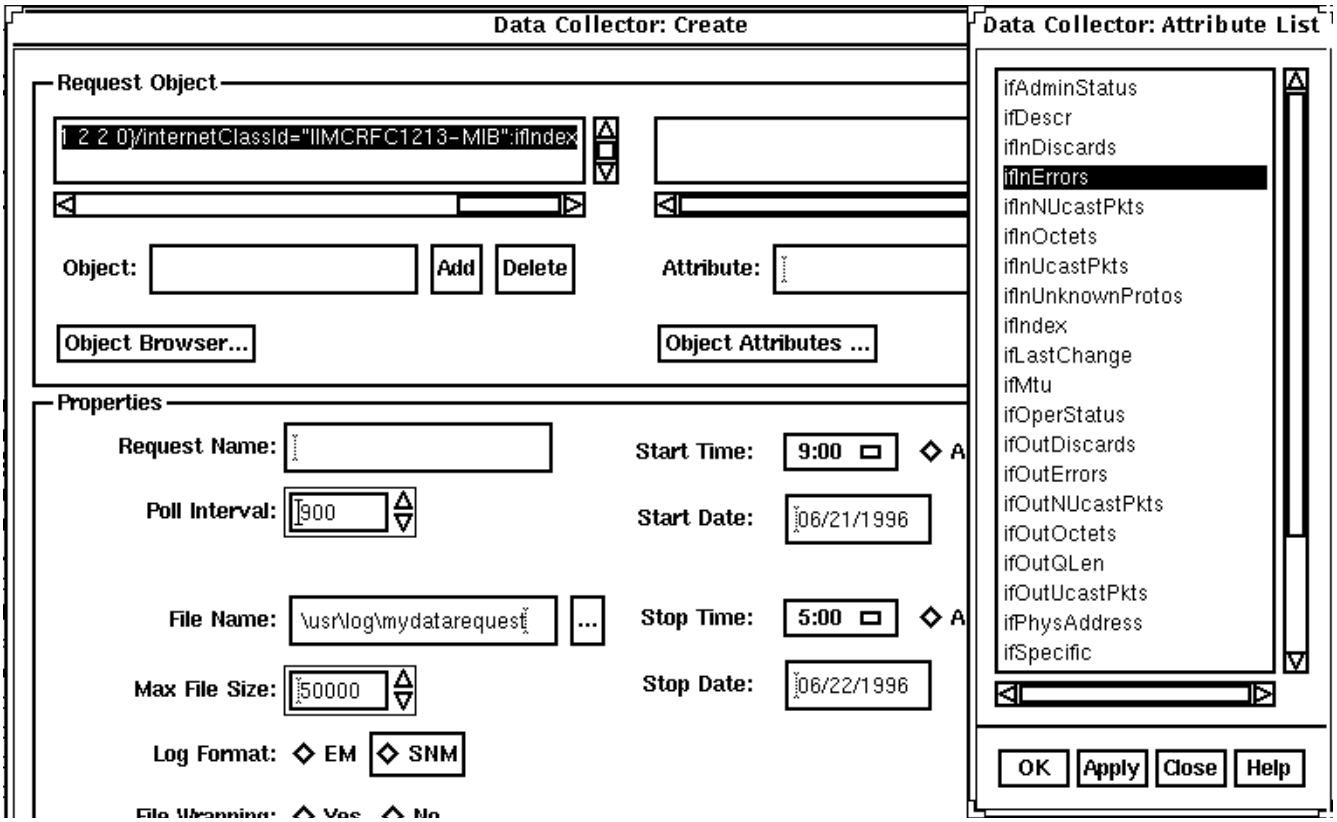


Figure 3-12 Creating a Data Request

3

Device Management Using RPC Agents

4

Overview	*page 4-1*
Managing Network Resources with RPC Agents	*page 4-4*

4.1 Overview

Solstice Enterprise Manager is shipped with a suite of agents developed for the Site/SunNet/Domain Manager (SNM) network management system. These agents communicate with a network manager, such as Solstice EM, using Remote Procedure Call (RPC) protocol within an Internet (TCP/IP) network environment. When deployed on systems in your network, these RPC agents and proxy agents can be used by Solstice EM as part of your strategy for managing network resources. The resource may be a machine, a component in a machine (such as a router interface card), or some other resource. The RPC agent may be local to or remote from that resource.

There are two types of SunNet Manager RPC agents: those that directly access managed resources and those that indirectly access managed resources. Most of the RPC agents provided with EM manage resources on the Sun workstations (or PCs running Solaris for x86) where they are installed. For example, the `diskinfo` agent provides file system usage data.

The second type of agent provides the ability to manage resources that reside in other Sun workstations or in other vendors' devices. Such agents are called *proxy agents*. Proxy agents run on machines running Solaris, called *proxy*

systems, and use protocol translation mechanisms to provide the necessary access to the managed resources. The proxy system can also be a workstation in a different subnet or domain from where the EM MIS is running.

As illustrated in Figure 4-1, SNM agents and proxies use Remote Procedure Call (RPC) protocol to communicate with a management station. However, an SNM proxy agent may use a different management protocol in gathering information from other agents.

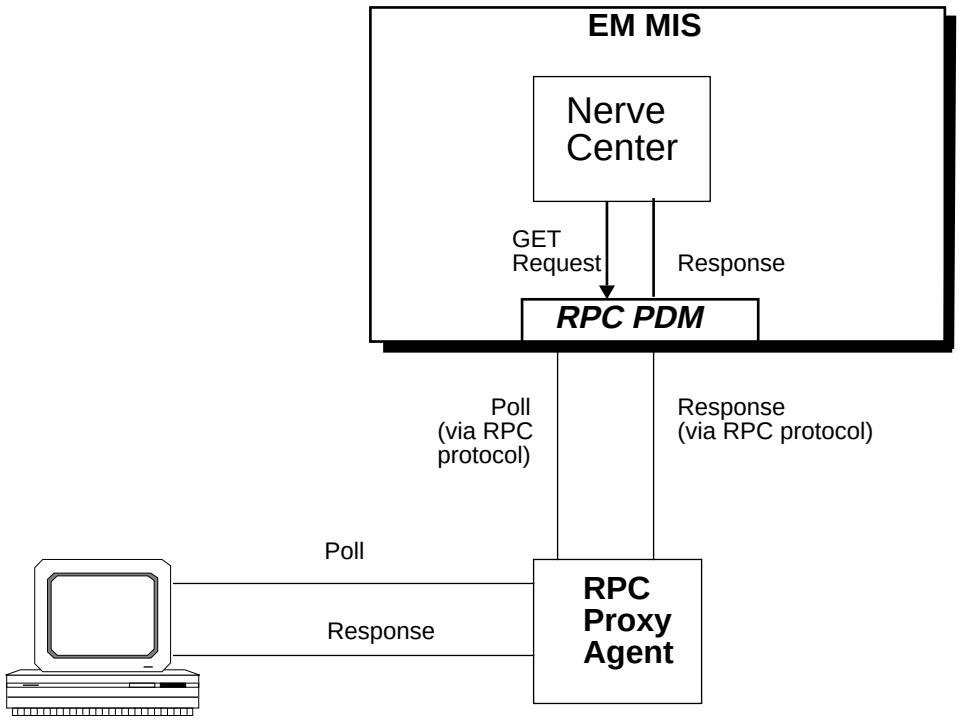


Figure 4-1 Communication with RCP Agents in Direct Polling Requests

Solstice EM Nerve Center requests can obtain information from RPC agents in two ways:

- **Direct polling by the Nerve Center** — A request running in the Nerve Center can directly poll the agent, at intervals specified in the request template. The goal of such a request is to obtain the values of the specified attributes directly from the agent. Building request templates that do direct polling of RPC agents is described in Chapter 10, "Building Request Templates." Communication between manager and managed resource in a direct polling request is illustrated in Figure 4-1.
- **Offload polling activity to the RPC proxy agent** — A Nerve Center request can send a one-shot message, called an *SNM event request*, to an RPC proxy agent. The event request causes the RPC proxy agent to begin polling for a threshold specified in the event request. The event request also specifies the polling interval. Polling of the managed resource is thus handled by the RPC agent rather than the Nerve Center. This minimizes the polling work required of the MIS and allows the polling to be distributed to a site closer to the resource being polled. If the event defined in the event request occurs, the RPC agent sends event information to the SNM Event Dispatcher (`na.event`) running on a specified management station (by default, this is the station that initiated the request). When an SNM event notification arrives at an MIS machine, this information is forwarded to the EM MIS by EM's SNM Event Forwarder (`em_snmfwd`). Communication between manager and agents in SNM event requests is illustrated in Figure 4-2.

  Building templates that use the Nerve Center's SNM event request capability is described in Chapter 11, "Building Templates for SunNet Manager Event Requests."

Note – If you are using SNM Consoles to manage segments of your network, these SNM Consoles will be initiating SNM event requests and tracking network topology changes. Event and topology information received by these SNM management stations can be forwarded to an EM MIS using Cooperative Consoles. This type of distributed management scenario is described in Chapter 5, "Using Cooperative Consoles with Enterprise Manager."

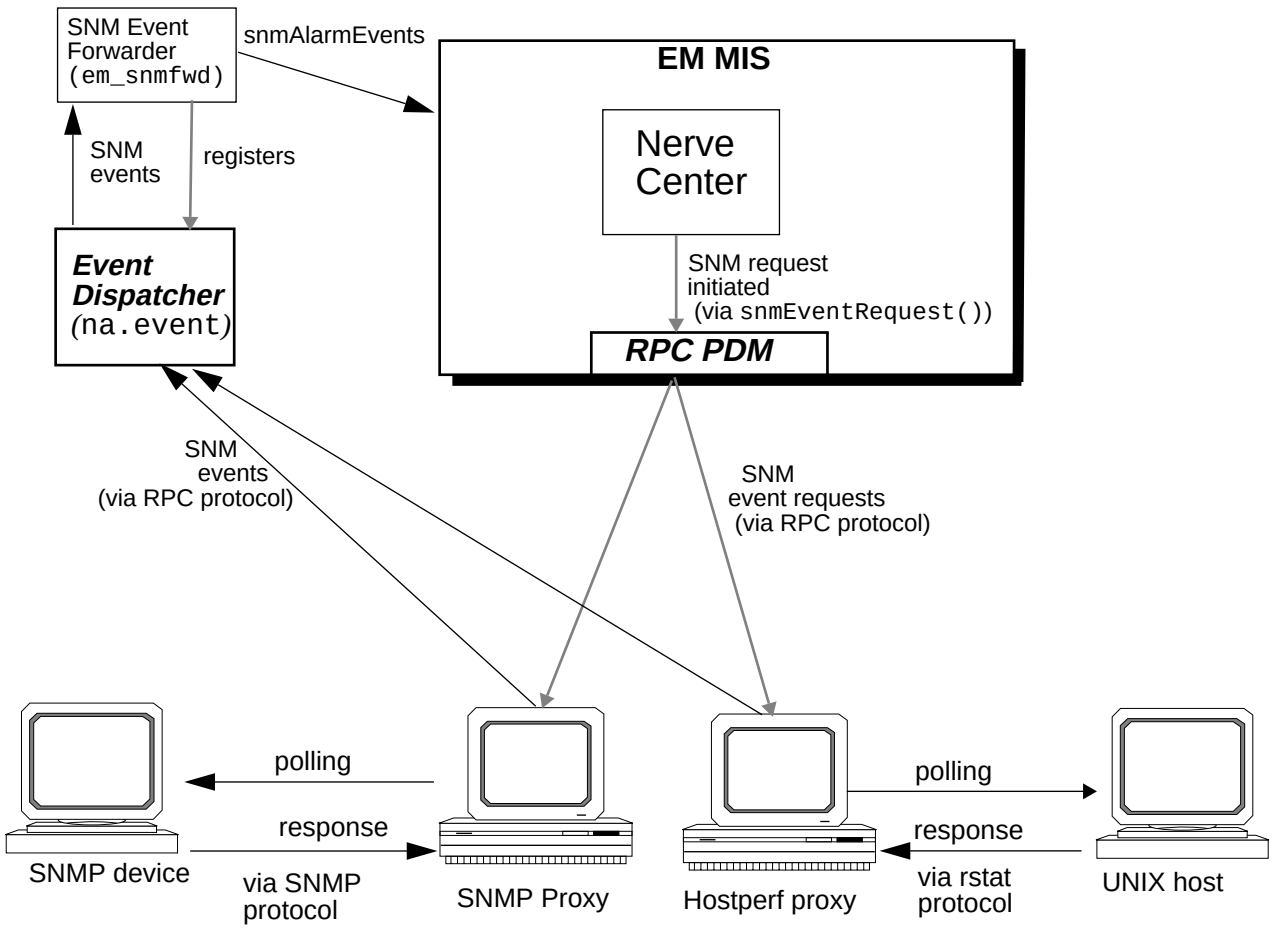


Figure 4-2 Using SNM Event Requests with Solstice EM

4.2 Managing Network Resources with RPC Agents

The steps for using RPC agents to manage network resources are as follows:

1. **Installing and Configuring the RPC agents.**
   Proxy agents may be installed on either the machine to be managed or on a remote system, called a *proxy system*. RPC agents can be installed on three types of machine:

   - SPARC machines running Solaris 1.x (SunOS 4.x)
   - SPARC machines running Solaris 2.x (SunOS 5.x)
   - PCs running Solaris 2.x for x86

   Use `pkgadd` to install the RPC agents package (`SUNWsnmag`) on Solaris 2.x systems (machines to be managed or proxy systems).

   To install agents on a machine is running Solaris 1.x, use the `getagents` script.

   For information on installation of RPC agents on the three types of system listed above, refer to the *Solstice Enterprise Manager Installation Guide*.

   If you have written your own SNM agent or have a third-party SNM agent, copy the file for that agent, with its accompanying configuration files, to the target machine. (You will also need to load a GDMO translation of the SNM schema file into the MIS; this is discussed in Step 2 below.)

   For information on configuring the SNMP proxy agent (`na.snmp`), or the SNMP Version 2 proxy agent (`na.snmpv2`), refer to Chapter 7, “SunNet Manager SNMP Proxy Agents.”

2. **Adding Object Classes to the MIS based on SNM Schemas**
   If you are using only those SNM agents shipped with Solstice EM, you can skip to Step 3.

   The definition language used internally in the Solstice EM MIS to describe object classes is the Guidelines for the Definition of Managed Objects (GDMO), outlined in the ISO/IEC 10165-4 standard. An object class defines the structure of the management information of a managed resource.

   Schema files are the corresponding method for object type definition native to SNM. Schema files are used for loading information about the management capabilities of RPC agents into the SunNet Manager Console. Solstice EM includes a schema-to-GDMO compiler for translating native SNM schema files into GDMO documents. If you want Solstice EM to acquire knowledge of the capabilities of an SNM-compatible RPC agent that you have written, or which you have acquired from a product vender, you must convert the agent’s schema file to pertinent GDMO and ASN.1 files,

and these must be loaded into the MIS. The procedure for accomplishing these tasks is described in Chapter 18, “Adding an Object Class Based on an SNM Schema File to the MIS.”

However, you will only need to do a schema-to-GDMO conversion if you have SNM schemas not provided with EM. For all SNM schemas shipped with EM, corresponding GDMO documents are already provided with the product, and these are loaded into the MIS at startup.

Note – If you are using an RPC-based SNMP proxy agent (`na.snmp` or `na.snmpv2`), SNM schema files must also be provided on the proxy system to enable the proxy agent to map Management Information Bases (MIBs) for SNMP devices that are to be managed. For more information refer to Chapter 7, “SunNet Manager SNMP Proxy Agents.”

---

3. **Configuring the managed object in the MIS.**
   The object in the MIS that represents the target managed resource must be configured to indicate support for appropriate SNM agents. There are two ways to accomplish this task:

   - If you use Discover to populate your MIS, you can configure Discover to query hosts for RPC agents and automatically configure objects to indicate RPC agent support when it adds them to the database. The RCP agent selection sheet in the Discover Properties window is shown in Figure 4-3. You can also select the host that will be configured as the proxy system for devices that are manageable via RPC. For more information, refer to the “Discover” chapter in the *Solstice Enterprise Manager Reference Manual*.

- You can also manually configure RPC agent support for objects in the MIS using the Object Configuration Tool (OCT). OCT is invoked from the EM Viewer. Refer to the “Object Configuration Tool” chapter in the *Solstice Enterprise Manager Reference Manual*.

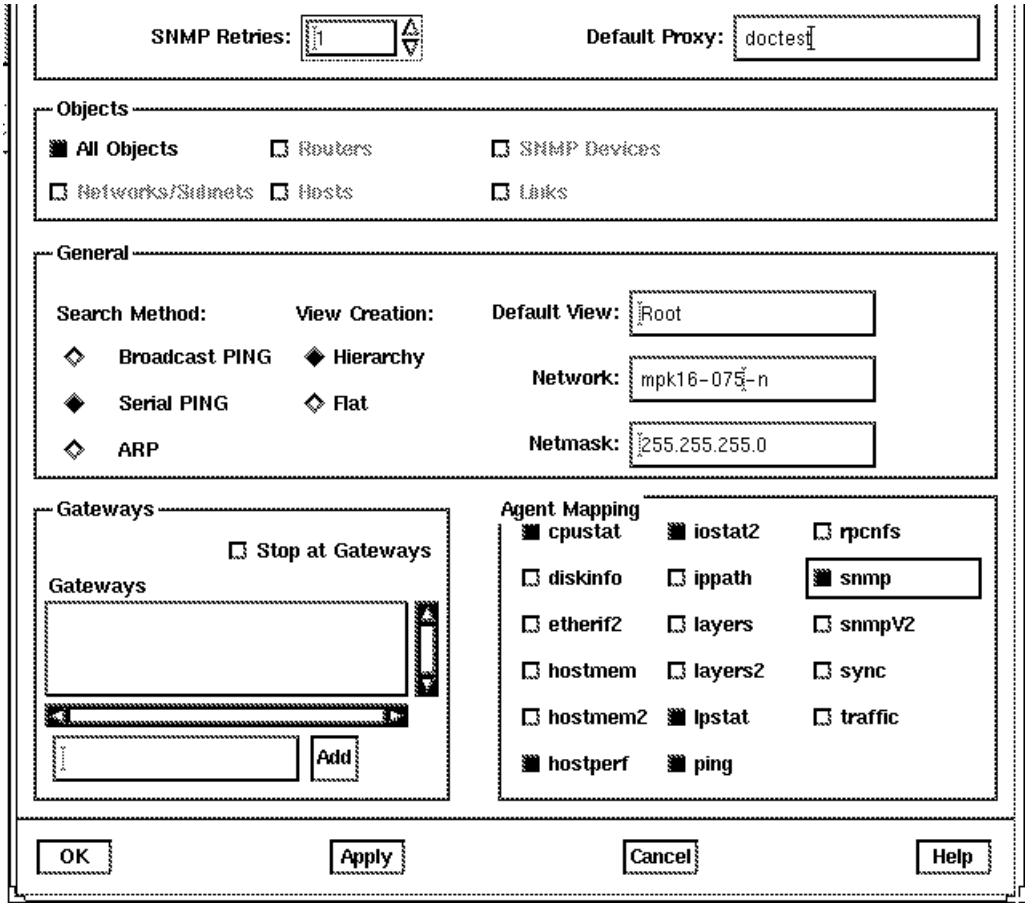


Figure 4-3 Selecting RPC Agents to be Configured during Network Discovery

4. **Building request templates for RPC agents.**
   The Nerve Center module in the MIS contains the request-handling capabilities of EM. Nerve Center requests are based on request templates, which are built using the Request Designer application. A key building block in request templates are request **conditions** — sets of instructions

defined using the EM Request Condition Language (RCL). RCL supports both direct polling of RPC agents as well as setting thresholds for polling by proxy agents. A number of predefined request templates for use with RPC agents are shipped with Solstice EM; these are described below in Table 4-1. If these are sufficient for your needs, you can skip to Step 5. If you wish to build additional request templates for RPC agents, you may want to consult the following sources of information:

- For guidance on building direct polling request templates, consult Chapter 10, “Building Request Templates.”
- For information on the Request Designer, refer to Chapter 12, “Request Designer.”
- For information on Request Condition Language, refer to Chapter 13, “Request Condition Language.”
- RCL provides two built-in functions, `snmEventRequest()` and `snmKillRequest()`, for starting and stopping SNM event requests. Building templates for SNM event requests is described in Chapter 11, “Building Templates for SunNet Manager Event Requests.”
- For additional information on RCL functions that can be used in building request templates, refer to Chapter 14, “RCL Functions.”

5. **Launching requests at target RPC devices.**
   To launch RPC requests at target objects, select the targets in the Viewer and invoke the Requests tool from the Viewer's Tools menu. The Requests tool provides a list of the request templates that are available in the MIS. The requests for managing RPC devices are described in Table 4-1.

Table 4-1 RPC Request Templates Shipped with EM

Template Name	What it does	Type of target
CheckCPU	Sends two SNM event requests to hostperf proxy agent to check CPU usage on target host. Generates minor alarm if usage greater than 50% or major alarm if greater than 80%.	UNIX host configured for hostperf
DeviceReachablePing	Sends an SNM event request to ping proxy agent to check target device for reachability. Generates a critical alarm if device is not reachable. Counts elapsed time since last "Device Unreachable" alarm and posts minor alarm (the icon "decays to cyan") if absence of further alarms indicates the device is available again.	Any device configured for ping
PingUpOrDown	Direct poll for reachability. Logs critical alarm if device is not reachable.	Any device configured for ping
SnmpPingBackoffReachable	Direct poll for SNMP sysUpTime attribute to determine if SNMP daemon is running. If no response, polls for reachability via ping proxy agent to determine whether the non-response to the SNMP poll is due to the SNMP agent being down or due to the unavailability of the host machine. Generates a major alarm if the system is unreachable.	Any system configured for both SNMP and ping

Alternatively, the auto manager can be used to launch SNM requests. This is the most efficient method for using SNM requests to manage large numbers of devices. Refer to the "Automatic Management" chapter in the *Solstice Enterprise Manager Reference Manual.*

6. **Monitoring alarms.**
   Nerve Center requests that manage resources using RPC agents typically log alarms in response to two types of event information:

   - Information obtained by direct polling of the device
   - Incoming snmAlarmEvents generated by an RPC agent when a threshold defined in an SNM event request has been crossed

   When a Nerve Center request posts an alarm using the RCL alarm-logging functions, this generates a nerveCenterAlarm which is, by default, logged to the AlarmLog. Logged alarms can be examined, acknowledged, or cleared using the EM Alarm Manager application.

   When SNM events are sent to the MIS machine, either in response to an SNM event request initiated by the Nerve Center, or forwarded to the MIS machine from a remote SNM Console via Cooperative Consoles, these events are forwarded to the SNM Event Forwarder daemon (`em_snmfwd`). The SNM Event Forwarder posts incoming SNM events to the MIS as `snmAlarmEvents`. Because snmAlarmEvents are, by default, not logged to the AlarmLog by default, they are not monitored by the Alarm Service and therefore do not affect fault status color in the Viewer.

   Be default, only alarms logged to the AlarmLog affect the fault status indication (icon color) of devices in the Viewer. The Alarm Service (a part of the MIS) monitors the alarm log and uses the highest severity of outstanding (uncleared) alarms to determine the fault status of the device.

   However, a request that listens for incoming snmAlarmEvents can use the RCL alarm-logging functions to post appropriate nerveCenterAlarms to the AlarmLog. The RCL subscription functions enable a request to listen for specified types of events.

   If you are using Cooperative Consoles to forward SNM events from remote Site/SunNet/Domain Manager Consoles, you might also want to design templates that use the RCL event subscription functions to listen for incoming snmAlarmEvents, and log appropriate alarms.

   For information on the Alarm Service, Viewer, and Alarm Manager, refer to the appropriate chapters in the *Solstice Enterprise Manager Reference Manual*.

Using Cooperative Consoles with Enterprise Manager 5

Overview	*page 5-1*
Filtering Criteria for Information Forwarding	*page 5-3*
Cooperative Consoles Configuration and Operation	*page 5-4*
Receiving SunNet Manager Alarms	*page 5-7*

5.1 Overview

Cooperative Consoles (CC) provides the ability to implement forwarding of information about selected changes in the state of critical network devices or changes in selected aspects of network topology from management stations running Site/SunNet/Domain Manager (SNM) to one or more Solstice EM managers.

The supported configuration of Cooperative Consoles information forwarding between SNM and EM management stations is a periphery-to-center configuration. This is a distributed management scenario in which management of particular network segments is conducted by SNM Consoles at various sites and there is a one-way forwarding of selected information from the SNM stations to a central EM MIS. The EM MIS thus functions as a central office “manager of managers.” A periphery-to-center configuration is illustrated in Figure 5-1.

There are several types of information that Cooperative Consoles can forward from SNM Console stations to Solstice EM:

- **SNM Events** — These are generated by SNM agents or proxy agents when they detect that a specified threshold has been crossed while polling a target network resource. The polling activity is initiated by an SNM event request issued by the SNM Console. The SNM event request defines the threshold (such as network memory usage greater than 80%) that triggers the generation of the SNM event. The SNM agent or proxy agent uses Remote Procedure Call (RPC) protocol to communicate with the management station.
- **Topology Traps** — The SNM Console generates traps when changes are made to the SNM database, such as addition of a new element or loading of a background image for a view.
- **Glyph State Traps** — The SNM Console generates glyph traps when the user changes the glyph state — for example, if a user resets the glyph state after receipt of an alarm.
- **SNMP Traps** — Cooperative Consoles can forward SNMP traps received by SNM's SNMP trap daemon (`na.snmp-trap`). However, you may prefer to use EM's SNMP trap daemon (`em_snmp-trap`) for distributed SNMP trap forwarding and configurable event type conversion.

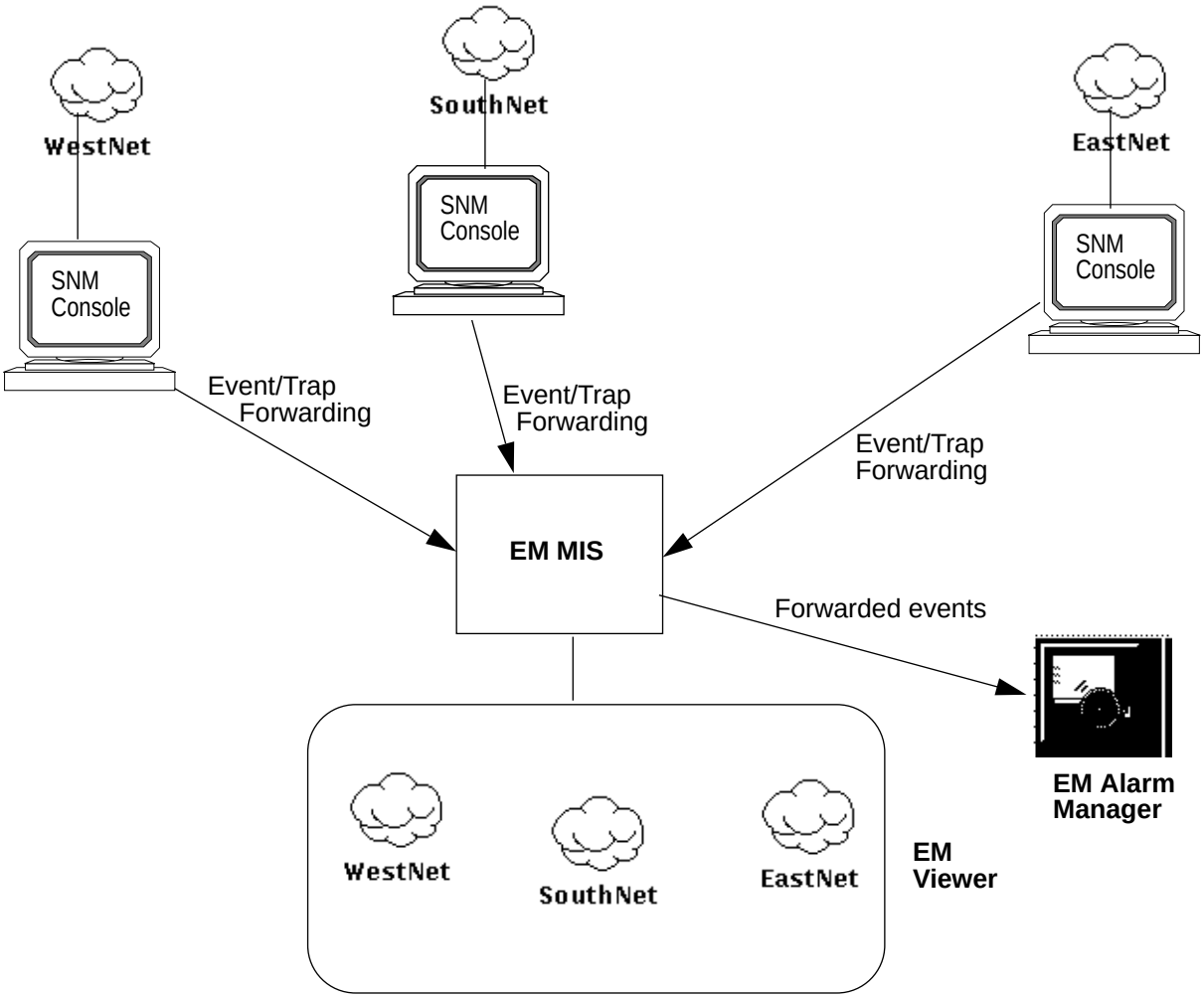


Figure 5-1 Forwarding of Information to Central Management Station

5.2 *Filtering Criteria for Information Forwarding*

The flexible filtering capabilities of CC allows you to select event and topology information to be forwarded on the basis of the following criteria:

- **Type** of the managed resource
  You can choose to forward events by element type, such as routers.
- **Hostname**
  You can select events by the name of the originating device.
- **Priority**
  For example, you might choose to forward only SNM events with High priority.
- **Viewname**
  For example, if certain key objects are in an SNM Console view called "CriticalElements", you could specify forwarding of events for the objects in that view.
- **View type**
  You can select events on the basis of the type of view that the object is in. For example, events from objects in views of type `building` could be selected for forwarding.
- **Event type** — You can select the type of event or trap to be forwarded. For example, you might want to forward only SNM events. If you want to forward SNMP traps, you can choose to forward only standard SNMP traps, or traps can be selected on the basis of the enterprise MIB. For example, 3Com or Cisco traps could be selected for forwarding, and ranges of traps can be selected for the enterprise-specific traps.

5.3 Cooperative Consoles Configuration and Operation

The executable software modules required in setting up a Cooperative Consoles connection between an SNM Console and an EM MIS are:

- **Receiver Application** — A Receiver is to be installed on the EM MIS machine. The Receiver initiates the forwarding of information from remote SNM Consoles to the local MIS. The Receiver maintains a Registration List of the remote SNM stations that it attempts to register with for receipt of event and topology information. You use the CC Configuration Tool to set up the local Receiver's Registration List. When a connection to a remote SNM is running, the Receiver uses the SNM database API functionality in the EM MIS to update the EM MIS to reflect changes in the views on the remote SNM Consoles. EM's support for the SunNet Manager database API is described in Chapter 6, "SunNet Manager Application Support." If CC

forwarding of information has been set up to create a “mirror” on the EM MIS of a particular SNM Console view, then moving or deleting an element in that view on the SNM Console is reflected in the “mirror” in the EM Viewer.

- **Sender Daemon** — A Sender daemon is to be installed on each SunNet Manager host that is to forward event and topology information to the EM MIS. CC’s event and topology filters are used by the Sender daemon. The CC Configuration Tool is used on the sending SNM stations to configure these filters. The periphery-to-center configuration is the only configuration currently supported for EM. In this configuration, no Sender daemon is installed on the EM MIS machine.
- **Configuration Tool** — This is the user interface for configuring operation of the Sender and Receiver processes. Configuring the Receiver on the EM MIS machine also requires installation of the CC Configuration Tool.

Steps to follow in setting up Cooperative Consoles on the EM MIS machine:

1. **Install the CC Configuration Tool and Receiver packages**
   This is described in the *Solstice Enterprise Manager Installation Guide*.

2. **Set your** `LD_LIBRARY_PATH` **to support the Receiver application.**
   Ener a command such as the following to set this environment variable correctly:

```
host% setenv LD_LIBRARY_PATH /opt/SUNWconn/em/lib:/opt/SUNWconn/lib:${LD_LIBRARY_PATH}
```

   Because the CC Receiver is an SNM application, you should consult the general instructions for use of SNM applications in Chapter 6, “SunNet Manager Application Support.”

3. **Add the CC Configuration Tool and the CC Receiver to the EM Application Launcher.**
   This is described in the “Application Launcher” chapter in the *Solstice Enterprise Manager Reference Manual*. You will need to tell the Launcher the path to the CC executables. The default path to the CC Receiver is as follows:

```
/opt/SUNWconn/snm/bin/cc_receiver
```

4. **Use the CC Configuration Tool on the remote SNM Console machines to configure the appropriate Sender daemon filters for event and topology forwarding to the EM MIS.**

5. **Use the CC Configuration Tool to set up the Receiver's Registration List on the EM MIS machine.**

For information on configuring the CC Sender daemon and the CC Receiver application, refer to the *Solstice Cooperative Consoles Administration Guide.*

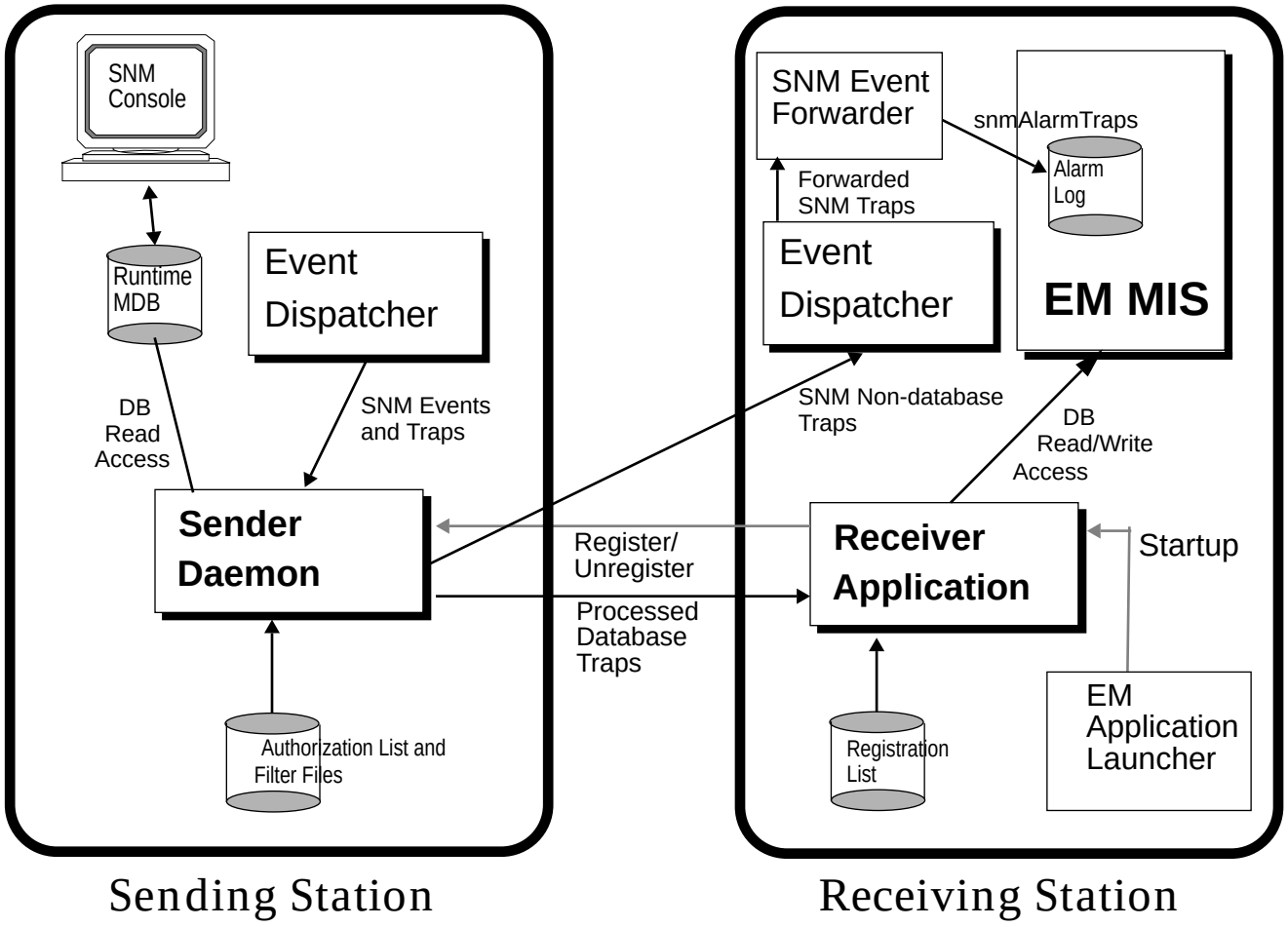


Figure 5-2 Information Forwarding from SNM Console to EM MIS

5.4 *Receiving SunNet Manager Alarms*

The CC Sender daemon on a remote SNM Console can be configured to send SNM events, and notification of user actions clearing these alarms (glyph reset), to the MIS machine. The Sender daemon reformats these SNM events (and glyph reset events) as SNM traps and sends them to the SNM Event

Dispatcher (`na.event`) on the MIS host. The SNM Event Forwarder (`em_snmfwd`) on the MIS machine registers with the Event Dispatcher to receive all SNM events and traps. The Event Forwarder converts the SNM traps into snmAlarmTraps and sends these to the MIS. By default, these event notifications are logged to the AlarmLog.

SNM Console users can configure SNM event requests to indicate fault status of the target device in several ways:

- Dimming of a glyph
- Blinking of a glyph
- Color by priority

Priority is the attribute of an SNM event that represents the severity of an event on the managed resource. If the user has selected color by priority, the SNM Event Forwarder maps SNM priorities to perceivedSeverity values as indicated in Table 5-1. The SNM Event Forwarder also translates dimming or blinking of glyphs into perceivedSeverity values, as indicated in Table 5-1.

Table 5-1 Mapping of SNM Console Fault Indications to perceivedSeverity Values

SNM Event Priority	**SNM Fault Status Indicator**	**snmAlarmTrap perceivedSeverity Value**	**Default EM Icon Color**
Low	color by priority	Minor	Cyan
Medium	color by priority	Major	Orange
High	color by priority	Critical	Red
	blinking	Indeterminate	Yellow
	dim	Warning	Yellow
	glyph reset	Indeterminate	Blue
	pending	Warning	Yellow

The Alarm Service, which controls the fault status color of icons in the Viewer, monitors the perceivedSeverity of alarms posted against a device, and sets fault status to reflect the highest severity of outstanding (uncleared) alarms against a device. Incoming snmAlarmTraps will thus affect fault status color of icons in the Viewer.

If a user resets a glyph to clear an alarm on the SNM Console, a glyph state reset trap is sent to `em_snmfwd` on the MIS machine which generates an snmAlarmTrap with a perceivedSeverity of “Indeterminate.”

When glyph fault status indications are propagated to higher-level views in the SNM Console, a glyph reset is also propagated to those views. Glyph reset traps are thus forwarded for the views that contain the element. These are translated into separate “clear” snmAlarmTraps for the corresponding views in the EM MIS.

Note – If SNM event requests are initiated by the MIS, incoming SNM events from the RPC proxy agents are received by the SNM Event Dispatcher on the MIS host as SNM events (not SNM traps). These are also forwarded to the SNM Event Forwarder (`em_snmfwd`); however, these event notifications are posted to the MIS as snmAlarmEvents. By default, snmAlarmEvents are *not* logged to the AlarmLog. For more information, refer to Chapter 11, “Building Templates for SunNet Manager Event Requests.”

5

SunNet Manager Application Support

6

Overview	*page 6-1*
SNM Applications' Access to Solstice EM Features	*page 6-4*
Adding an SNM Application to Solstice EM	*page 6-6*
Importing an SNM Database into EM	*page 6-9*
SNM Applications' Access to SNM Agents (Over Solstice EM)	*page 6-10*
Solstice EM Applications' Access to SNM Agents	*page 6-12*

This chapter describes the areas in which SunNet Manager applications can interoperate with Solstice Enterprise Manager (EM) as part of an overall network management solution.

Note – For purposes of this guide, *SunNet Manager* (SNM) refers to the 2.2 or later releases of SunNet Manager, and releases of Solstice Site Manager and Solstice Domain Manager. SunSoft makes no claims of compatibility of Solstice EM with versions of SNM prior to 2.2.

6.1 Overview

For the purpose of describing SNM/EM interoperability, we define an **SNM application** as an application that uses the SNM API to access the SNM database and/or to access SNM agents. We define a Solstice EM application as

one that uses the native Solstice EM API (called the Portable Management Interface, or PMI) to access objects in the Solstice EM Management Information Tree (MIT).

Solstice EM and SNM are compatible with each other in the following ways:

- Dynamically-linked SNM applications that use the SNM API to access database elements can run without modification over Solstice EM, to access objects in the Solstice EM MIT. SNM database-access functions, such as `snmdb_open()`, `snmdb_add()`, and `snmdb_delete_from_view()`, are translated by the compatibility library, `libnetmgt_db.so`, into the Solstice EM PMI. You must set or prepend to the `LD_LIBRARY_PATH` environment variable to include the location of the compatibility library, which, by default, is `/opt/SUNWconn/em/lib`.

Note – If you have previously installed SunNet Manager, make sure that `/opt/SUNWconn/snm/lib` does not occur prior to `/opt/SUNWconn/em/lib` in your `LD_LIBRARY_PATH`.

- Dynamically-linked SNM applications that use the SNM API to access SNM agents can run without modification over Solstice EM. These applications use the native SNM `libnetmgt.so` library, which is shipped with Solstice EM. The `libnetmgt.so` library is stored, by default, in `/opt/SUNWconn/snm/lib`. Accessing this library requires that you set `LD_LIBRARY_PATH` to include its location.
- Solstice EM applications can access SNM agents—which are shipped with Solstice EM—through the RPC Protocol Driver Module (PDM) (see Figure 6-1). This capability allows you to take advantage of the power of Solstice EM while retaining the ability to manage SNM agents. Support for Solstice EM applications accessing SNM agents requires no action on the part of the application programmer and user.

  In addition to SNM agents shipped with Solstice EM, Solstice EM supports RPC agents that have been written for SNM but were not shipped with that product.

Support for SNM applications and agents requires some minor configuration steps, which are discussed in Section 6.3, “Adding an SNM Application to Solstice EM.”

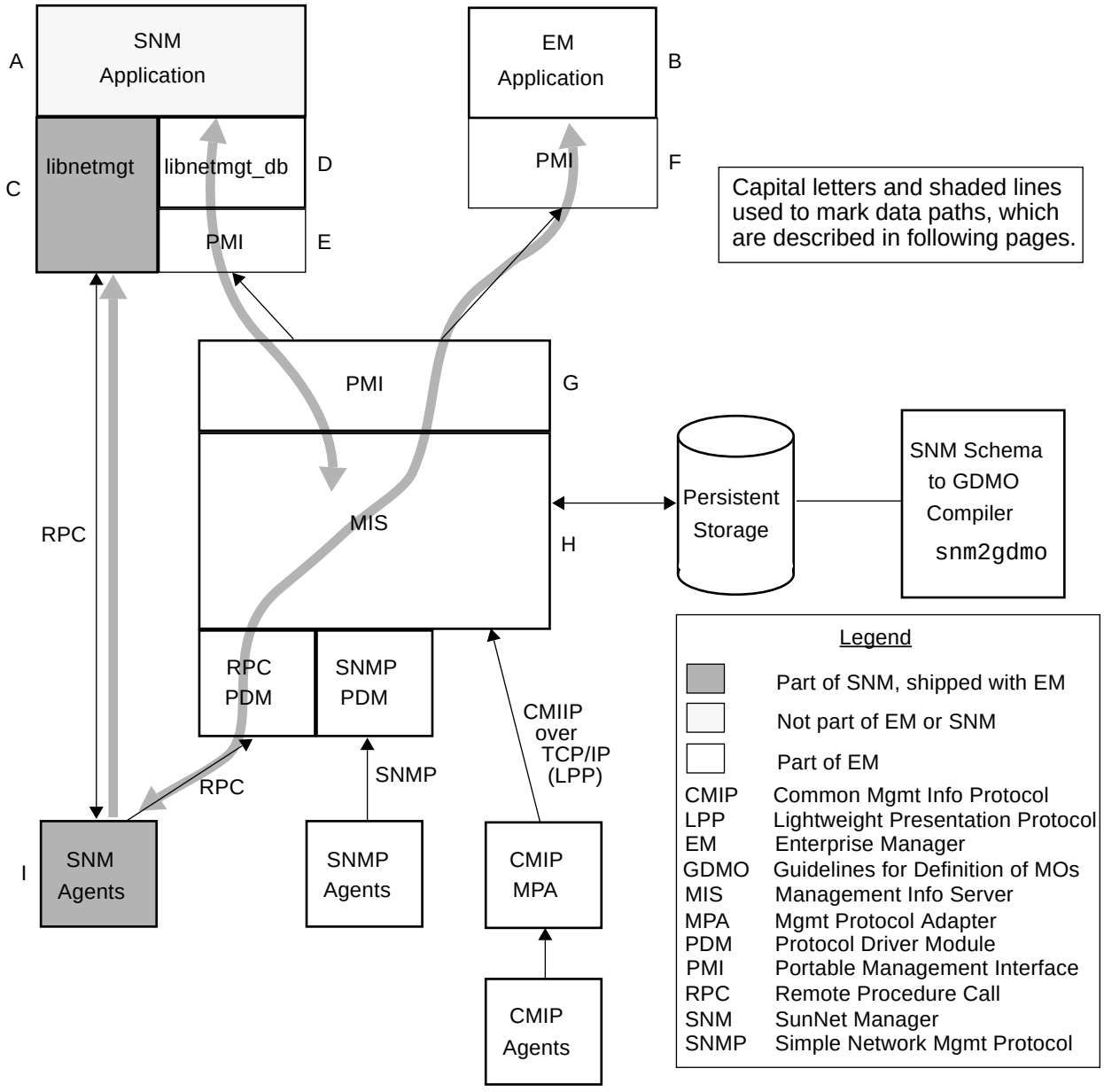


Figure 6-1 SNM-Solstice EM Compatibility

The areas of SNM-Solstice EM compatibility are illustrated in Figure 6-1. This figure is the basis of the discussion that follows.

The areas of compatibility described in the bullets on page 6-2 are illustrated in Figure 6-1 as follows:

- SNM applications accessing Solstice EM features: Path A to D to E to G to H.
- Solstice EM applications accessing SNM agents: Path B to F to G to H to I.
- SNM application accessing SNM agents: Path A to C to I.

In Figure 6-1, for path A to C to I, note that SNM applications access SNM agents through the library `libnetmgt`, just as they do while running the SNM Console.

The following subsections discuss each area of compatibility in some detail.

6.2 SNM Applications' Access to Solstice EM Features

Solstice EM support for SNM applications accessing Solstice EM features is illustrated in Figure 6-2.

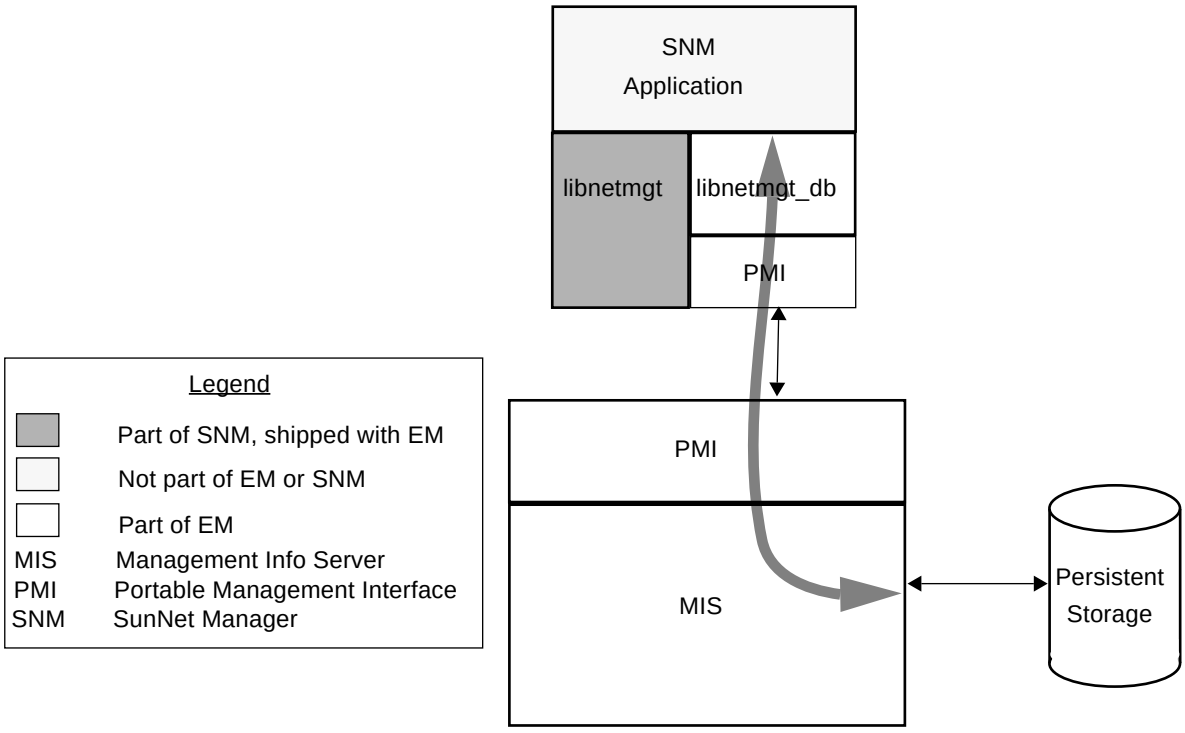


Figure 6-2 SNM Application Accessing Solstice EM Features

An SNM application can access Solstice EM features if and only if the application uses the SNM API only as specified in the *Solstice Site/SunNet/Domain Manager Application and Agent Development Guide* Such applications can access Solstice EM features without any modification to code, without any recompilation or relinking.

Note – Applications that access the SNM management database directly, bypassing the published API, do not run correctly.

Part of the requirement for SNM API conformance is the requirement, spelled out in the *Site/SunNet/Domain Manager Application and Agent Development Guide*, that applications be dynamically linked for compatibility with future releases. The current release of Solstice EM is such a future release.

6.3 Adding an SNM Application to Solstice EM

To run your SNM application with Solstice EM do the following:

1. **Install the SNM API and RPC agents packages (`SUNWembc` and `SUNWsnmag`) on the MIS machine, if you have not already done so.**
   Installation of these packages is described in the *Solstice Enterprise Manager Installation Guide*.

2. **Convert third-party SNM icons to EM glyph format.**
   Element types shipped with Site/SunNet/Domain Manager have already been mapped to EM icons by default. This step is only necessary if you have added third-party icons, not shipped with SNM, that you wish to use with an SNM application accessing EM features. These icons must be converted from SNM Xview format to EM X-pixmap (pm) format. To make this conversion, do the following:

   a. **Convert the SNM glyph to pbm format.**
      You can use the Open Windows icontopbm utility to make this conversion:

```
% /usr/openwin/bin/icontopbm <element-type>.icon > <element-type>.pbm
```

Note – Typically <*element-type*> is the same as the SNM element type name. For example, component.bridge would have an icon named bridge.icon. However, not all third-party SNM icons follow this rule. Note that the EM icon file name must be of the form <*element-type*>`.pm` where <*element-type*> is the name of the element type.

---

   b. **Convert the icon from pbm format to pm format.**
      There are various graphic utilities available that you could use to convert an icon from pbm to pm format. Both the ImageMagick `convert` utility and `netpbm` are shareware packages that you can use to convert the icons from pbm format to X pixmap format. Both packages can be

downloaded, using `ftp`, from the X consortium's `ftp` server (`ftp.x.org`). ImageMagick can be downloaded from: `/contrib/R5contrib-fixes/ImageMagick`. `Netpbm` can be downloaded from: `/R5contrib/netpbm-1mar1994.tar.gz`.

c. **Place the converted pm icon file <*icon*>.`pm` in the $EM_HOME/`glyphs` directory.**

3. **Convert third-party SNM schemas to GDMO documents.**
The default `elements.schema`, `cooptools.schema`, and `netware_elements.schema` files, shipped with SNM, have already been converted to GDMO documents for you, and these are incorporated into the EM MIS by default. However, if you have customized the `elements.schema` file with new entries, or added third-party schema files, these schemas must be converted to GDMO documents and loaded into the MIS. The `em_snm2gdmo` compiler is provided for accomplishing this task. For step-by-step guidance, refer to Chapter 18, "Adding an Object Class Based on an SNM Schema File to the MIS."

4. **Use the `em_snm_type_import` utility (as root) to incorporate new SNM element types, defined in SNM schema files, into the EM environment.**
The syntax for this utility is:

```
# $EM_HOME/bin/em_snm_type_import -file <schema-file>
```

For example::

```
#./em_snm_type_import -file /opt/CSCOcw/snm/struct/cisco.record
```

You only need to do this step if you have custom or third-party element types that you want to incorporate into EM. This utility updates entries in $EM_HOME/`config/SNM2EM_type_mappin`, which maintains the mapping of SNM element types to EM element types. By default, this file contains the mappings for elements defined in the default SNM `elements.schema`, `cooptools.schema`, and `netware_elements.schema` files. The following files are also updated:

- $M_HOME/`install/em_platform/bc_map`

- $EM_HOME/config/em_viewer.cf

New entries are added only for types that are not already present. When you run `em_snm_type_import`, a log is generated in the current directory of the shell where you invoked the command. This log is written to the file `em_snm_type_import.log`. Warnings are generated in the log if matching pixmap format icons have not yet been provided for the imported types. An example of log output would be the following:

```
Warning: Need synfleet-router.pm file in $EM_HOME/glpyhs directory
Warning: Need syn-novell-server.pm file in $EM_HOME/glpyhs directory
Warning: Need baystack-100-conc.pm file in $EM_HOME/glpyhs directory
Warning: Need syn-fddi-segment.pm file in $EM_HOME/glyphs directory
```

5. **Run `em_services -r` to re-initialize the MIS.**
   You need to do this step only if you have done Step 2, Step 3, or Step 4.

Note – Running `em_services -r` recompiles the GDMO and ASN.1 documents. Any existing topology data in the MIS is lost. If you have existing topology data in the MIS that you want to save, you can use the Topology Import/Export tool to export the data prior to running `em_services`. You can then use this same tool to import the data into the MIS after re-initialization.

6. **Add the path to the Solstice EM version of `libnetmgmt` to your `LD_LIBRARY_PATH` environment variable.**
   Assuming you installed Solstice EM in `/opt/SUNWconn/em`, enter a command such as the following to set this environment variable correctly:

```
host% setenv LD_LIBRARY_PATH /opt/SUNWconn/em/lib:/opt/SUNWconn/lib:${LD_LIBRARY_PATH}
```

Following this command, you can successfully run your SNM application.

7. **Set the `EM_SERVER` environment variable if you want to run the SNM application remote from the MIS machine.**
   With regard to applications, Solstice EM has a special feature not available in SNM: Solstice EM allows you to run applications remote from the MIS. This capability is supported through CMIP-over-TCP/IP connections, allowing you to avoid the high bandwidth use and inconvenience of remote X window sessions. In SNM terms, this feature is the equivalent of running

SNM applications on a machine remote from the Console machine (which is not possible in SNM, where Console, database, and applications reside on the same machine).

Solstice EM extends support for remote applications to SNM applications, thereby providing to those applications a feature not available to them in their native SNM environment. To allow your SNM application to connect to an MIS on a remote machine, you must set the `$EM_SERVER` environment variable. This environment variable is available to Solstice EM applications as well. To set this variable, enter a command such as the following:

```
host% setenv EM_SERVER <remote_MIS_machine>
```

With `$EM_SERVER` thus set, subsequent invocations of an SNM (or Solstice EM) application automatically connect to *<remote_MIS_machine>*.

For SNM applications, Solstice EM supports:

- multiple remote applications connecting to (and thereby sharing the data in) a single MIS.
- multiple remote applications connecting to MISs running on multiple machines.

8. **Add an icon for the SNM application to the EM Launcher.**
Invoke the Configure Applications window in the Application Launcher to add the SNM application to the launcher. For information on adding applications to the EM Application Launcher, refer to the "Application Launcher" chapter in the *Solstice Enterprise Manager Reference Manual*. If you select "Yes" in the EM Application field in the Configure Applications window, the icon will be grayed out whenever the MIS has disconnected.

6.4 *Importing an SNM Database into EM*

The `em_snmdb_import` utility enables you to import a SunNet Manager topology database into the runtime database of a Solstice EM MIS. The SNM database must have been previously saved to an ASCII file, using the SNM Console's File➤Save➤Management Database... option to save the SNM database to an ASCII-format file.

The command to import the ASCII-format SNM database file is as follows:

```
# em_snmdb_import -import <filename>
```

Note – The `em_snmdb_import` utility retains the layout of elements within views. However, SunNet Manager event requests in the SNM database are not loaded into the EM MIS.

6.5 *SNM Applications' Access to SNM Agents (Over Solstice EM)*

Solstice EM support for SNM applications accessing SNM agents over Solstice EM is illustrated in Figure 6-3.

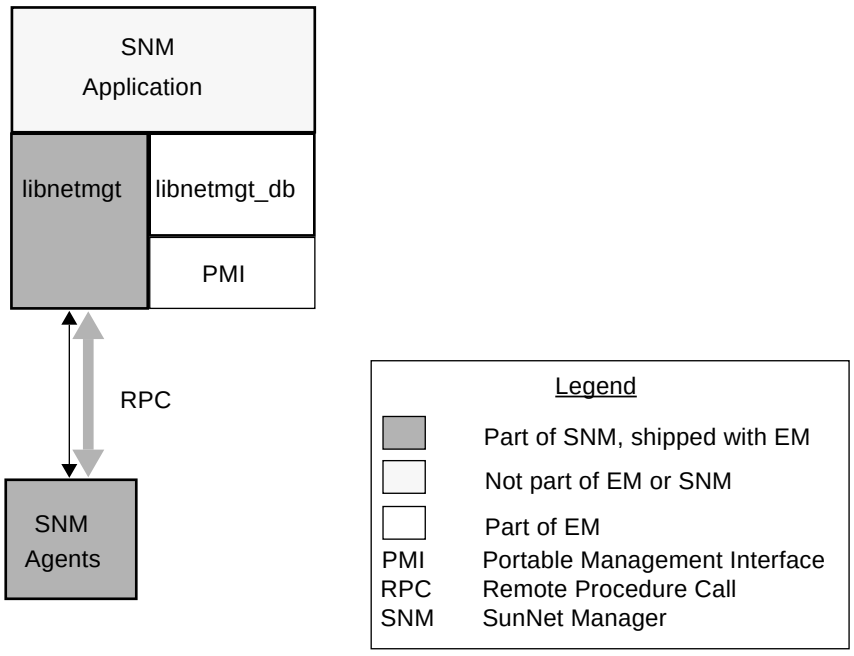


Figure 6-3 SNM Application Accessing SNM Agents over Solstice EM

Under Solstice EM, an SNM application accesses an SNM agent just as it would under SNM, through the `libnetmgmt` library, which is shipped with Solstice EM. The advantage to using Solstice EM rather than SNM is that, as with using Solstice EM applications to access SNM agents, data obtained from agents can be stored in the MIS, which provides a number of user- and programmer-level features that are not present in SNM.

As shipped with Solstice EM, SNM configuration files, such as `snm.conf`, `snmpd.conf`, `snmp.hosts`, and `snmp.traps`, are stored in their normal, SNM 2.x locations and are used in the same way as in SNM 2.x. The default locations of these files are:

```
/etc/opt/SUNWconn/snm/snm.conf
/etc/opt/SUNWconn/snm/snmpd.conf
/var/opt/SUNWconn/snm/snmp.hosts
/var/opt/SUNWconn/snm/snmp.traps
```

The default SNM elements.schema, netware_elements.schema, and cooptools.schema files, required by SNM applications, are incorporated in the EM environment by default. If you have customized the `elements.schema` file, or have added third-party element definitions, then you must follow the steps outlined in Section 6.3, “Adding an SNM Application to Solstice EM.”

The SNM agent and schema files reside, by default, in `/opt/SUNWconn/snm/agents`. Third-party agents and schemas are integrated in the Solstice EM environment just as they were in the SNM environment. As with SNM, in Solstice EM you would add an entry for `na.snmp.schemas` to `snm.conf` for the location of additional third-party SNMP schemas.

The requirement for `$LD_LIBRARY_PATH` for SNM applications accessing SNM agents is identical to the requirement SNM applications accessing Solstice EM features, as described in Section 6.3, “Adding an SNM Application to Solstice EM.” That is, you append the location of the Solstice EM library file to `LD_LIBRARY_PATH`, with a command such as the following:

```
host% setenv LD_LIBRARY_PATH /opt/SUNWconn/em/lib:/opt/SUNWconn/lib:${LD_LIBRARY_PATH}
```

SNM applications also have available the $EM_SERVER environment variable, for connecting to a remote MIS, as described in Section 6.3, “Adding an SNM Application to Solstice EM.”

6.6 Solstice EM Applications’ Access to SNM Agents

Solstice EM support for Solstice EM applications accessing SNM agents is illustrated in Figure 6-4.

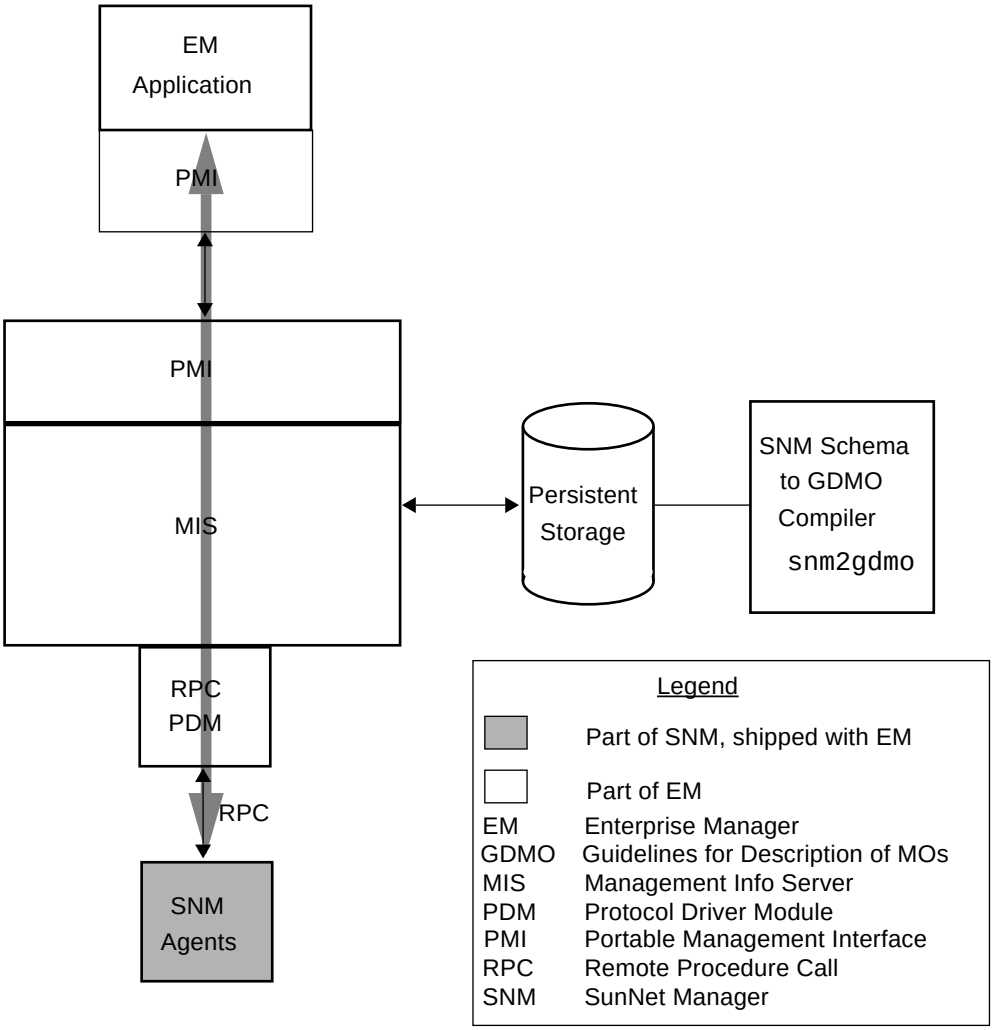


Figure 6-4 Solstice EM Applications Accessing SNM Agents

If you install Solstice EM in a network in which you use SNM, you can use Solstice EM applications to access SNM agents, just as you would access those agents with an SNM application. In fact, a number of the applications shipped

with Solstice EM—including the Data Viewer, Object Configuration tool, Log Manager, and Alarm Manager—have built-in access to or support for SNM agents.

The advantage of using Solstice EM applications to access SNM agents, instead of running SNM, is that the data obtained from the agents becomes part of the MIS. The MIS has a wealth of tools and functions available, in applications such as the Log Manager and the Alarm Manager, and Nerve Center request capability, for manipulating data in ways not possible in SNM.

6.6.1 Configuration

Solstice EM's application support for SNM agents is seamless. That is, it requires no configuration or any other action on your part. The complete set of SNM agents is shipped with Solstice EM, so you can immediately access SNM agents, such as `ping`, `rstat`, or `lpstat`.

6.6.2 Agent Support

All of the agents shipped with SNM are also shipped with Solstice EM. This means that Solstice EM applications have access to all of the RPC agent functions available to SNM applications.

In addition for agents shipped with SNM, Solstice EM provides support for Remote Procedure Call (RPC) agents that you might have written for SNM, or acquired from a third-party vender. The product has an `snm2gdmo` compiler that allows you to convert SNM schema files to GDMO documents, which can be loaded as objects into the MIS. This is described in Chapter 18, "Adding an Object Class Based on an SNM Schema File to the MIS."

6.6.3 Support for SNM Proxy Agents

The complete nature of Solstice EM's support for SNM agents means that Solstice EM supports proxy agents that you might have or might choose to write. Proxy agents are protocol translators, speaking to the MIS with the SNM RPC protocol and speak to managed objects using a different protocol, which might be a proprietary protocol, or a standard protocol such as SNA or X.25. For information on how to write an RPC agent for EM, refer to the "Writing RPC Agents for EM" chapter in the *Solstice Enterprise Manager Application Development Guide*.

SunNet Manager SNMP Proxy Agents

7

Overview	*page 7-1*
SNMP Proxy Agent Operation	*page 7-4*
SNMP Version 2 Support	*page 7-11*

7.1 Overview

The SunNet Manager (SMM) agents provided with Solstice EM include proxy agents to support Simple Network Management Protocol (SNMP) and SNMP Version 2. Proxy agents allow for distribution of polling of SNMP devices to multiple locations in the network.

This chapter describes the configuration and operation of the SunNet Manager SNMP proxy agents. For information on installing the SNM agents and proxies, refer to the *Solstice Enterprise Manager Installation Guide*.

Note – For purposes of this guide, *SunNet Manager* (SNM) refers to the 2.2 or later releases of SunNet Manager, and releases of Solstice Site Manager and Solstice Domain Manager. SunSoft makes no claims of compatibility of Solstice EM with versions of SNM prior to 2.2.

SunNet Manager requests can be launched from the EM MIS using the request-handling capabilities of the EM Nerve Center (as described in Chapter 11, “Building Templates for SunNet Manager Event Requests”). Polling of the managed resource at the specified intervals is handled by the SNM proxy agent rather than the Nerve Center, minimizing network traffic and the polling work required of the MIS.

Proxy agents run on one of the following platforms:

- Sun workstations running SunOS 4.x
- Sun workstations running Solaris 2.x
- PCs running Solaris 2.x/x86.

The Solstice EM MIS communicates with the SNMP proxy agents using the same Remote Procedure Call (RPC) protocol as other SNM agents. The SNMP Version 1 proxy agent (`na.snmp`) communicates with other network devices using the SNMP protocol defined in RFC 1157. The SNMP Version 2 proxy agent (`na.snmpv2`) is discussed below in Section 7.4, “SNMP Version 2 Support.”

The SNMP proxy agent allows you to manage any number of management information bases (MIBs) in which you can define either standard SNMP MIB objects or enterprise-specific objects. The proxy agent uses a SunNet Manager schema file to map objects described in a MIB and SunNet Manager attributes. A schema file is the representation of a MIB used by SunNet Manager.

Communication between the Solstice EM MIS and SNMP devices, using the RCP-based SNMP proxy agents, thus requires three representations of the MIB structure:

- The SNMP MIB on the agent system containing the managed resource
- The SNM schema mapping of the MIB, which resides on the proxy system
- The GDMO and ASN.1 documents, defining the managed object class, which is loaded into the MIS

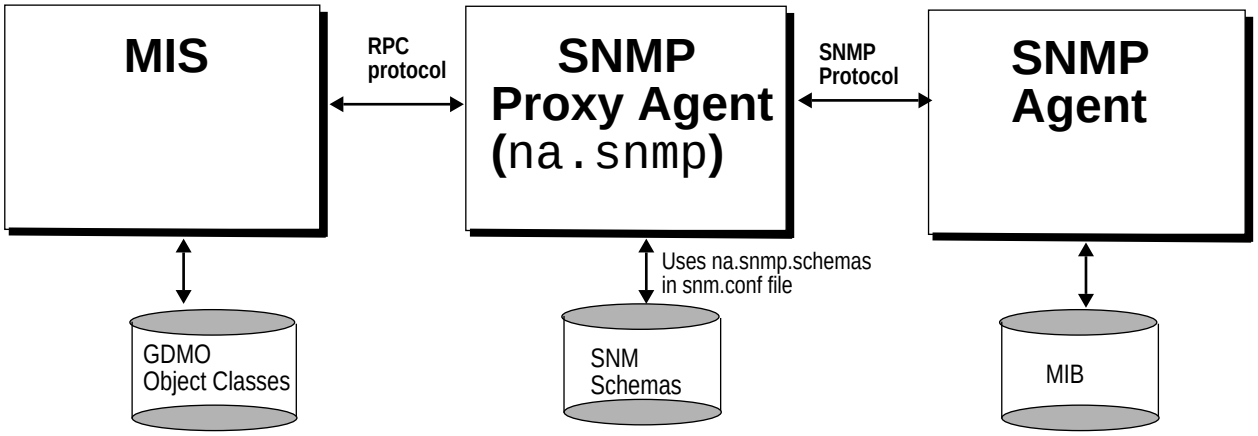


Figure 7-1 MIB, GDMO, and Schema Definitions

Generating GDMO object classes from SNMP MIBs is described in Chapter 17, "Adding a MIB to the MIS." How to generate SNM schema files from MIBs is discussed later in this chapter. To ensure successful operation, there must be an identical mapping of object definitions between the SNMP MIBs, the GDMO documents and the SNM schema files — see Figure 7-1.

The following SunNet Manager SNMP schemas are supplied with Solstice EM:

- `snmp.schema` describes MIB I, as defined by RFC 1156.
- `snmp-mibII.schema` describes MIB II, as defined by RFC 1213.
- `snmpv2-mibII.schema` describes MIB II, as used by SNMP version 2. See the "SNMP Version 2 Support" section for a description of SNMPv2 support. This schema is used only by the `na.snmpv2` agent.
- `sun-snmp.schema` describes the MIB associated with the SNMP agent (`snmpd`) for Sun workstations. This schema file provides MIB II support with Sun enterprise-specific extensions. For more information about the `sun-snmp.schema`, refer to the *Site/SunNet/Domain Manager Reference Manual*.

Except for the two MIB II files (which differ only in the RPC number specified), each of the schema files listed above is a subset of the file that follows it. That is, `snmp.schema` is a subset of the two MIB II files, which are, in turn, are a subset of `sun-snmp.schema`.

The SNMP proxy agent can simultaneously access any of the above-mentioned schemas, as well as other enterprise-specific schemas that you might create. The SNMP proxy agent uses the keyword `na.snmp.schemas` in the `snm.conf` file to locate the directories where the SNMP schema files reside.

The following section describes in detail how the SNMP proxy agent works. Note that many of the operations of the proxy agent are defined by arguments passed in the SNM request or with keywords in the `snm.conf` file on the proxy system. Refer to the `snn.conf` entry in the *Site/SunNet/Domain Manager Reference Manual* for information on the keywords that are related to the SNMP proxy agent.

7.2 SNMP Proxy Agent Operation

The default operation of the SNMP proxy agent is configured by values specified in the `snm.conf` file. These parameters are identified by various keywords. The affect of these settings is described below. The SNMP proxy agent operation is illustrated in Figure 7-2.

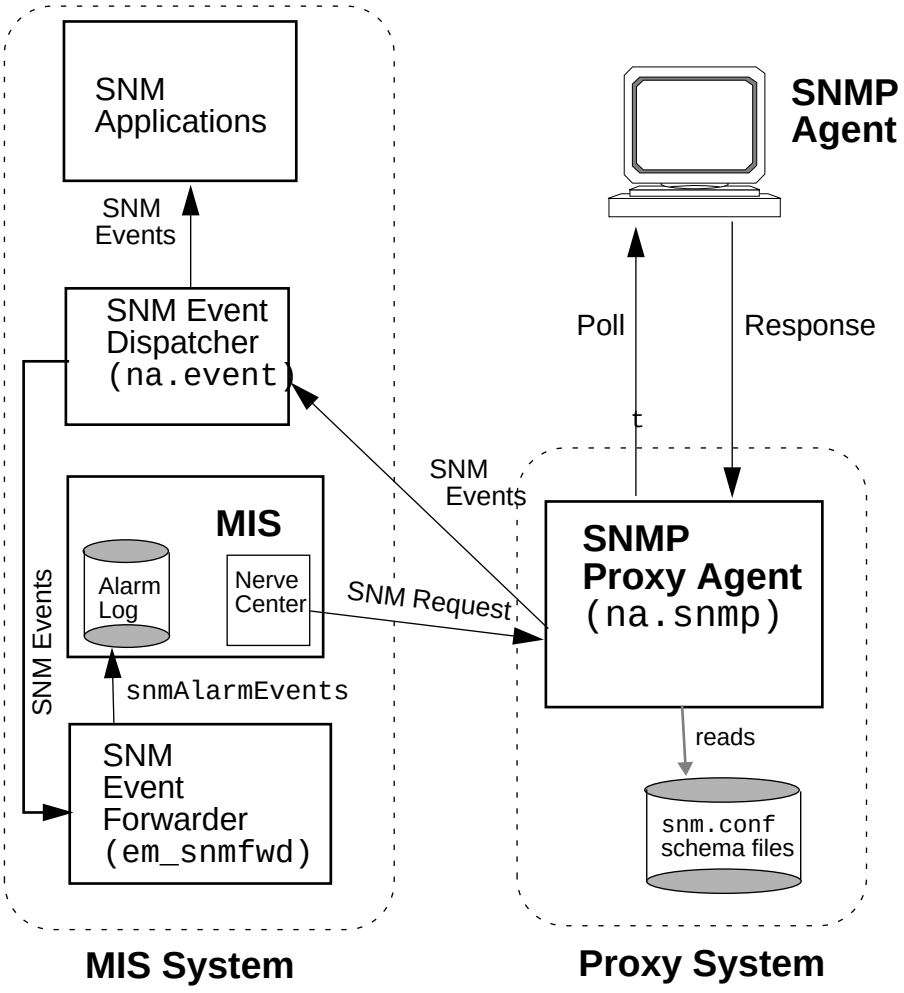


Figure 7-2 SNMP Proxy Agent Operation

When the SNMP proxy agent starts up (normally via `inetd`) it loads all the SNMP schemas located in the directories specified by the keyword `na.snmp.schemas` in the `snm.conf` file on its host system. Only SNMP-related schemas (schemas that contain an `rpcid` keyword value of '100122') are loaded.

When the SunNet Manager SNMP proxy agent receives a request for an SNMP agent on a particular device, it performs the following sequence of operations:

1. It checks whether there are any new or modified SNMP related schema files since the last request. If the proxy agent finds a new or modified schemas in any of the directories specified by the `na.snmp.schemas` keyword in the `snm.conf` file on the proxy's system, it loads the schema file.

2. It passes the request to an existing agent subprocess or forks a new subprocess, if needed, to handle the request. A single subprocess can handle multiple SNMP requests from an instance of a management application. The maximum number of subprocesses that the SNMP proxy agent can fork is set by the keyword `na.snmp.max-subprocs` in the `snm.conf` file. At installation, this value is set to 20. The maximum number of requests that a subprocess can handle is set by the keyword `na.snmp.max-requests` in the `snm.conf` file. At installation, this value is set to 50.

3. It checks whether the request contained any optional arguments. Requests sent by the Solstice EM Nerve Center may include arguments in an SNMP request. These arguments can include:

   a. the name of the schema to be used with the request. If, for some reason, the specified schema does not contain the attribute group specified in the request, the proxy agent attempts to use the schema specified by the keyword `na.snmp.default-schema` in the `snm.conf` file on its host system. At installation, the default schema is set to be:

   - `/usr/snm/agents/snmp-mibII.schema` for Solaris 1.x installations
   - `/opt/SUNWconn/snm/agent/snmp-mibII.schema` for Solaris 2.x installations.

     This schema supports the MIB II definition.

   b. a community name that specifies the SNMP community name the proxy agent is to use when reading or writing attribute values. If no community name is specified, `public` is used for both Get and Set requests.

   c. a request timeout that specifies the number of seconds the proxy agent is to wait for a response to a request sent to the target system. If no request timeout is specified, the proxy agent uses the value specified by the keyword `na.snmp.request_timeout` in the `snm.conf` file on its system. At installation, the value is set to 5 (seconds).

4. The proxy agent then sends an SNMP message to the device and waits for a response.

If the proxy agent is sending a Get request, the proxy sends up to three SNMP requests per reporting interval. (The maximum number of SNMP requests sent is specified by the keyword `na.snmp.max_attempts` in the `snm.conf` file—by default the value is set to 3.) For each SNMP PDU sent, the proxy waits for the specified request timeout for a response from the device. As mentioned previously, the request timeout can be an optional argument in the request. If it is not specified in the request, request timeout is either the request timeout value specified in the SNMP host file for the device or the value of the keyword `na.snmp.request_timeout` in the `snm.conf` file.

If the proxy agent does not receive a response after sending three SNMP requests, it sends a "No response from system" report to the Event Dispatcher (`na.event`) (The keyword `na.snmp.trap-if-no-response` in the `snm.conf` on the proxy system determines whether the proxy agent sends a trap or an error report. At installation, the keyword's value is `true`—send a trap report.) The proxy agent then waits until the next reporting interval to send out another set of SNMP requests. If no reporting interval has been specified in the request, the proxy agent sends out SNMP requests every 30 seconds. If the proxy agent does not receive a response when the last report is due, it sends both an error report and a trap report to na.event if `na.snmp.trap-if-no-response` is `true`.

If the proxy agent is sending a Set request, the proxy waits for the specified request timeout for a response before timing out. There is no attempt to re-send the request. The reason for this is as follows: Because UDP is the transport mechanism, there is no guarantee of message delivery, thus there is no way to determine whether the request or the response to the request was lost. If you do not receive a response from your initial Set request, you should perform a Get request to see whether or not the Set operation was successful.

5. When the proxy agent receives a response from the target device, it sends a report to the Event Dispatcher (`na.event`) on the management machine that initiated the request.

   If the proxy agent does not receive an acknowledgment from the event dispatcher within a specified time, the proxy agent terminates the request. The specified time that the proxy waits for the event dispatcher to acknowledge the report is specified by the `na.snmp.report_timeout` keyword in the `snm.conf` file. At installation, the keyword's value is set to 5 (seconds).

Normally, if the SNMP proxy agent is not performing any requests, it will exit. The keyword `na.snmp.exit-if-no-requests` in the `snm.conf` file allows you to specify otherwise.

7.2.1 SNMP Trap Daemon (`em_snmp-trap`) Operation

Asynchronous or unexpected event notifications (traps) from SNMP agents are handled by the SNMP trap daemon (`em_snmp-trap`), which may run on one or more machines on the network. The daemon listens for incoming traps on the SNMP trap port (port 162). The trap daemon does the following with incoming traps:

- SNMP traps are converted to CMIP event notifications, as specified by the trap daemon's `trap_maps` configuration file, and sent to the MIS.
- SNMP traps are also translated into SunNet Manager traps for use by SNM applications. SNM applications that register with the event dispatcher receive the incoming SNM traps forwarded by the trap daemon. The trap daemon uses a SunNet Manager SNMP trap file, which contains information on enterprise-specific traps.
- You may also specify forwarding of raw SNMP traps to other managers.

Configuration of the EM SNMP trap daemon is described in Chapter 8, "Mapping SNMP Traps to CMIP Event Notifications."

7.3 Schema Files

If you do not already have an SNM schema file for the device you want to manage via the RPC-based SNMP proxy agent (`na.snmp`), use the `mib2schema` utility to convert an existing MIB file for the device. The mib2schema utility supports conversion of MIBs adhering to the following Internet standards:

- RFC 1156 — MIB-I
- RFC 1213 — MIB-II
- RFC 1155 — SMI
- RFC 1212 — Concise MIB definition
- RFC 1215 — Defining traps

To create a schema file for managing devices via the SNMP Version 2 proxy agent (`na.snmpv2`), use the `v2mib2schema` utility to convert the MIB to a V2-compatible schema. The `v2mib2schema` utility is described below in Section 7.4.3, "Using the v2mib2schema Program."

Note – Nested groups or tables are *not* supported in SNM schema files.

You may need to manually edit the resulting schema file produced by `mib2schema`. The areas that are likely to require changes are:

- When `mib2schema` encounters an OCTET STRING, it inserts `-C ???` in place of a format string. If you want to format octet strings in a particular way, search the schema file for occurrences of `-C ???` to replace `???` with the required format string. If a format string is specified, the SNMP proxy agent formats each octet of the attribute value it receives from an SNMP agent before sending the attribute value to a SunNet Manager rendezvous. You may, however, choose *not* to enter any format string. In this case, the contents of the OCTET STRING will be printed as is.

  The format string is the same as the `sprintf(3S)` format argument. Up to 16 octets can be formatted; each byte is sent to `sprintf` as a separate, unsigned character. For example, the format string:

```
%02.2X:%02.2X:%02.2X:%02.2X:%02.2X:%02.2X
```

  causes an OCTET STRING containing a 48-bit Ethernet address to be formatted in standard colon notation (for example, 08:00:20:07:8F:93).

Note – The format string and the length of the OCTET STRING to be formatted must match. All bytes specified in the format string are displayed. If the OCTET STRING is smaller than the format string, unexpected characters may be displayed in the formatted output.

  Note that the `-C` format parameter is only used if the parameter `-T STRING` is specified for the attribute. If the parameter `-T STRING` is specified and `-C` format is not specified, the attribute is displayed as either octets or as a string, depending upon whether the attribute is an octet or display string.

An example of the characteristics string for the `ifPhysAddress` attribute in the `ifStatus` table is shown below:

```
"-N ifPhysAddress -O 1.3.6.1.2.1.2.2.1.6 -T STRING -A RO
-C %2.2X:%2.2X:%2.2X:%2.2X:%2.2X:%2.2X -X equal -F 0"
```

This results in the display:

`ifPhysAddress`=08:00:20:09:A0:D5

- Some SNMP devices cannot return groups or tables with a large number of attributes; this is due to local space limitations. When this happens, the SNMP proxy agent returns an error message that the response is “too big”. This means that very large groups or tables need to be split into smaller groups or tables to be received by the SNMP proxy. `mib2schema` does not automatically split groups or tables. Generally, if a group has more than 15 fields, it is a good idea to split the fields up into smaller groups. You can choose your own name for subgroups.

In addition to the schema file, the `mib2schema` utility produces an object identifier file (with the `.oid` suffix) that contains a table of object identifiers and names. The object identifier file is required only if you want SNMP traps forwarded as SNM traps to SunNet Manager Consoles. For SNM Console support, the contents of the .oid file need to be added to the SNM Object Identifier Database, using the SNM `build_oid` utility. For more information, refer to the `build_oid` entry in the *Site/SunNet/Domain Manager Reference Manual*.

`mib2schema` may also produce a trap definition file (with the `.traps` suffix), depending upon whether traps were specified in the MIB. This file is used for mapping enterprise-specific traps into SunNet Manager trap format for use by the SNM Console. Refer to the *Solstice Site/SunNet/Domain Manager Administration Guide* for more information.

If `mib2schema` cannot determine the key for a table characteristics field in the schema file, it inserts `-K ???` into the schema file.

7.4 SNMP Version 2 Support

Solstice EM provides support for SNMP Version 2 through the SunNet Manager SNMP Version 2 proxy agent (`na.snmpv2`). This section assumes you are familiar with SNMPv2 concepts. Instructions for installing and de-installing SNMPv2 are in the *Solstice Enterprise Manager Installation Guide.*

SunNet Manager provides a proxy agent that supports SNMPv2. This proxy agent allow you to get data and event information from and set attribute values for devices managed through SNMPv2.

There is also an SNMP agent for Sun workstations called the `snmpv2d` daemon. The MIS communicates with this daemon through the SNMP proxy agent. The `snmpv2d` daemon also allows Sun workstations to be managed by other SNMPv2 and SNMP stations. For more information about the `snmpv2d` daemon, see the `snmpv2d` entry in the *Site/SunNet/Domain Manager Reference Manual.*

The following sections discuss the differences between SNMP and SNMPv2. For information about the SNMPv2 configuration files, see the following man pages:

`v2install`(1), `acl.pty`(5), `agt.pty`(5), `context.pty`(5), `mgr.cnf`(5), `mgr.pty`(5), `snmpv2d.conf`(5), and `view.pty`(5).

These man page entries are also provided in hardcopy and AnswerBook form in the *Site/SunNet/Domain Manager Reference Manual.*

Note – When the Discover tool locates SNMP devices on your network, it cannot determine whether the devices support functionality specific to SNMPv2.

7.4.1 SNMPv2 Enhancements

The key enhancements from SNMP to SNMPv2 are in the following categories:

- Structure of Management Information (SMI)
- Protocol operations
- Manager-to-manager capability
- Security

7.4.1.1 Structure of Management Information

The SMI for SNMPv2 is based on the SMI for SNMP. The SNMPv2 SMI provides more extensive specification and documentation of managed objects and MIBs.

Several new data types were created for SNMPv2. These include a 64 bit-counter (`Counter 64`) and the `UInteger32` type which allows representation of integers in the range 0 to 2^{32} - 1.

The SNMPv2 `OBJECT-TYPE` macro includes an optional `UNITS` clause, which contains a textual definition of the units associated with an object. This clause is useful for any object that represents a measurement in units (ex. “seconds”). The `OBJECT-TYPE` macro for SNMPv2 also incudes a `MAX-ACCESS` clause which allows you to specify the maximum level of access.

7.4.1.2 Protocol Operations

SNMPv2 has three new protocol data units (PDU). The SNMPv2 trap PDU works in a way similar to that of the SNMP trap PDU, but it uses the same format as most other SNMPv2 PDUs. This eases the receiver processing task.

A major enhancement for SNMPv2 is the `GetBulkRequest` PDU. This PDU can significantly minimize the number of protocol exchanges required to retrieve a large amount of management information.

The third additional PDU is the `InformRequest` PDU. This is sent by an SNMPv2 manager, on behalf of an application, to another SNMPv2 manager. The PDU provides management information to an application using the second SNMPv2 manager.

7.4.1.3 Manager-to-Manager Capability

Manager-to-Manager operations are supported through the use of the manager-to-manager MIB. This MIB is a set of objects which describe the behavior of an SNMPv2 entity acting in a manager roll. For more information, see RFC 1451.

7.4.1.4 Security

SNMPv2 uses the Secure SNMP (S-SNMP) party concept for security. Improvements over S-SNMP include the elimination of ordered delivery mechanism and simplification of the clock synchronization algorithm. In addition, SNMPv2 introduces the context concept. Contexts provide for more efficient storage of access control and MIB view information. SNMPv2 uses both DES and MD5 for message security and authentication.

7.4.2 SNMPv2 Files

You can install SNMPv2 as an agent (`snmpv2d`), a manager (`na.snmpv2`), or both. The required files are installed as part of the current product. Installation steps are the same for both agents and managers. Before running the v2install script, you will need to create the three configuration files required by the `v2install` script. The files are:

`agents` — contains names of hosts on which the snmpv2d agent will be installed

`mgrs.v1` — contains names of hosts that will be running SNMPv1 managers (`na.snmp`)

`mgrs.v2` — contains names of hosts that will be running SNMPv2 managers (`na.snmpv2`)

See the `v2install`(1) man page, or the `v2install` entry in the *Site/SunNet/Domain Manager Reference Manual*, for detailed information about these files. Procedures for installing (or removing) SNMPv2 software are in the *Solstice Enterprise Manager Installation Guide*.

7.4.3 Using the v2mib2schema Program

A program, `v2mib2schema`, has been included with the current product to allow you to translate your own SNMPv2 MIBs to SNM schema files.

Be aware that SunNet Manager schemas do not have the flexibility of SNMPv2 MIBs, so changes to the MIB may be necessary before `v2mib2schema` can successfully parse it.

Although `v2mib2schema` parses TEXTUAL-CONVENTIONS clauses, it currently ignores them, so later references to the new types will cause syntax errors. See the v`2mib2schema`(5) man page (or `v2mib2schema` entry in the *Site/SunNet/Domain Manager Reference Manual*) for more details.

Mapping SNMP Traps to CMIP Event Notifications

8

Trap Daemon Operation	*page 8-1*
The Structure of SNMP Traps	*page 8-4*
Default Trap Mapping	*page 8-6*
Trap Daemon Behavior When no Mapping Is Provided	*page 8-11*
Customizing the Mapping of SNMP Traps	*page 8-11*
Format of Trap Mapping Records	*page 8-17*
Distributed Trap Handling	*page 8-26*

8.1 Trap Daemon Operation

Simple Network Management Protocol (SNMP) agents have the ability to generate event notifications on their own initiative when certain conditions are detected; these notifications are called *traps*. A Solstice EM daemon — `em_snmp-trap` — listens for incoming SNMP traps for forwarding to management stations. The `em_snmp-trap` daemon can be distributed to multiple machines in the network.

The trap daemon does the following with incoming SNMP traps:

- SNMP traps are converted to CMIP event notifications and sent to the MIS. Like other EM applications, `em_snmp-trap` uses the Portable Management Interface to communicate with the MIS. The trap daemon's mapping of

SNMP traps into CMIP notifications can be customized via entries in the daemon's trap mapping file (`trap_maps`); this is described in Section 8.5, "Customizing the Mapping of SNMP Traps."

- SNMP traps are also translated into SunNet Manager traps for use by SunNet Manager applications. Any SNM application (such as the SNM Console) that registers with the SNM Event Dispatcher (`na.event`) on a manager system receives the incoming SNM traps forwarded by the trap daemon. The trap daemon uses a SunNet Manager SNMP trap file (`snmp.traps`), which contains information for interpretation of enterprise-specific traps. To configure the SNMP trap daemon for use with SunNet Manager, follow the *Solstice Site/SunNet/Domain Manager Administration Guide* guidelines for the SNM trap daemon (`na.snmp-trap`). (The SNM trap conversion functionality of `na.snmp-trap` is a subset of the functionality of the Solstice EM SNMP trap daemon.)
- You can also specify forwarding of raw SNMP traps to other managers. How to configure this capability is described in Section 8.7.1, "Forwarding SNMP Traps to Other Managers." This is configured when you install the Solstice EM trap daemon. The installation script prompts you for the host name and port number of the managers that are to receive the forwarded SNMP traps.

Note – It is unnecessary to run both the Solstice EM SNMP trap daemon and the SunNet Manager SNMP trap daemon (`na.snmp-trap`) on the same system because they listen at the same port (port 162) and the SNM trap-daemon handling is a subset of the functionality of `em_snmp-trap`.

SNMP trap daemon operation is illustrated in Figure 8-1.

For information on how to install `em_snmp-trap`, refer to the *Solstice Enterprise Manager Installation Guide*.

Note – In the current release, `em_snmp-trap` is supported only on Sun workstations running Solaris 2.4 or later software.

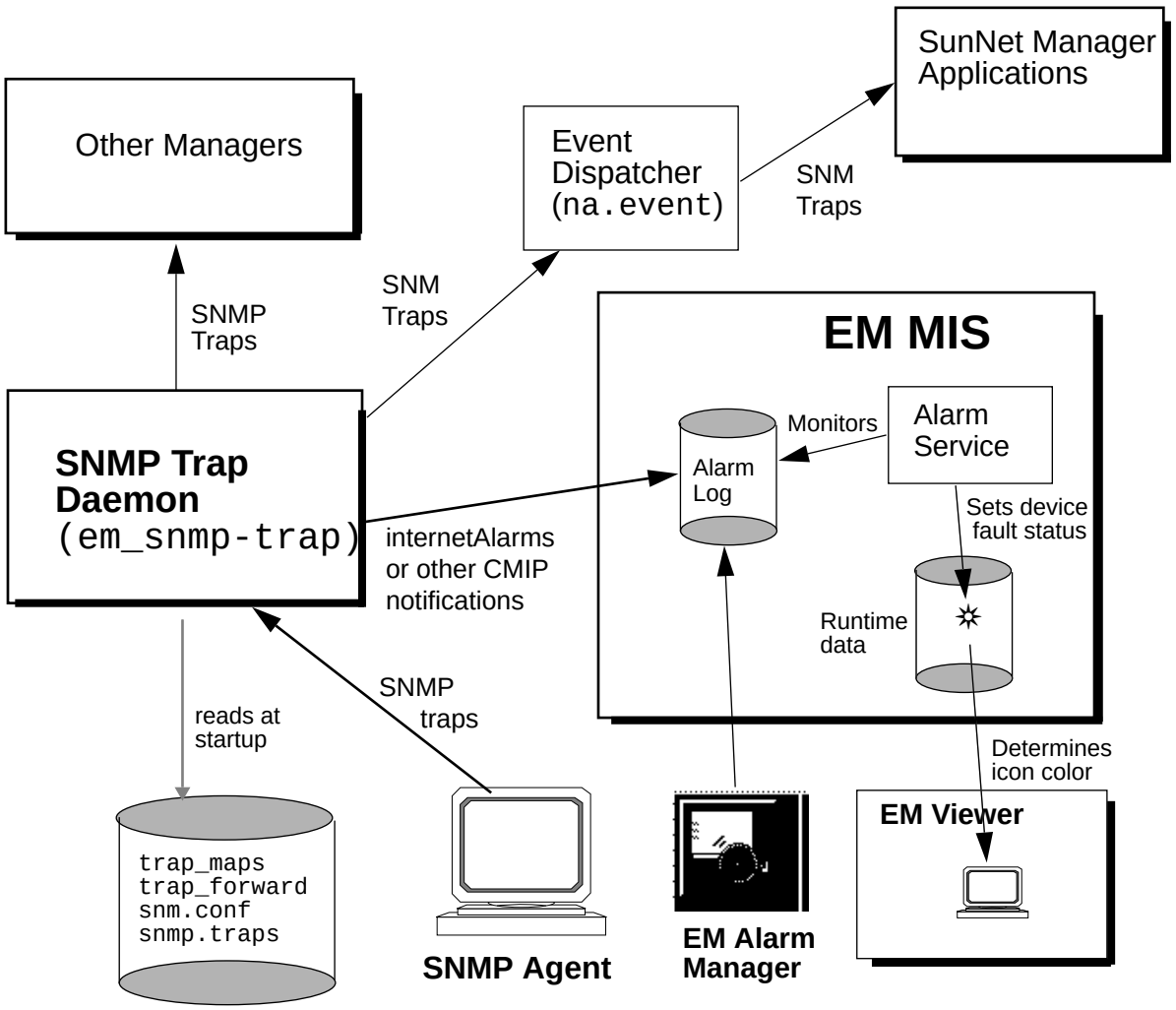


Figure 8-1 em_snmp-trap Operation

There are two ways the SNMP trap daemon can be started or stopped:

- The `em_services` command is used to start and stop all of the EM daemons at once, including the trap daemon and the MIS. (For information on the `em_services` command, refer to the "Management Information Server" chapter in the *Solstice Enterprise Manager Reference Manual*.)

- If you want to start or stop the trap daemon by itself, you can use the `em_trapd` script. There are two commands:

  `em_trapd start` — This command starts the trap daemon.

  `em_trapd stop` — This command stops the trap daemon.

  You should use these commands if you want to start or stop the trap daemon on an MIS machine without also starting or stopping the MIS.

At startup the SNMP trap daemon spawns at least two child processes. One process is responsible for translation of traps to SunNet Manager format and forwarding to the SunNet Manager event dispatcher (`na.event`). In addition, one additional child process is spawned for each of the MIS hosts the trap daemon connects to. (You specify the target MIS machines during installation of the trap daemon.) Each of these processes sets up a CMIP over TCP/IP connection to the MIS on a particular host, and is responsible for conversion of SNMP traps to CMIP event notifications.

The trap daemon's mapping of incoming SNMP traps into CMIP event notifications is determined by user-configurable mapping records in the trap daemon's `trap_maps` file. The `trap_maps` file is an ASCII text file that resides in the `/etc/opt/SUNWconn/em/conf` directory; the trap daemon reads this file whenever it starts.

8.2 The Structure of SNMP Traps

In the discussion of trap-mapping in this chapter we will be making reference to the various fields that comprise the SNMP trap Protocol Data Unit (PDU). The SNMP trap PDU has the fields indicated in Figure 8-2.

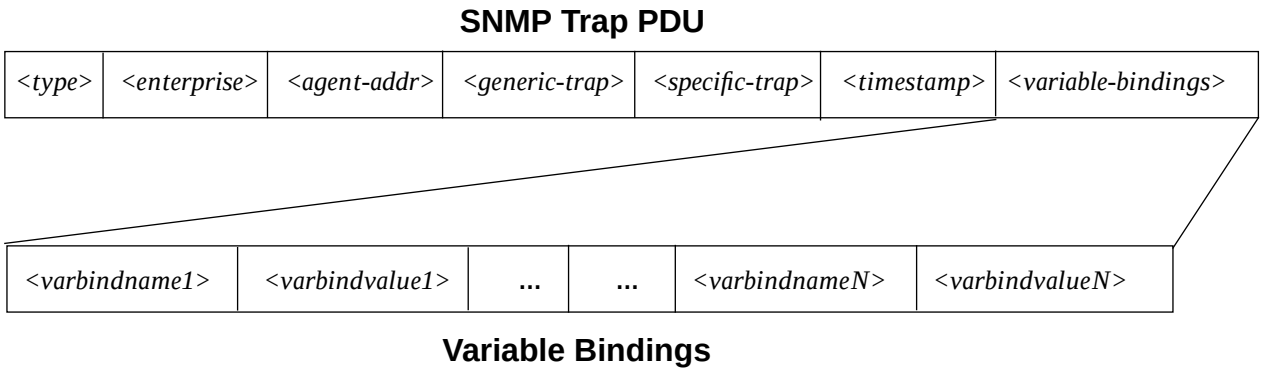


Figure 8-2 SNMP Trap PDU Structure

The fields are:

- <*type*> — Indicates the type of SNMP message. (In this case, it indicates that this is a trap PDU.)
- <*enterprise*> — Indicates the subsystem that generated the trap, as indicated by the `sysObjectID` attribute.
- <*agent-addr*> — This is the IP address of the source of the trap.
- <*generic-trap*> — This is an integer value in the range 0 to 6 indicating the standard trap type. The standard trap types are listed in Table 8-1.
- <*specific-trap*> — A device-specific value providing more information concerning the nature of the event.
- <*timestamp*> — Time between the last reinitialization of the agent system and the time when the trap was generated.
- <*variable-bindings*> — Information that varies depending upon the particular implementation by the product vender. The format consists of attribute / value pairs. Each attribute name is followed by its value.

The <*generic-trap*> and <*specific-trap*> fields contain values that indicate the nature of the trap. The possible values for <*generic-trap*> are described in Table 8-1.

Table 8-1 Standard SNMP Trap Types

Value of <generic-trap>	**Trap Type**	**Description**
0	coldStart	The originating SNMP device is reinitializing itself, typically due to unexpected reboot.
1	warmStart	The originating SNMP device is reinitializing itself, typically due to normal restart.
2	linkDown	One of the agent's communication links is down. The first name/value pair in the variable bindings is the ifIndex for the interface.
3	linkUp	One of the agent's communication links has come up. The first name/value pair in the variable bindings is the ifIndex for the interface.
4	authenticationFailure	The originating system has received a protocol message that has failed authentication.
5	egpNeighborLoss	An External Gateway Protocol peer has been marked down.
6	enterpriseSpecific	Further information about the event is indicated in the <*specific-trap*> field.

8.3 Default Trap Mapping

A default trap-mapping is configured automatically when you install the EM trap daemon. When an SNMP trap arrives, the default method for converting this into a CMIP event notification, as follows:

8.3.1 Default Method for Specifying the Source of the Alarm

When an SNMP trap arrives, `em_snmp-trap` extracts the IP address from the <*agent-addr*> field in the SNMP trap and uses this information to determine if there is an object configured in the MIS to represent that agent system. By default, a `cmipsnmpProxyAgent` object instance in the MIS is used to represent SNMP agent systems.

- If there is a cmipsnmpProxyAgent object in the MIS corresponding to the IP address in <*agent-addr*>, `em_snmp-trap`'s default method of operation is to set the originating system's `cmipsnmpProxyAgent` as the source object instance for this alarm.
- If there is no managed object instance in the MIS corresponding to the IP address of the SNMP trap, the trap daemon attempts to retrieve the hostname of the source agent, but if this is not possible, the trap deamon sets the value of cmipsnmpProxyAgentId to "<*IP-address*>".
- If there are multiple objects in the MIS that have network addresses that match the IP address of the trap, `em_snmp-trap` seeks a match on the SNMP Community String values included in the trap header. For example, an SNMP agent may be a proxy for legacy devices, and the Community String provides information that is used to identify the source device for the alarm.

The trap daemon's default method of mapping SNMP traps into CMIP event notificaitons is determined by a mapping entry in the default `trap_maps` file. The default mapping uses a single scheme to convert traps for any <*enterprise*> identifier. The mapping is based on generic trap type.

8.3.2 Default `perceivedSeverity` Values

Severity is the presumed importance or impact of an event. Following the ITU X.721 standard, Solstice EM uses an attribute in event notifications called perceivedSeverity to represent severity of events. The Alarm Service, which monitors the alarm log, uses the perceivedSeverity value in event notifications to determine the fault status indication for devices. The Alarm Service sets the fault status of a device to the highest perceivedSeverity of outstanding (uncleared) alarms for that device. The fault status of a device, as determined by the Alarm Service, is represented in the Viewer by icon color. The default

mapping of severities to colors is described in Table 8-2. If event notifications are to affect fault status indication, they need to have a perceivedSeverity value.

Table 8-2 Default Color-Coding of Severities

Integer Value	Severity	Default Color
1	Critical	Red
2	Major	Orange
3	Minor	Cyan
4	Warning	Yellow
5	Cleared	No color
0	Indeterminate	Blue

SNMP, however, lacks a systematic concept of the severity of a trap. A function of the trap-mapping is to assign a severity to event notifications based on information in the SNMP trap. The default trap-mapping uses the <*generic-trap*> value to make severity assignments as follows:

- coldStart traps — critical
- warmStart traps — major
- linkDown traps — major
- linkUp traps — cleared
- authenticationFailure traps — warning
- egpNeighborLoss traps — minor
- enterpriseSpecific traps — indeterminate

Note that linkUp traps automatically clear previous linkDown traps from the same router.

You can easily change these severity assignments if you wish. For example, the mapping record for `linkDown` traps in the default `trap_maps` file contains the following line:

```
perceivedSeverity=major;
```

If you want linkDown traps to have a severity of critical, simply edit the trap_maps file to replace “major” with “critical”. This change takes effect when the trap daemon is restarted. Customizing the trap mapping is discussed in

detail in Section 8.5, “Customizing the Mapping of SNMP Traps.” For information on the Alarm Service, refer to the “Alarm Service” chapter in the *Solstice Enterprise Manager Reference Manual*.

8.3.3 *Default probableCause Values*

The default trap-mapping assigns integer values to the `probableCause` attribute in event notifications as follows:

- coldStart traps — 100
- warmStart traps — 200
- linkDown traps — The value from the first variable binding attribute/value pair. By convention, this is the `ifIndex`, indicating the number of the interface.
- linkUp traps — The value from the first variable binding attribute/value pair.
- authenticationFailure traps — 500
- egpNeighborLoss traps — 600
- enterpriseSpecific traps — Set to the specific trap type, in localForm (integer).

  For example, if an enterpriseSpecific trap has a trap specific type of 99, this will be the value of probableCause for the enterpriseSpecificTrap notification generated by the trap daemon.

8.3.4 *Default* `additionalText` *Information*

The default trap-mapping uses the additionalText field in the event notification to contain the following information:

- The enterprise identifier from the trap <*enterprise*> field
- The <*specific-trap*> value (typically 0 for traps other than enterpriseSpecific traps)
- The attribute/value pairs from the trap variable bindings

8.3.5 *Default Event Notification Type*

By default, the trap daemon converts SNMP traps into Solstice EM-specific event notifications as indicated in

The default trap-mapping options ensures that every incoming SNMP trap matchs some trap-mapping record in the `trap_maps` file. This is done by including in the `trap_maps` file a mapping block that specifies a mapping for each <*generic-trap*> type. The default mapping block is of the following form:

```
enterprise 1.3.6.1.4.1
{
<mapping-record-1>
...
<mapping-record-N>
}
```

The identifier "1.3.6.1.4.1" acts like a wildcard in that it matches the <*identifier*> field of every trap.

Mapping blocks can also be added to the `trap_maps` file that use other enterprise identifiers to map SNMP traps generated by agents supplied by particular venders. How to do this is discussed below in Section 8.5, "Customizing the Mapping of SNMP Traps."

8.3.6 Default Location of Information from Trap Variable Bindings

The default mapping scheme loads the attribute/value pairs from the trap variable bindings into the `additionalText` field of the event notification.

8.4 Trap Daemon Behavior When no Mapping Is Provided

This section describes how the trap daemon handles incoming SNMP traps in any situation where no explicit mapping is provided by the `trap_maps` file. This situation could happen, for example, if you delete the default mapping block, or if some of the records within it are deleted.

If an incoming SNMP trap fails to match any entry in the `trap_maps` file, the trap daemon converts the SNMP trap into an `internetAlarm`, in accordance with the ISO-Internet Management Co-existence (IIMC) standard. The IIMC standard defines the use of the ISO/ITU Common Management Information Protocol (CMIP) for integrated management of TCP/IP networks that are managed using SNMP. The IIMC standard prescribes the following:

- Event notification type is internetAlarm.

- The perceivedSeverity value of internetAlarms is set to indeterminate.
- The alarm is posted against the `cmipsnmpProxyAgent` that represents the agent system.
- The attribute/value pairs that comprise the trap variable bindings are loaded into the additionalInformation field of the internetAlarm.

The user-configurable trap-mapping capability of the Solstice EM trap daemon is designed to address these limitations of SNMP and the IIMC standard. This capability allows you to configure the trap daemon to extract information from SNMP traps to create more meaningful alarms, tailored to your particular network management needs.

8.5 Customizing the Mapping of SNMP Traps

The Solstice EM trap-mapping capability is designed to enable you to customize the mapping of SNMP traps to CMIP event notifications.

8.5.1 Overview

SNMP lacks a systematic notion of the severity of an alarm. Also, the IIMC standard lacks a systematic method for determining the source component for a trap within the agent system. The user-configurable trap-mapping capability of the Solstice EM trap daemon is designed to address these limitations of SNMP. This capability allows you to configure the trap daemon to extract information from SNMP traps to create more meaningful alarms, tailored to your particular network management needs.

The trap-mapping activity of the SNMP trap daemon can be customized by editing the `trap_maps` file. Your modifications take effect after the trap daemon is restarted.

8.5.2 Enterprise Mapping Blocks

The trap mapping file consists of blocks of records, with each block identified by the keyword **enterprise**. Each block is in the following form:

```
enterprise <enterprise-object-identifier>
{
<trap-mapping-record1>
  ....
<trap-mapping-recordN>
}
```

The mapping records (one or more) for a given enterprise are grouped within a pair of curly braces. Enterprise object identifiers are specified in dot-dot notation.

Enterprise blocks in the `trap_maps` file select incoming traps for mapping if the the <*enterprise-object-identifier*> in the block heading matches the <*enterprise*> field in the trap. Three important aspects of the enterprise heading:

- **A enterprise block <*enterprise-object-identifier*> does not need to be identical with the <*enterprise*> field of the trap.** For example, if the trap <*enterprise*> identifier is “1.3.6.1.4.1.42.1.2”, this will match an enterprise block with “1.3.6.1.4.1” in the heading. So long as the enterprise block identifier is contained in the <*enterprise*> field, starting at the left, a match will occur.
- **A trap is mapped by the first enterprise block in the `trap_maps` file whose enterprise heading it matches.** Therefore, if you want to add a block with the following enterprise identifier,:

```
enterprise 1.3.6.1.4.1.46
```

be sure to add this block before the default mapping block, which has the following identifier::

```
enterprise 1.3.6.1.4.1
```

If this default enterprise block were the first block in the `trap_maps` file, it selects all incoming traps and any blocks after it in the file would never map any traps. In general, mapping blocks with longer <*enterprise-object-identifier*> strings should be at the beginning of the trap_maps file.

If an incoming trap has an <*enterprise*> field that matches tue <*enterprise-object-identifer*> of one of the blocks in the file, but fails to match any entry in that block, it may still be mapped if it matches another enterprise block later in the file. Traps are checked against the enterprise blocks in the `trap_maps` file sequentially, starting at the top.

The default mapping block should remain at the bottom of the file; traps will never be tested against any blocks beneath the default block since the default block will map any trap.

8.5.3 *Mapping Records*

If the trap daemon determines that an incoming trap matches a mapping block on its <*enterprise*> identifier, the trap daemon then uses the first mapping record within the selected block that matches the trap on the following two fields:

- <generic-trap>
- <specific-trap>

If the trap fails to match any record in the enterprise mapping block on trap type, the trap daemon checks the following enterprise blocks in the file for a possible match. If the trap matches no mapping record in any matching enterprise block, it is mapped into an `internetAlarm`, in the manner described in Section 8.4, “Trap Daemon Behavior When no Mapping Is Provided.”

Mapping blocks can be used to provide a mapping for enterprise-specific traps. For example, if the agent software provided with a server generates an enterprise-specific trap (indicated by a <*generic-trap*> value of 6) with a <*specific-trap*> value of 5 when the machine’s internal temperature exceeds an acceptable threshold, this could be mapped to a CMIP `environmentalAlarm` with `probableCause` and `perceivedSeverity` set to appropriate values. The type of value appropriate to an alarm attribute depends upon the GDMO definition of that event type.

SNMP traps can be mapped to any type of CMIP event notification the MIS knows about. The following event notifications are defined in the MIS by default:

Defined by the ISO X.722 standard

- objectCreation
- objectDeletion
- attributeValueChange
- relationshipChange
- stateChange
- communicationsAlarm
- environmentalAlarm
- equipmentAlarm
- integrityViolation
- operationalViolation
- physicalViolation
- processingErrorAlarm
- qualityofServiceAlarm
- securityServiceOrMechanismViolation
- timeDomainViolation

Defined by the IIMC standard

- internetAlarm

Solstice EM-specific

- snmAlarmEvent
- snmAlarmTrap
- nerveCenterAlarm
- coldStartTrap
- warmStartTrap
- linkDownTrap
- linkUpTrap
- authenticationFailureTrap
- egpNeighborLossTrap
- enterpriseSpecificTrap

The structure of these event notifications is described in the “Standard Event Notifications” appendix in the *Solstice Enterprise Manager Reference Manual.*

You could also create your own custom event types, as described in Chapter 15, “Adding New Event Types.”

When selecting an event type for trap mapping, you will also want to ensure that the selected event type is logged to the alarm log. This will ensure that the incoming traps cause appropriate changes in Viewer icon color. By default, only the following event types are excluded from the `AlarmLog`:

- snmAlarmEvent
- objectCreation
- objectDeletion
- attributeValueChange
- stateChange

8.5.4 How to Customize SNMP Trap Mapping

The steps in configuring `em_snmp-trap` for SNMP trap mapping are as follows:

1. **Collect information on enterprise-specific traps.**
   If you want to add mapping blocks to map enterprise-specific traps, consult the vender documentation for SNMP devices deployed in your network to determine which variable bindings and specific trap values to use for mapping into event notification attributes and to identify components that are sources of events.

2. **Devise your mapping scheme.**
   There are four aspects to the mapping:

   - Creating enterprise blocks, if desired.
     These should be entered above the default enterprise block in the `trap_maps` file.
   - Creating records within a block that map traps to event notification type based on generic and specific trap values.
     This is discussed below in Section 8.4, “Trap Daemon Behavior When no Mapping Is Provided.” You can also create mapping records that instruct the trap daemon to discard matching traps.
   - Creating a mapping for event notification attribute values within mapping records.
     This is discussed below in Section 8.5, “Customizing the Mapping of SNMP Traps.”

- Adding an FDN map to mapping records, if desired.
  An FDN map is a template that is used by the trap daemon to identify the component element that is the source of the event. This is discussed below in Section 8.6.1, “Using FDN Templates to Specify the Source of a Trap.”

3. **Verify that the event types selected for mapping are logged to the alarm log.**
   Use the Log Manager to check the discriminator that selects events for logging to the alarm log. If your selected event type is excluded, you may want to change the log discriminator.

4. **Edit the `trap_maps` file.**
   Using your favorite text editor (such as `vi`), add your mapping elements to the file, with each record conforming to the format shown in Figure 8-3.

5. **Save the file.**
   The file location is `/etc/opt/SUNWconn/em/conf/trap_maps`.

6. **Restart the trap daemon.**
   To stop the trap daemon, enter the following command (as root):

```
#em_trapd stop
```

   Restart the trap daemon by entering the following command (as root):

```
#em_trapd start
```

Note – During this operation, any traps that arrive on the system are lost.

7. **Verify that there are no error messages at startup.**
   When `em_snmp-trap` reads the trap_maps file at startup, it prints error messages if it encounters any parsing errors in the trap mapping table. Verify that no errors occur when `em_snmp-trap` is restarted.

Note – Startup of `em_snmp-trap` is terminated if errors are detected in the `trap_maps` file.

8.6 Format of Trap Mapping Records

Each record in an enterprise block in the `trap_maps` file has the format shown in Figure 8-3

```
  GENERIC-TRAP <generic-trap>
[ SPECIFIC-TRAP <specific-trap> ]
  NOTIFICATION  <alarm-type>  | DISCARD
[ ATTRIBUTE-MAP <attr-name>=<attr-value>; ]
[                                  <attr-nameN>=<attr-valueN>;  ]
 FDN-MAP[ <FDN-template> ];;
```

Figure 8-3 Trap Mapping Record Format

SPECIFIC-TRAP is an optional entry. Typically, <*specific-trap*> will be specified only when <*generic-trap*> is 6, indicating an enterprise-specific trap.

<*generic-trap*> is an integer in the range of 0–6.

SNMP traps are selected for mapping to specified CMIP event notifications only if they match a mapping record on enterprise object identifier and generic and specific trap type. If there is a match on these three values, the trap is converted to the CMIP event notification type indicated by the keyword NOTIFICATION. For example, you might choose to map an SNMP linkDown trap to a CMIP communicationsAlarm, as in the following example:

```
enterprise 1.3.6.1.4.1.42
{GENERIC-TRAP 2
NOTIFICATION communicationsAlarm
ATTRIBUTE-MAP
    perceivedSeverity=varbindvalue3;
    probableCause=varbindvalue2;
FDN-MAP
    internetClassId={interfaces 0}/internetClassId={ifTable 0}/internetClassId={ifEntry
varbindvalue1};;}
```

Table 8-3 Example: Mapping an SNMP `linkDown` Trap

The type of event notification specified by NOTIFICATION in a mapping record can be any CMIP event notification which the MIS knows about. Alternatively, you can use the keyword DISCARD to indicate that a matching trap is to be discarded by the trap daemon.

A mapping for one or more event attributes can be entered after the keyword ATTRIBUTE-MAP. <*attr-name*> must be a valid attribute for the event type specified by <*alarm-type*>.

<*attr-value*> can be one of the following:

- A constant
  In the following example

```
ATTRIBUTE-MAP
     perceivedSeverity=critical;
     probableCause=localValue : 100;
     additionalText=”Network memory usage greater than 80%”;
```

  the severity of the alarm is set to critical and a string constant is passed as additionalText.

  If a constant is used for <*attr-value*>, it must be of a type appropriate for the particular event notification attribute in proper ASN.1 string format.

- A trap variable binding value
  For example:

```
probableCause=varbindvalue2;
```

  In this example, “varbindvalue2” indicates the value of the second variable binding name/value pair in the SNMP trap.

- The keyword $ALLVARS
  This keyword is used only with a text field. The $ALLVARS keyword specifies that the text field is to receive the following information:

  - The <*enterprise*> identifier of the trap
  - The <*specific-trap*> value
  - All of the attribute/value pairs comprising the trap variable bindings

For example:

```
additionalText = $ALLVARS;
```

An example of output from this mapping would be an additionalText field that looks like this::

```
enterprise = 1.3.6.1.4.1.42 , specificTrap = 1 , ifNumber = 5 ,
ifType = other , ifIndex = 3 , ifDescr = THIS IS A STRING
```

Each mapping of an attribute to a value must end in a semicolon.

The EM Alarm Service requires that an alarm have a perceivedSeverity value (an integer value in the range of 0 to 5) in order to map to Viewer icon colors that represent the importance of a network event. The valid severities and their default associated icon colors are listed in Table 8-2.

The mapping of severity to displayed color is controlled by the EM Nerve Center; this mapping is user-configurable via the Request Designer application. Refer to the “Request Designer” chapter in the *Solstice Enterprise Manager Reference Manual* for more information.

The interpretation of <*specific-trap*> values for enterprise-specific traps depends upon the particular implementation by the product vender. You will need to consult the product documentation for SNMP devices in your network to determine an appropriate mapping to CMIP event notifications.

SNMP traps that do not match any record in the mapping file on <*enterprise*>, <*generic-trap*>, and <*specific-trap*> are mapped to default IIMC internetAlarms as described in Section 8.4, “Trap Daemon Behavior When no Mapping Is Provided.”

Some useful considerations in customizing the `trap_maps` file:

- ATTRIBUTE-MAP is an optional entry. However, you will want to include a mapping for at least the attributes defined as REQUIRED attributes in the GDMO definition of the alarm type specified as NOTIFICATION.
- Two semicolons are required to mark the end of each record.
- Case is ignored for all keywords in the `trap_maps` file.
- Comments can be interspersed in the file. Any line that begins with a pound sign (#) as the first character in the line at the left is treated as a comment.

Note – If you specify a mapping of attributes for internetAlarms, the only attributes that will be included in the alarm are the required attributes and any optional attributes whose mapping you have specified.

8.6.1 Using FDN Templates to Specify the Source of a Trap

When SNMP traps arrive at the MIS system, `em_snmp-trap` extracts the IP address from the <*agent-addr*> field in the SNMP trap and uses this information to determine if there is an object configured in the MIS to represent that agent system. By default, a cmipsnmpProxyAgent object instance in the MIS is used to represent the agent system. If there is a cmipsnmpProxyAgent object in the MIS corresponding to the IP address, `em_snmp-trap`'s default method of operation is to convert the trap to an `internetAlarm` and set the originating system's `cmipsnmpProxyAgent` as the source object instance for this alarm. For example, if a linkDown trap arrives from router `bigguy` with IP address 129.144.55.67, `em_snmp-trap` sets the following as the fully distinguished name (FDN) for the alarm:

```
/systemId=name:"gatoloco"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2 4 1
0}/cmipsnmpProxyAgentId="bigguy"
```

Figure 8-4 Sample FDN for cmipsnmpProxyAgent Object Instance

This might not be the object instance that represents the specific component on which the event occurred. To point the event notification at the particular component object, you can specify a template to build an FDN that points to the specific component that is the source of the event, such as an interface on a router or a circuit in a switch. This template is indicated in the trap mapping record by the FDN-MAP keyword.

8.6.1.1 Understanding FDNs and RDNs

An FDN specifies an absolute path through the Management Information Tree (MIT) to an object instance. The FDN specifies the path to an object instance by indicating its "containment" relationships. Just as an object instance is a

software entity that represents a particular network resource, the containment relationship between objects is used to model physical containment relationships, such as that between a router and its interface cards.

The format of an FDN is as follows:

```
/<naming-attribute1>=<value1>/<naming-attribute2>=<value2>/<naming-attribute3>=<value3>
```

Figure 8-5 FDN Format

You can think of an FDN as analogous to an absolute path to a file in a UNIX file system. Each *<naming-attribute>=<value>* pair is a relative distinguished name (RDN) that specifies an object instance relative to the object specified by the portion of the FDN to its left.

An FDN consists of a concatenation of RDNs, with a slash (/) separating the RDNs. An RDN specifies an object instance only relative to the object which contains it. For example, the following RDN specifies the `internetSystem` group within our cmipsnmpProxyAgent for `bigguy`:

```
internetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}
```

Thus, if this RDN is appended to the FDN for bigguy's cmipsnmpProxyAgent in Figure 8-4, the result is an FDN that points to the internetSystem object instance for this SNMP agent:
:

```
/systemId=name:"gatoloco"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2 4 1
0}/cmipsnmpProxyAgentId="bigguy"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}
```

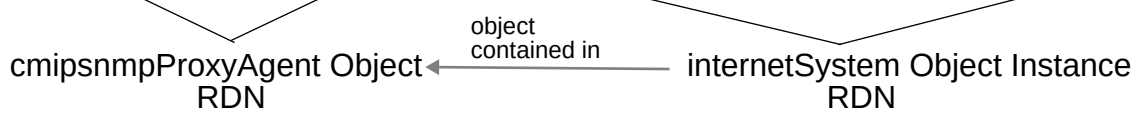


Figure 8-6 Sample FDN for internetSystem Group Object Instance

In this example the particular MIS system where this cmipsnmpProxyAgent object instance resides is indicated by `systemId=name:"gatoloco"`.

systemId, cmipsnmpProxyAgentId, and internetClassId are examples of *naming attributes*. A naming attribute is an attribute whose value is an identification that is unique within the object that contains it (for example, a unique interface index within a router or a unique hostname within a subnet).

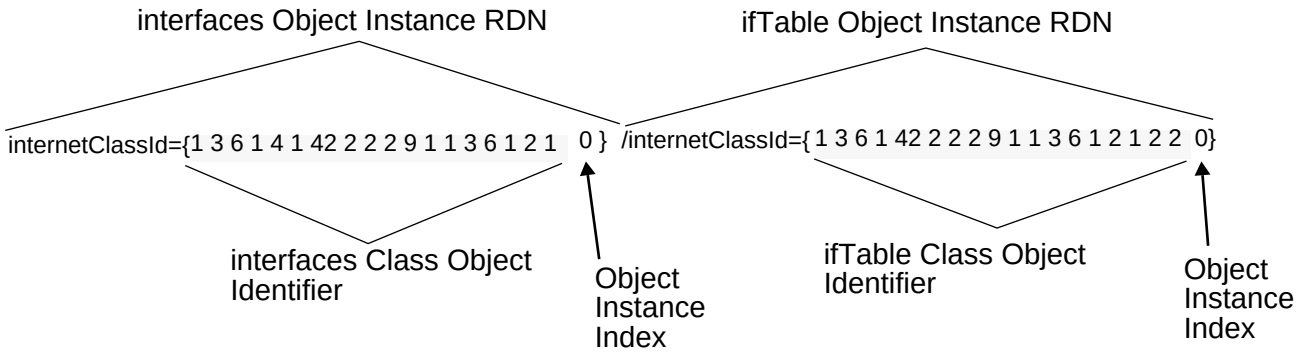


Figure 8-7 Sample ifTable FDN

The value of a naming attribute is an Object Identifier that specifies the object instance. The Object Identifier for the instance consists of the Object Identifier value for its class plus an object instance index, as illustrated in Figure 8-7. Scalar objects have an index of 0. For example, because there is only one interfaces object instance under the cmipsnmpProxyAgent, the object instance index for the interfaces object will always be 0.

8.6.1.2 Building FDN Templates

The function of an FDN template is to enable `em_snmp-trap` to compose an FDN that represents the target component within the agent system.

The FDN template is preceded by the keyword FDN-MAP (see Figure 8-3). FDN templates can follow one of two formats:

Standard Format:

```
<naming-attribute>={<object-class-name>  <instance-index>}/<naming-attribute>={<object-class-name>  <instance-index>}
```

Absolute Format:

```
/<naming-attribute>=<value>/<naming-attribute>=<value>
```

where <*value*> is either a constant or specifies a variable binding value. A variable binding value is specified by expressions of the form `varbindvalue1`, `varbindvalue2`, and so on.

- The *standard* FDN template lacks an initial slash at the far left. This indicates that the FDN built from the template is to be appended to the FDN that specifies the `cmipsnmpProxyAgent` object instance FDN. The object instance representing the target component is thus contained under the cmipsnmpProxyAgent object. The standard format enables `em_snmp-trap` to more finely specify the component within the agent system represented by the default `cmipsnmpProxyAgent` object. The example in Figure 8-8 shows an FDN template with the standard format. Class names are used in specifying attribute values.
- The *absolute* format FDN template specifies the full FDN path to the target component from root. The absolute format FDN template is distinguished by the presence of an initial slash at the left. The initial slash indicates to `em_snmp-trap` that it is not to append the FDN built from the template to the default `cmipsnmpProxyAgent` FDN. Class names cannot be used in specifying attribute values. Constants or variable binding values are used to indicate attribute values. For example:

```
/systemId=name:"bigguy"/myClassId=varbindvalue3
```

The example in Table 8-4 has the FDN template shown below.

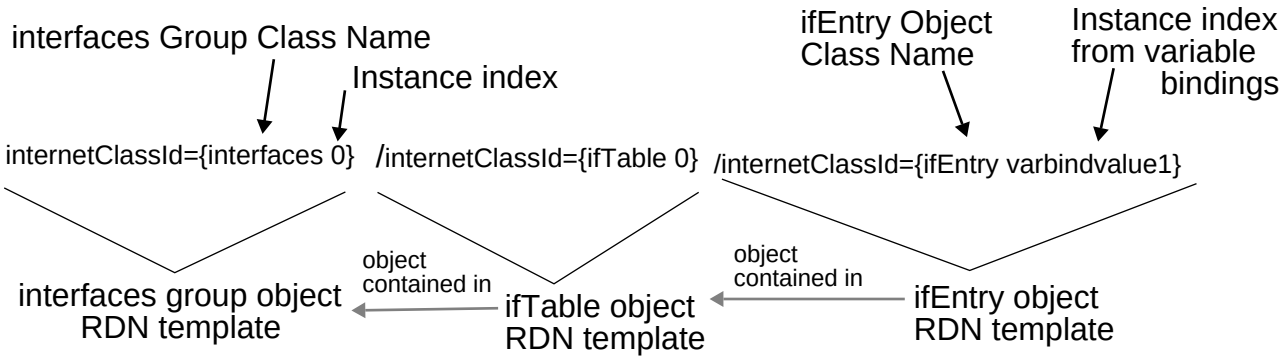


Figure 8-8 Sample FDN Template

In this example, “interfaces”, “ifTable”, and “ifEntry” are object class names. `em_snmp-trap` converts the class name to its corresponding Object Identifier value and appends the <*instance-index*> value. For example, “interfaces” will be replaced with “1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 2” and “0” — the instance index — will be appended. Because there is only one interfaces object instance under the cmipsnmpProxyAgent, the object instance index for the interfaces object is always 0. The resulting Object Identifier is:

```
{1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 2 0}
```

This Object Identifier specifies the SNMP `interfaces` group object instance. Similarly, “ifEntry” is converted to its Object Identifier value and the specific interface index obtained from the trap <*varbindvalue1*> field is appended. For example, this might be an index of 2. The resulting Object Identifier specifies the object representing the particular interface card.

An SNMP trap variable binding field used in a template is specified in the following form:

`varbindvalue`*N*

where *N* is the number of the variable binding you want to use.

8.6.2 Mapping Restrictions

The following restrictions apply in mapping SNMP traps to CMIP event notifications:

- If you are specifying a mapping from a variable binding value to an event notification attribute value, the ASN.1 value of a CMIP notification attribute must be the same as the value of the SNMP trap field. Data type conversions are not allowed.
- If you are mapping variable bindings to event notification attributes, there must be a one-to-one mapping for SNMP variable bindings. Multiple variable bindings cannot be concatenated to form a single notification attribute.The exception to this is the use of the $ALLVARS keyword to specify mapping of all variable bindings into the additionalText field of the event. An example is shown above under Section 8.6, “Format of Trap Mapping Records.”

8.7 Distributed Trap Handling

The SNMP trap daemon can be distributed to machines in your network other than workstations running the MIS. The names of MIS machines for forwardng of event notifications are specified when the trap daemon is installed.

However, if the trap daemon is to connect to the MIS on another machine, you will need to do the following:

1. **Edit the `/var/opt/SUNWconn/em/conf/EM-config` file on the MIS machine.**
   Add the name of the trap deamon machine to the entry EM_ACCESS_TRUSTED_HOSTS. For example, if you have installed the trap daemon on the machine empress, the `EM-config` file on each MIS machine it is to connect to should have the following line:

```
EM_ACCESS_TRUSTED_HOSTS:    empress
```

2. **Restart the MIS, if necessary.**
   If the MIS is already running on the target MIS machine, use the `em_services` command to restart the MIS.

8.7.1 Forwarding SNMP Traps to Other Managers

The Solstice EM SNMP trap daemon has the ability to forward raw (unprocessed) SNMP traps to other managers. When you install the SNMP trap daemon, you are prompted for the hostname and port for each SNMP manager that is to receive forwarded SNMP trap PDUs. This information is stored in the `trap_forward` file in the `/etc/opt/SUNWconn/em/conf` directory; this is an ASCII text file that the `em_snmp-trap` daemon reads whenever it starts.

The `trap_forward` file has the following format:

```
MIS_HOSTS: <MIS-host1>,<MIS-host2>
SNMP_HOSTS: <mgr_hostname1>:<port1>, <mgr_hostname2>:<port2>
```

The MIS_HOSTS line contains the names of the machines where an MIS is running that the trap deamon is to connect to. Each MIS machine names are separated by commas.

The SNMP_HOSTS line contains the hostname and port number for each SNMP manager that is to receive the raw SNMP traps. Entries for multiple managers are separated by commas.

Nerve Center Overview 9

Overview	*page 9-1*
Guide to Nerve Center Documentation	*page 9-2*
Request Terminology	*page 9-3*
Nerve Center Operation	*page 9-6*

9.1 Overview

The Solstice EM product provides the MIS support and applications that enable you to detect conditions in a network and take action in response. Collectively, this MIS functionality and related tools are referred to as Solstice EM's Nerve Center.

The Nerve Center is comprised of the following parts:

- Request facilities in the Solstice EM MIS that handle the sending and receiving of requests and generate request-driven polling
- Request Condition Language (RCL) — A script language that allows you to express what you want to do to monitor and perform threshold-checking in your network. You use this language in the Request Designer to build sets of instructions called "conditions." Conditions are the building blocks of request templates.

- A set of applications that allow you to create, save, and debug request templates, and launch and monitor requests. These applications include:
  - The Request Designer, which allows you to create request templates that are the basis of requests.
  - The Requests tool, which allows you to launch requests against specific network elements. Requests tool also allows you to view, stop, and examine running requests.
  - Facilities in the Request Designer (and via the `em_ncimport` and `em_ncexport` command-line utilities) that allow you to export request templates and their components to ASCII file for easy replication of request components from one MIS to another.
  - The `em_debug` utility, which provides facilities for debugging of request templates.
- A Nerve Center Interface Library that provides programmers the means to write applications to create, launch, and retrieve information from requests. The Nerve Center Interface Library is described in the "Nerve Center Interface Library" chapter in the *Solstice Enterprise Manager API Syntax Manual*.

9.2 Guide to Nerve Center Documentation

Information on Nerve Center components can be found in the following places in the Solstice EM documentation:

- This chapter describes the Nerve Center request terminology and operation.
- Chapter 12, "Request Designer," describes the features and usage of the Request Designer application.
- Chapter 13, "Request Condition Language," describes the components of the Request Condition Language used to build Nerve Center request templates.
- Chapter 14, "RCL Functions," describes the built-in functions that can be used in building RCL conditions.
- The Requests tool is described in the "Viewer" chapter in the *Solstice Enterprise Manager Reference Manual*.
- The Nerve Center interface library is described in the *Solstice Enterprise Manager API Syntax Manual*.
- Chapter 10, "Building Request Templates," provides step-by-step guidance on building and debugging request templates using the Request Designer, RCL, and `em_debug`.

- Chapter 11, “Building Templates for SunNet Manager Event Requests,” provides guidance in building templates that initiate SunNet Manager event requests.
- The request templates shipped with Solstice EM are described Chapter 3, “Managing Devices.”

9.3 Request Terminology

Knowledge of the following terms is useful in understanding how requests work in Solstice EM. Terms in italics are defined within this section. Terms themselves start at the left margin of the page.

action

What can happen when a *transition* from one *state* to another occurs as the result of a *condition* returning a value of true. The *Nerve Center* supports three types of actions: execution of a condition, execution of a Unix command, or sending a mail message. The default is that no action be taken upon a transition. You can have any number of actions in any combination of types. See *severity.*

attribute

A formal characteristic of a managed object. The *Nerve Center* polls for attributes as part of a *request*. In a request, you can poll only for those attributes that are defined in the MIS. Solstice EM is shipped with a set of attributes defined in GDMO documents, which are compiled into the MIS.

condition

A set of instructions written in the *Request Condition Language* (RCL). A condition serves two functions in *requests*:

- A single condition can be used to define when a request targeted at a specific set of (one or more) managed objects undergoes a *transition* from one *state* to another (or loops back to the same state). You must have exactly one condition associated with each transition. Where more than one transition out of a given state is possible, each defined by a distinct condition, the *Nerve Center* evaluates the conditions in the order they are entered in the *request template*.
- A second function of a condition is as an *action*, taken in response to a transition. A condition is one of three types of action (the others being the sending of mail and the invocation of a Unix command). Multiple

conditions can be invoked as actions resulting from a single transition. When multiple actions result from a given transition, Nerve Center executes them in the order entered in the request template.

When used to test whether to make a transition, the value of the condition is the value returned by the last statement in the condition. The value is either true (nonzero) or false (zero). When used as an action, the value returned by the condition is not used by the *Nerve Center.*

Nerve Center

That portion of the MIS that polls and receives notifications from the agents of managed objects. The main job of the Nerve Center is to start and maintain *requests.*

notification

A message initiated by an agent, reporting a change in the state of a managed object. The set of notifications that the *Nerve Center* knows about — and therefore the notifications that you can receive through a request — are defined in GDMO documents that are compiled into the MIS.

A notification is a superset of an alarm, which is an extraordinary event, such as an equipment failure. Solstice EM has a default set of alarms. Through the *Request Condition Language*, requests can subscribe to receive incoming notifications and take appropriate action.

polling

The process by which the *Nerve Center* in the MIS periodically obtains data from an agent according to the specifications of a *request template.* The goal of *polling* is to obtain the *attributes* of a managed object.

poll rate

Specifies the delay until the first poll and the interval between successive polls. The Request Designer associates a name and a number (of seconds) with each poll rate. This application offers a ready-made list of poll rates and allows you to create your own poll rate.

Upon the conclusion of a poll rate interval, the *Nerve Center* begins to test the *conditions* for the *transitions* that lead from a given *state.* Each state has a poll rate and a *severity* associated with it.

request

The series of activities through which the *Nerve Center* polls for the *attributes* of managed objects and receives *notifications* from the agents of managed objects. A request is based on a *request template* and is typically targeted at a specific managed object. (A request that subscribes to receive event notifications may

be launched without being targeted at a specific object.) Each request is made up of multiple states, with, potentially, multiple transitions between those states.

You launch requests in the Requests window, which is started from the Viewer. Once started, a request remains alive until you delete it, in the Requests tool.

Request Condition Language (RCL)

A script language, similar to the C programming language, used to create *conditions*. The RCL is described in Chapter 13, "Request Condition Language," and Chapter 14, "RCL Functions."

request template

A state transition diagram consisting of one or more states, transition paths between states, and actions to be undertaken when a transition from one state to another occurs. Any number of requests can be created using a single request template. Each request template has a name, which you can use in other applications to start or view a request. When you start a request, you are typically applying a specific request template to a specific managed object.

severity

Severity has two meanings in the context of the Request Designer:

- Each *state* has a severity and a *poll rate* associated with it. A severity is made up of three items: a name, a number, and a color. As a value attached to a state, a severity has a meaning that applies only within the Request Designer itself — for example, by indicating the appropriate color to use in representing a state in the graphical State Machine display. The Request Designer offers a ready-made list of severities and allows you to create your own severities.
- A severity can also be attached as a *value* to a Nerve Center alarm posted to the alarm log, using one of the *RCL* alarm log commands (`alarm()`, `alarmOi()`, `alarmStr()`). This Nerve Center alarm severity is reflected in the color of the icon for the managed object as that object is displayed in the Viewer.

state

A description of a managed object at a point in time with respect to a *request*. At any given moment, a request, reflecting the target managed object, is in some state defined in that request or is undergoing a *transition* between states. While in a state, a request repeatedly tests the *conditions* associated with each transition leading from that state. *The interval between tests of a condition is determined by the state's poll rate.* In addition to a poll rate, each state has a

severity associated with it as well as a name and a description. Between any two states, there are, potentially, multiple transitions in both directions, with a single *condition* associated with each transition.

One state is required state: the ground state. This state must have a *severity* of "normal". Other states are of your choosing. There is no limit to the number of states, though a maximum of nine states (including Ground) can be displayed in the Request Designer's graphical state diagram display.

transition

The change from one *state* to another, which occurs when the *condition* associated with a transition evaluates to true. From one state, a *request* can make transitions to multiple states, including the state from which the transition started. Within a pair of states (or from and to itself) there can be multiple transitions in each direction. Each transition is associated with one condition. Where there are multiple transitions, the *Nerve Center* evaluates conditions associated with those transitions in the order the transitions are entered in the *request template*.

When a transition occurs, depending on the specifications in the request template, a set of *actions* might be performed — to set variables or send notifications, for example.

variable

A *Request Condition Language* (RCL) expression that can potentially take on different values, or have values assigned to it. RCL allows you to specify variables as you need them when you create a *condition*. It assigns types to these variables at runtime. The RCL also has a set of system variables, which you will find useful in a variety of different *request templates*.

9.4 Nerve Center Operation

9.4.1 How a Request Gets Information

A request can get information in the following ways:

1. It can *poll* for the attributes referred to in the conditions to be tested from its current state. The Nerve Center sets the values of the attributes before it "awakens" a request.
2. It can *subscribe* for event notifications. The Nerve Center sets the values of variables related to the notification before it "awakens" the request.

3. It can use a combination of (1) and (2). If a state uses both subscription and polling, its conditions can tell which of them "woke" the request by checking the value of `$messType`, described in Section 13.6, "Message Types."

9.4.1.1 Where and When a Request's Notifications Arise

An event that "awakens" a request can arise in either of two ways:

1. A notification initiated by an agent.

Agents originate notifications on their own. A request can subscribe to particular notifications. That is, it can ask the Nerve Center to pass it certain notifications. For example, a request might ask to receive all authentication-failure SNMP traps. A request for a specific managed object might ask to receive notifications that concern that object. After it has subscribed, the request has nothing to do but wait until a notification arrives.

The subscription specifies whether the request will receive:

a. All notifications of a particular type

or

b. All notifications that refer to a particular object

After a request subscribes to a notification, it receives all such notifications, whenever they arrive, regardless of the request's current state.

Requests can also use CMIS filters to select which notifications they want to receive.

2. A response to a poll

Some information is provided by agents only on request from management stations, that is, when *polled*.

A condition can refer to attributes of a managed object. Every reference to an attribute in a condition is interpreted as an implicit request to poll for that attribute. When a request goes through a transition and arrives at a state, it initiates periodic polls for the values of all the attributes it needs.

For each state, the template specifies a periodic poll rate. The poll rate specifies the delay until the first poll and the interval between successive polls.

When a request goes through a transition, in effect it sends the following request to the Nerve Center:

"Cancel any previous poll requests I made. Set a timer to go off every <*n*> seconds from now and every <*n*> seconds thereafter. (The number of seconds is the poll rate for the request's current state.) Whenever that much time has elapsed, poll to get me the values of the following attributes. When you have obtained values for all these attributes, wake me."

The list of conditions for a poll contains all attributes mentioned in any of the conditions leading from the current state and also all attributes mentioned in any of the actions that accompany transitions from the current state.

A state must be awakened by retrieving information from the MIS. Otherwise, a request hangs in a state forever. A condition as simple as the "jump" condition (`$x=map; true;`), supplied with Solstice EM, is sufficient to awaken a state. In this case, the value of the `map` attribute is accessed.

9.4.1.2 When Information From Managed Objects Can Arrive

Event-related information from managed objects are of two types:

Notifications that come from an agent arrive at unpredictable times. After a request has subscribed for certain types of event, it receives notification of all events that match its subscription. They are forwarded at once, regardless of the state the request is in. What the request does with them, or whether it even looks at them, depends on the conditions the request tests in its current state.The RCL offers the following functions that allow you to subscribe to events.

- `subscribe()`
- `subscribeOi()`
- `subscribeFilter()`

Messages in response to a poll arrive according to the schedule set by the current state's poll rate. The Nerve Center notifies the request when it has assembled the values for all the requested attributes. A message in response to a poll arrives no sooner than the number of seconds specified in the poll rate, but possibly later.

9.4.2 Variables and Attributes in a Request

All requests built from the same template use the same names for variables and attributes. Values associated with these names are specific to an individual request. Any of the conditions that the template uses can refer to those names.

When a template is created, the Request Designer automatically scans the definitions of all the conditions mentioned in the template for references to variables and attributes.

9.4.2.1 Attributes

When a condition contains the name of an attribute, the Nerve Center automatically looks up the name in the Solstice EM MIS's MetaData Repository (MDR). Provided the attribute occurs only once in the MDR, using to its name is sufficient to identify it. If the same attribute name occurs in different places in the MDR, the name can be qualified by including the name of the GDMO document in which it is declared. Refer to Section 9.4.5, "Specifying the Objects to be Polled," on page 9-12 for more information.

You cannot use a request to perform sets on attributes.

An ampersand precedes an attribute (or variable) name if you need to pass the address (rather than the value) of the attribute to an RCL function, such as `defined()` or `extract()`. For example:

```
NOT (defined(&sysUpTime);
```

9.4.2.2 System Variables

Certain names have standard meanings. If a condition refers to one of those names, the Nerve Center supplies the appropriate information. For example, if a condition needs to retrieve the time that a notification arrived, it can use the

system variable `$eventTime`. If it needs to retrieve the message type of the current notification, it can use `$messType`. See Chapter 13, “Request Condition Language” for a list of system variables and a list of the possible values of the `$messType` variable.

9.4.2.3 *User Variables*

Any condition can create a variable by using the name of the variable to the left of an = sign. The name of a variable must begin with the dollar sign ($). (The dollar sign distinguishes the names of variables from the names of attributes, because attribute names do not start with $.) For example, if you want the variable `$count` to be the number of consecutive times that confirmation of object X has been missing, some initial condition should contain a statement such as

```
$count = 0;
```

and some other condition should contain

```
$count = $count+1;
```

Using the name `count` is sufficient to declare it.

A variable must first be assigned a value before it can be compared to another variable or attribute. For example, a condition that has the statement

```
sysUpTime<$last_sys_up_time
```

should not be called if the variable `$last_sys_up_time` has not yet been assigned a value.

All the variables mentioned in a request template share a common name space. That is, any condition used in a request can see or set any of the request’s variables. However, no request has access to variables in another request.

9.4.2.4 How Notifications and Poll Responses Are Delivered

When a notification arrives, the Nerve Center sets the values of all the system variables involved in a request. Similarly, when a poll response arrives, the Nerve Center sets the values of all the relevant system variables and attributes.

At the point when a request starts testing its conditions, the Nerve Center has already set the values of variables or attributes it needs. However, if the request uses *both* subscription and polling, it should check the `$messType` system variable to determine which type of event "woke" it. Following a notification or poll response, the values of attributes are those from the previous notification or poll response, if there was one.

9.4.3 Where and When a Condition is Evaluated

A condition is evaluated independently for each managed object that is the target of a request when the request is in a state that tests the condition, and

- A poll response arrives, or
- An event notification arrives.

A notification to which the request subscribes can arrive at any time. A response from a poll cannot arrive until:

- The state's poll interval has expired, and
- The Nerve Center has returned the response from the poll.

9.4.4 Action at a Transition

The following subsections describe actions you can specify in your request templates.

9.4.4.1 Supported Actions

To invoke actions at a transition, you must select one or more of the actions from the Solstice EM list of supported actions. The types of supported actions are summarized in Table 12-1. For an action you can specify any combination of the supported actions.

Note – A condition can be invoked as an action. Indeed, conditions are a much more powerful way of defining actions than the use of Unix commands or mail. This permits the power of RCL, together with its access to the variables defined within an individual request, to be combined in writing individualized actions.

9.4.4.2 *Logging an Event*

A log object (described in the "Log Manager" chapter of the *Solstice Enterprise Manager Reference Manual*) stores selected event records. Each log object has a *discriminator construct*: a proposition that the MIS uses to decide whether to send a notification to the log or ignore it. In effect, each log object can adopt its own criteria for receiving notifications.

The Request Condition Language (RCL, see Chapter 14, "RCL Functions") provides alarm logging functions (`alarm()`, `alarmOi()`, `alarmStr()`, and `sendEvent()`) to write a log record to the log object `alarmLog`. Log records written as a result of the alarm logging functions meet the criteria of the default discriminator construct. The `alarm()`, `alarmOi()`, and `alarmStr()` functions log only nerveCenterAlarms. The `sendEvent()` function can be used to log other kinds of notifications (such as internetAlarms or communicationsAlarms). The events that are logged to a particular log depend upon the discriminator construct for that log.

9.4.4.3 *Forwarding an SNMP Trap*

The predefined condition called `InternetTrap` invokes the `SendTrap` function to send a trap notification to the host identified by the user-defined variable `$Host`. The condition uses the system variables `$eventType` and `$eventInfo`, which are set automatically upon receipt of an incoming trap. The definition assumes that `$Host` has previously been assigned the appropriate value, for example during the transition from the ground state.

9.4.5 *Specifying the Objects to be Polled*

You can write templates that specify a particular managed device as the target of the request. That request template would only be useful for managing that particular device, however. In most cases you will not want to write a new

template for each target object. Typically you will define a template that can be used to manage objects of the same type — a certain type of router manageable via SNMP, for example. The same template can then be launched against all the devices that share the same management characteristics (for example, routers that support the same SNMP MIB).

Using the $pollfdn system variable in templates helps you to define templates that can be targeted at different objects of the same type. In the following example the objective is to create a template that can extract the attributes under the `snmp-mibII system` group, such as `sysDescr`.

When a template is launched against a device selected in the Viewer, `$pollfdn` is initially set to the first distinguished name (FDN) in `$pollFdnSet`. The $pollFdnSet variable contains the set of FDNs that point to all the managed objects that have been configured for the selected device. For example, if the device is configured as SNMP and RCP manageable, $pollFdnSet will contain both an FDN for the cmipsnmpProxyAgent for the device, which represents the device's SNMP agent, and an FDN pointing to the RPC proxy table, which contains objects representing the Remote Procedure Call (RPC) agents the device supports.

For example, the host bigguy is an SNMP agent system and the device has been so configured in the MIS. When a request is launched against `bigguy`, the `cmipsnmpProxyAgent` is one of the objects in $pollFdnSet. The distinguished name (FDN) pointing to this object is the following:

```
/systemId=name:"gatoloco"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2
4]/cmipsnmpProxyAgentId="bigguy"
```

The `cmipsnmpProxyAgent` is the object in the MIS that represents the agent on the system being managed. The various attribute groups or tables accessible via the SNMP agent are represented by objects "contained" in the `cmipsnmpProxyAgent` object. Before it can establish polls for the object of interest, the request needs to do the following:

- select the cmipsnmpProxyAgent from among the objects in $pollFdnSet and set this object instance into a variable
- append to this object instance a further specification of the object it wishes to poll (for example, the `interfaces` group object)
- set $pollfdn to point to this object

In a fully distinguished name (FDN)

```
/<naming-attribute>=<value>/<naming-attribute>=<value>/<naming-
attribute>=<value>
```

each <*naming-attribute*>=<*value*> designation is called a “Relative Distinguished Name” (RDN), and each RDN designates an object, which is said to be “contained in” the object designated by the RDN to its left in the path. The initial slash at the left represents the local root of the Management Information Tree (MIT). In the example above, the `cmipsnmpProxyAgent` for `bigguy` is contained in the MIS on the system `gatoloco`. “Containment” relationships are reflected in the path to the object specified in the FDN.

For example, to set the `$pollfdn` to point to the `snmp-mibII internetSystem` group under the cmipsnmpProxyAgent object for bigguy, the template must concatenate the Relative Distinguished Name (`RDN`) for `the internetSystem` group to the cmipsnmpProxyAgent. The RCL appendRdn() function allows you to do this. The appended RDN is a string that specifies the <*naming-attribute*>=<*value*> pair for the `system` group. The affect of this `appendRdn` operation on our request launched against the system `bigguy` is to change the value of `$pollfdn` to the following:

```
/systemId=name:”gatoloco”/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2
4]/cmipsnmpProxyAgentId=”bigguy”/InternetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}
```

For examples that illustrate the use of appendRdn(), and the $pollFdnSet and $pollfdn variables, see Chapter 10, “Building Request Templates.”

Note – If you reset the value of $pollfdn in a condition to change the target of polling, the new value does not affect until after the request transitions to a different state.

9.4.6 Alarm Logging and the Alarm Service

The RCL alarm-logging functions — `alarm()`, `alarmStr()`, and `alarmOi()` — allow you to generate nerveCenterAlarm which, by default, are logged to the AlarmLog. Alarms logged to the alarm log can be viewed and cleared in the Alarm Manager.

The AlarmLog is also, by default, monitored by the Alarm Service. The Alarm Service is a module in the MIS that controls the fault status color in the Viewer. Fault status is an attribute of topology nodes, which are represented by icons in the Viewer. Each topology node has an attribute `topoNodeMOSet`, which points to a set of managed object instances (MOIs), representing the agents configured for the particular device.

The Alarm Service associates an alarm posted to the AlarmLog with a topology node if and only if that alarm is posted against one of the managed objects in the `topoNodeMOSet` for that topology node. The Alarm Service tracks the `perceivedSeverity` values of the alarms that are posted against each topology node. The highest perceivedSeverity value of uncleared alarms determines the fault status of the device. Thus, if a critical alarm is logged against host bigguy, the router icon, by default, turns red. If several minor alarms are then posted against bigguy, these do not cause the router icon to turn cyan unless the critical alarm has been cleared.

A request can use the `alarmOi()` function to clear a previous alarm it has posted against a device by posting an alarm with a severity of cleared. To determine which previous alarm the “clear” alarm is clearing, the Alarm Service looks for a match on `probableCause`. When a request uses alarm() or alarmStr() to log alarms, the probableCause value of the nerveCenterAlarm is set to the severity value indicated for that alarm. For example, if you use:

```
alarmStr(1,”Device is Down”);
```

to post a critical alarm, this alarm will have a probableCause of 1. To clear this alarm, you must use the alarmOi() function and set the probableCause value to 1, to match the critical alarm the request is clearing.

For example:

```
$info = StrToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{1,clear,\"Device is up\",3,1}");
alarmOi($save_pollfdn,$info);
```

When a request is launched at a target device in the Viewer, the $pollFdnSet RCL system variable for that request points to the managed objects that are comprised in the topoNodeMOSet for the selected topology node. The $pollfdn system variable is also initially set to point to the first managed object listed in $pollFdnSet.

The alarm() and `alarmStr()` functions posts a nerveCenterAlarm against the managed object that the $pollfdn variable points to at the time when the alarm-logging function is called. If you have reset the $pollfdn variable to point to an object other than one of those comprised in $pollFdnSet in your request, you should either reset $pollfdn to an appropriate managed object before calling `alarm()` or `alarmStr()` or else use the `alarmOi()` function, which enables you to specify the managed object against which the alarm is to be posted.

For example, if you reset $pollfdn to point to the internetSystem group under the cmipsnmpProxyAgent object, you can retain the original pointer to the cmipsnmpProxyAgent in a variable $snmpfdn, and then post an alarm using that variable:

```
alarmOi($snmpfdn,1);
```

For more information on the Alarm Service, refer to the "Alarm Service" chapter in the *Solstice Enterprise Manager Reference Manual*. For more information on alarm-logging, refer to Chapter 10, "Building Request Templates."

Building Request Templates 10

Overview	*page 10-1*
Designing Request Templates	*page 10-16*
Requests Based on Polling	*page 10-18*
Polling RPC Agents	*page 10-27*
Requests Based on Event Subscription	*page 10-33*
Debugging Your Templates	*page 10-40*

10.1 Overview

The functions and applications provided by the Solstice EM Nerve Center enable you to detect conditions in a network and take action in response.

Requests are the series of activities through which the *Nerve Center* polls for the *attributes* of managed objects, receives *notifications* from the agents of managed objects, and takes appropriate action in response. A request is typically initiated when a *request template* is launched from the Viewer's Requests tool. The same request template can be used as the basis of multiple requests, targeted at different managed objects. The Request Designer is an application that allows you to create request templates.

A request can obtain information about managed resources in several ways:

Direct polling

Polling is the obtaining of data from an agent, at intervals specified in the request template. The goal of polling is to detect specified attributes of a managed object. When a request polls for the current values of object attributes, this activity is carried out directly by sending GET requests to the MIS.

Indirect polling

Polling for device attributes can be offloaded to RPC agents or proxy agents. The periodic checking of thresholds on devices is carried out by the agent or proxy, not the Nerve Center. Indirect polling is initiated when a request generates a certain type of action — a SunNet Manager (SNM) event request — targeted at an RPC agent or proxy. Indirect polling is more efficient than direct polling for checking thresholds on large numbers of devices because it minimizes the polling burden on the MIS and distributes the polling activity to an RPC proxy agent closer to the device being polled. (Building request templates that initiate SNM event requests is discussed in Chapter 11, “Building Templates for SunNet Manager Event Requests.”)

Subscribing for events

A message generated by an agent on its own initiative when a specified event is detected on a managed resource is an *event notification*. For example, a CMIP agent may generate a communicationsAlarm when a remote connection goes down.

A request template can listen for specified event notifications; this is called an event *subscription*. The RCL subscription functions are used to specify the type of event notification the request is to receive. The desired events can be specified by managed object, event type, or through the use of a CMIS filter. The subscribed-for event notifications are forwarded to the request by the MIS when they arrive. For example, if you are using the Solstice EM SNMP trap mapping capability to convert router SNMP `linkDown` traps to CMIP communicationsAlarms, you might design a template that subscribes for communicationsAlarms and then takes appropriate action when communicationsAlarms are received.

The MIS also generates event notifications; and, thus, a request could be defined that listens for specified event notifications from the MIS. For example, a request could subscribe for `objectCreation` event notifications generated when client applications connect to the MIS.

Combining polling and event-subscription

These different ways of obtaining information about managed resources can be combined in the same request. For example, a request might initiate an SNM event request and, at the same time, subscribe for snmAlarmEvents from the target object. If an SNM agent or proxy detects the occurrence of the threshold specified in the SNM event request, it emits an SNM event notification, which arrives at the MIS as an `snmAlarmEvent`. The request may wait for the arrival of snmAlarmEvents, and take appropriate action when this happens. (A sample template that subscribes for snmAlarmEvents is described in Chapter 11, “Building Templates for SunNet Manager Event Requests.”)

10.1.1 Building Blocks of Requests: States, Transitions, and Conditions

A request template is comprised of a finite number of *states* and *transitions* between states. States in a request are used to represent the presumed state of a managed resource, as indicated by information available to the request. For example, if a request is polling to determine if a device is up or down, then, clearly, two states you would want in such a request would be Up and Down, to represent these possible states of the device. If the request is in the Up state when the device fails to respond to a poll, then you will want the request to move to the Down state to indicate this change in the state of the device. A move from one state to another in a request is called a *transition*.

A state can also loop back to the same state in a transition. For example, if a request is in the Up state and a poll indicates the target device is still up, passing this test may cause the request to loop back to the Up state.

In designing a template you will need to define when each transition should take place; this is done by selecting a *condition* that defines when the transition is to occur.

A condition is made up of instructions written in Request Condition Language (RCL). You assign a name to a condition when you save it in the Request Designer. Saving a condition stores it in the MIS for use in templates. A

condition can be used to define when a transition from state A to state B is to occur. If you want a request to move from Up to Down, for example, when a target system is not reachable, you will need to use a condition that tests for that circumstance to define the transition.

Nerve Center checks conditions in the order they occur in the template. To use a condition to define when a transition is to occur, the condition must evaluate to true or false when checked by the Nerve Center.

A condition that defines a transition is evaluated only if the current state is "woken up." The current state is woken up only if one of the following occur:

- The request has received an incoming event notification.
- The condition attempts to access a an attribute.

For example, if a request subscribes for coldStartTraps and the condition defining a transition tests for the arrival of an event, an incoming coldStartTrap will "wake up" the state and the condition is then evaluated.

10.1.2 State Machine Diagrams

A representation of a finite set of states, and the possible paths between those states, is a *finite state machine.* Before you start building a request template, you may wish to draw a state machine diagram in which you show the various device states you want to represent, the paths between them, the types of information that the request is to make use of to determine when to make each transition, and the *actions* that you want the request to take when it makes a transition (for example, logging an alarm or sending an e-mail message).

A state diagram shows how a request template works. Figure 10-1 shows a very simple, yet valid, example that illustrates request-related concepts.‘

Note – The "severities" that attach to template states in the Request Designer do not control the fault status indication (icon color) of devices in the Viewer; severities of states only affects the color attached to states in the Request Designer's graphical display. Fault status color of devices in the Viewer is determined by the alarms logged against those devices. If you want the fault status color of icons to change when a request transitions from one state to

another, you can control this using RCL alarm-logging functions. This is discussed below in Section 10.1.4, “Controlling Fault Status Color in the Viewer.”

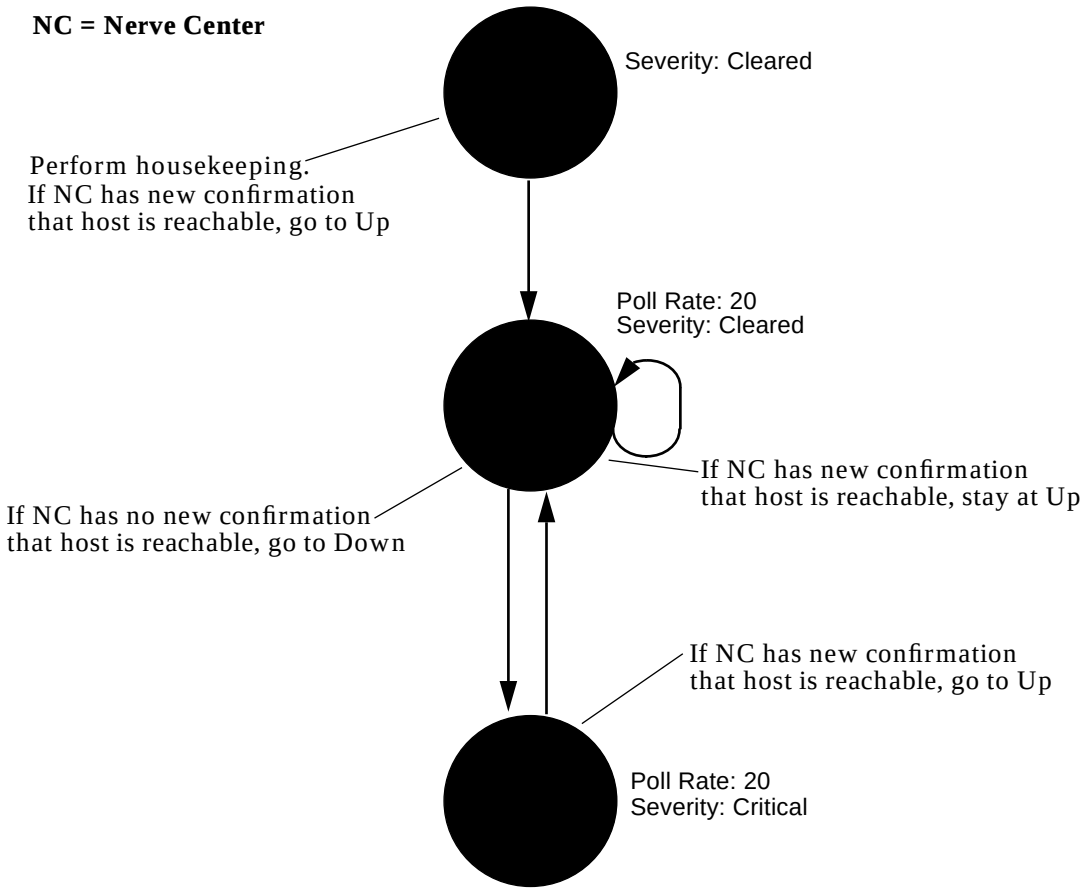


Figure 10-1 Request Example with Poll Rates and Severities

Figure 10-2, following, is a state diagram for an example request template that adds the “Missed” and “InitComplete” states to the previous example template.

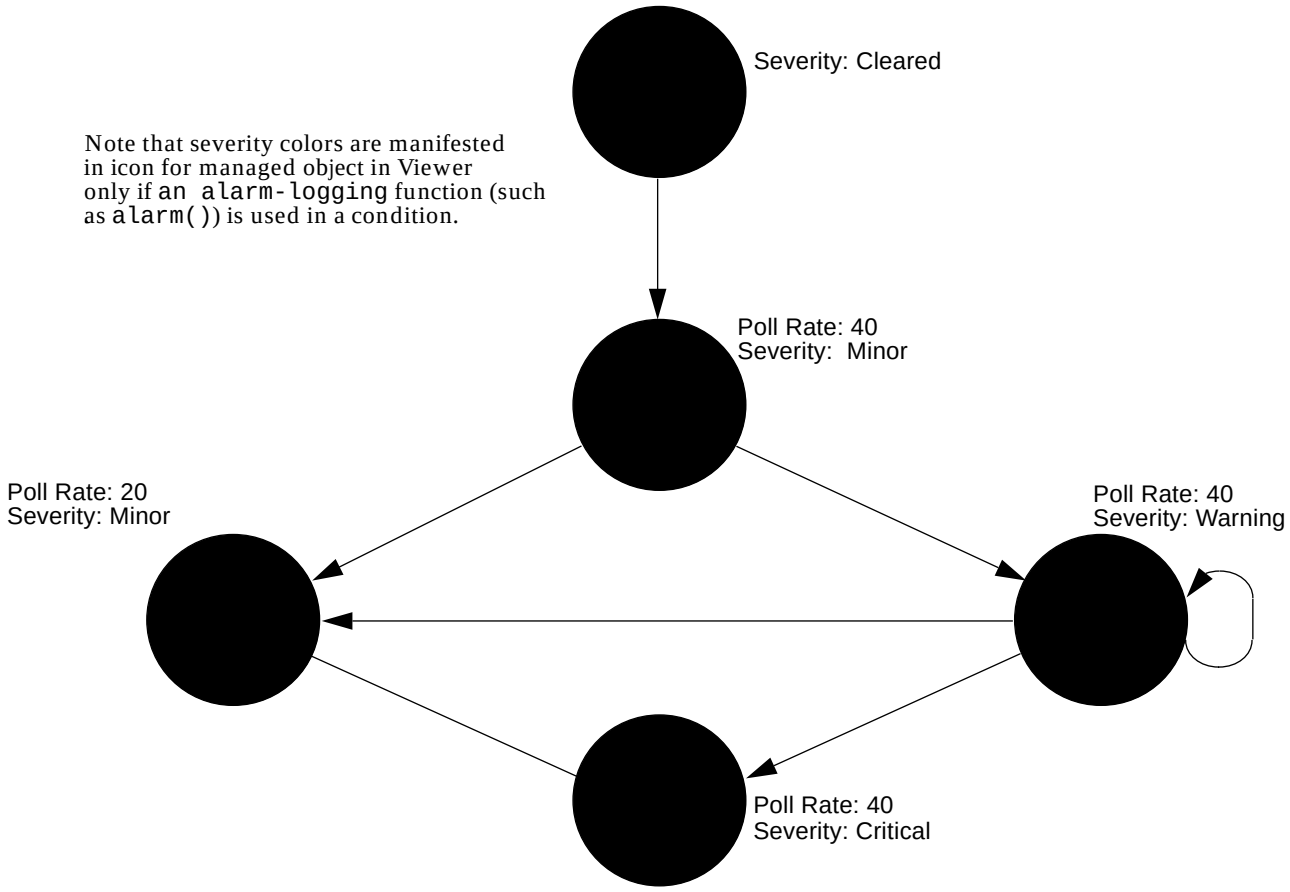


Figure 10-2 Request Example with Poll Rates and Severities

Figure 10-3, following, shows the same template as shown in Figure 10-2 with the conditions associated with each transition.

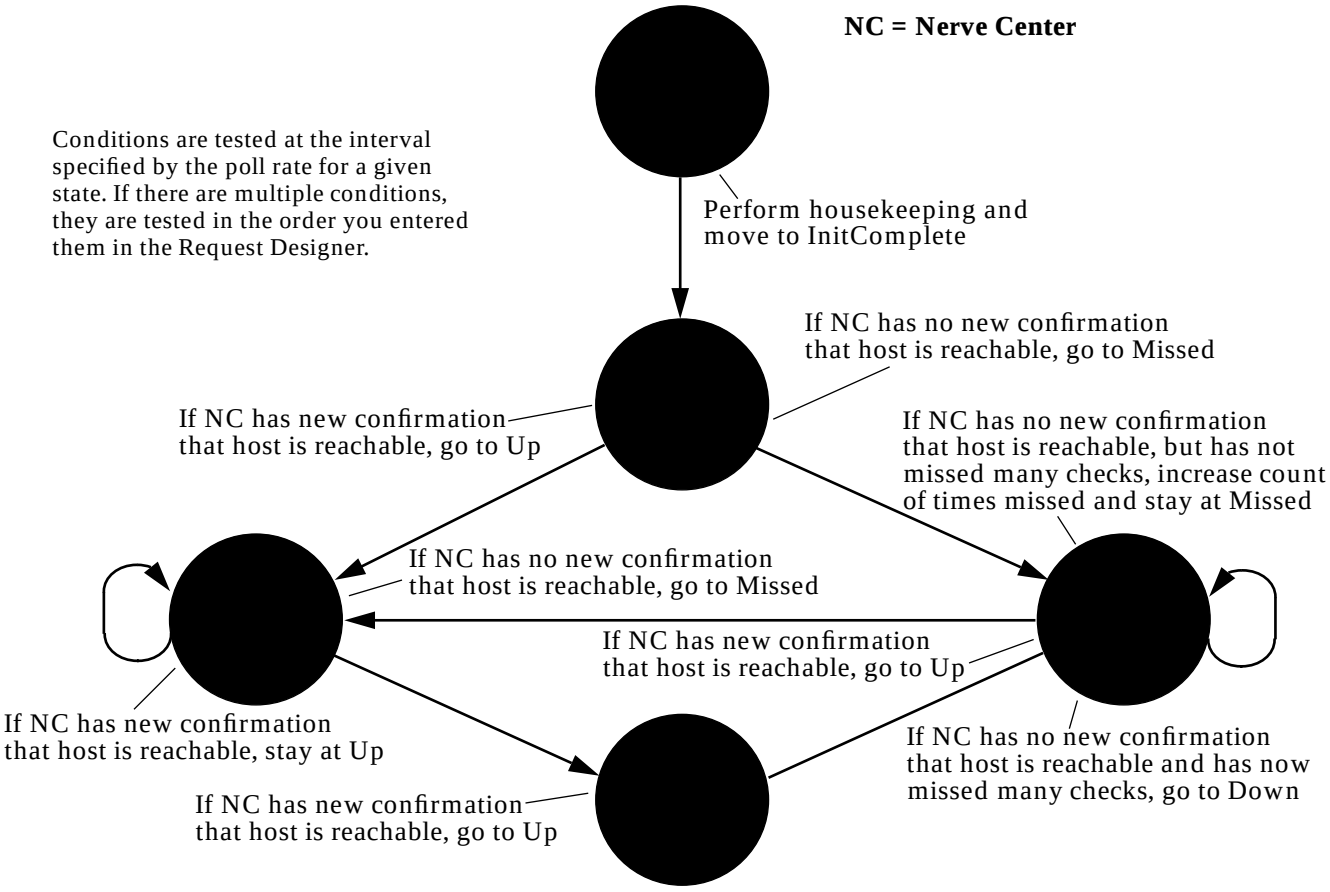


Figure 10-3 Request Example with Conditions

In Figure 10-3, the conditions are described in English. In an actual request template, you define conditions in the Request Condition Language.

10.1.3 Sample Request Template

Figure 10-4 illustrates the workings of a sample request template `IsSnmpSystemUp`, which is shipped with Solstice EM. This template does what its name suggests: tells you whether the SNMP daemon is running on a target machine. You can use this template as the basis for additional templates. For example, you might add a "Missed" state, as illustrated in Figure 10-2.

Condition names are user-created; they are not pre-defined in RCL. Conditions are saved in the MIS under separate names to make it possible for you to use the same condition in multiple requests. The `AlarmCriticalOi` condition, for example, logs a nerveCenterAlarm with a severity of critical. This is a condition that you may want to use in a wide variety of request templates.

In the Request Designer's graphical display, the "IsSnmpSystemUp" request template is displayed much as it appears in Figure 10-4, without the condition code. You can infer much of the essential work in building a template from the figure:

- create states
- define transitions between states
- specify a condition for each transition
- specify actions to take place for each transition, if needed

The IsSnmpSystemUp template has five states: Ground, Error, Poll, Up, and Down.

10.1.3.1 Setting the Target Managed Object

The initialization of the template takes place in the transition from Ground to Poll. The first step in initializing the request is to set the target of the poll.

When you launch a request against a device selected in the Viewer, Nerve Center places all of the managed objects configured for the device into the system variable `$pollFdnSet`. "Managed objects" are internal representations in the MIS of the agent capabilities supported by the device. If you configured Discover to search for RPC-based SunNet Manager agents when you populated your MIS, devices that have both SNMP and RPC agents are configured in the MIS to indicate this. Fully distinguished names (FDNs) pointing to these managed objects in the MIS are thus loaded into the $pollFdnSet variable when you launch requests against such a device.

Nerve Center uses another system variable — `$pollfdn` — to hold the target of the request. When a request is launched, Nerve Center initially sets $pollfdn to the first FDN in $pollFdnSet. However, the first agent name in $pollFdnSet may not be the appropriate agent to support this particular request.

The task performed in the template's initialization is thus to check $pollFdnSet to determine if the device is configured with the agent capability appropriate for the request, and, if so, to set $pollfdn to the appropriate object from those contained in $pollFdnSet. The IsSnmpSystemUp template, for example, must be targeted at a device that supports SNMP, and $pollfdn needs to be set to point to the cmipsnmpProxyAgent object, which represents the SNMP agent system. Thus, the SetInternetSystem condition searches through the FDNs contained in the $pollFdnSet for the target device to find a match on "cmipsnmp". If no match is found, then the request knows the device is not configured appropriately for this template.

The `SetInternetSystem` condition uses an RCL WHILE loop to accomplish this. The WHILE loop needs to loop for as many times as there are objects in $pollFdnSet. The RCL numElements() function is used to discover how many objects this is:

```
$num = numElements(&$pollFdnSet);
```

If SetInternetSystem does find a match on “cmipsnmp”, it uses the RCL AppendRdn() function to set the target for the poll to the SNMP RFC 1213 internetSystem group object “contained” under the default cmipsnmpProxyAgent, which represents the agent system:

```
$num = numElements(&$pollFdnSEt);
$save_pollfdn = $pollfdn;
$res = FALSE;
$count = 1;
WHILE ( $count <= $num )
{
   $numstr = AsnToStr($count,TRUE);
   $dn = extract(&$pollFdnSet,$numstr);
   $dn1 = extract(&$dn,”distinguishedName”);
   $dnstr = AsnToStr($dn1,TRUE);
   $res = anyStr($dnstr,”cmipsnmp”);
   if ($res == TRUE)
   {
     $pollfdn = appendRdn($dn,”/internetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}”);
      $count = $num + 1;
   }
   $count = $count + 1;
}
false;
```

Note that the condition ends with the statement “false;”. This guarantees that this condition will not cause a transition to another state. This is to ensure that the next transition out of the Ground state in the template is not passed over but is evaluated. If the Boolean variable $res is false at the end of the SetInternetSystem condition, we know that the request has been targeted at a device that is not configured with a cmipsnmpProxyAgent.

You may want to warn the user when this happens. In the IsSnmpSystemUp template, the check_not_ok condition checks for $res having a value of false, and causes a transition to the Error state if this occurs. A warning alarm is posted, indicating that the target device is not configured properly for this template.

Thus, we see that there are three phases to initialization in the IsSnmpSystemUp template:

- Determining if the target device is configured with the appropriate agent support for this template

- Setting `$pollfdn` to point to the appropriate managed object
- Transitioning to an Error state (and sending an appropriate warning) if the search of `$pollFdnSet` indicates the device is not configured with the appropriate agent support

10.1.3.2 Polling for an SNMP Attribute

The `internetSystem` group contains the system description (`sysDescr`) attribute. If the Nerve Center can obtain the value of this attribute from the agent, the `IsSnmpSystemUp` request knows that the agent is running. The `IsSysDescr` condition, which checks for the value of the sysDescr attribute, is therefore used to define the transition to the Up state. This condition contains the following code:

```
defined(&sysDescr);
```

The `defined()` function can be used to poll for any attribute that has been defined in the GDMO document for the target managed object. If you try to add a condition that refers to an attribute that is not defined in a GDMO document that has been loaded into the MIS, the Request Designer displays an error message and the condition is not saved to the MIS. (You can use the SNMP Browser application to examine the attributes and groups supported by GDMO documents that the MIS knows about.)

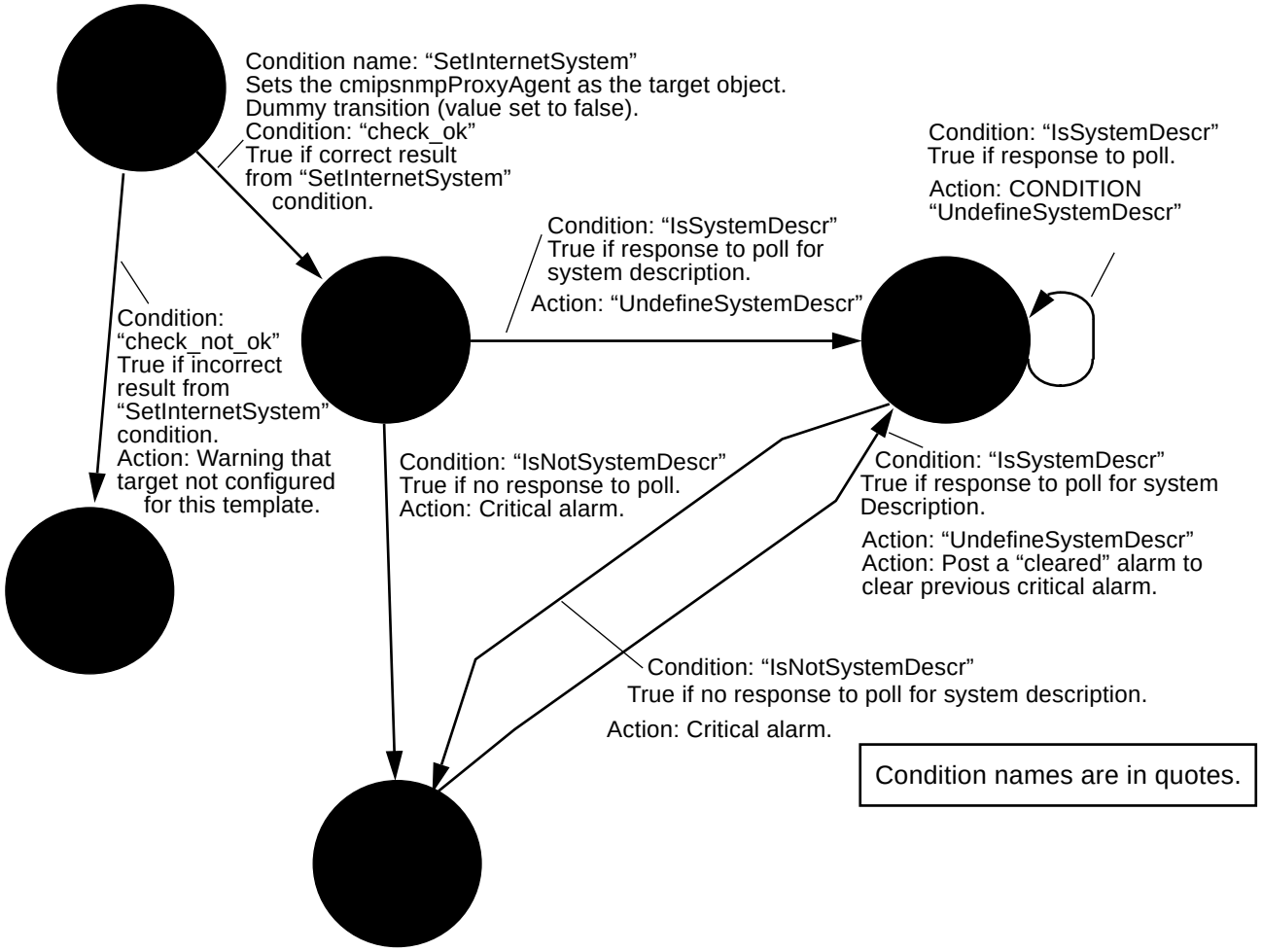


Figure 10-4 IsSnmpSystemUp Sample Request Template

If the target object’s sysDescr attribute value does not already exist in the memory space allocated for the running request, the Nerve Center retrieves the attribute from the agent. But the Nerve Center does not attempt to retrieve the

attribute if a value for sysDescr already exists. This is why the `UndefineSysDescr` condition is invoked as an action after each transition. `UndefineSysDescr` contains the following RCL statement:

```
undefine(&sysDescr);
```

This removes the attribute sysDescr from the memory allocated to the running request, forcing the Nerve Center to access the remote agent each time a new defined(&sysDescr) is called.

10.1.4 Controlling Fault Status Color in the Viewer

Your network topology is represented in the Viewer by icons once you have populated the MIS. The objects you are viewing are called *topology nodes*. Each topology node has an attribute, `topoNodeSeverity`, that represents the fault status of the device. When the value o f this attribute changes, the icon changes color to represent a change in the fault status of the device. The setting of the fault status of topology nodes is controlled by the Alarm Service — a module in the MIS that tracks incoming alarms posted to the alarm log. The Alarm Service keeps a tally of the `perceivedSeverity` values of all outstanding (uncleared) alarms logged against each topology node. The Alarm Service sets the fault status indication of a topology node to match the *highest* `perceivedSeverity` value amongst the outstanding alarms against the device.

Corresponding to each device represented in the Viewer are objects in the MIS — called *managed objects* (MOs) — that represent the agent capabilities supported by the device — for example, a cmipsnmpProxyAgent object in the MIS is used to represent a remote SNMP agent. There may be multiple managed objects that correspond to a single topology node. For example, host `bigiron` may be configured in the MIS to indicate that it can support both SNMP and RPC management. Incoming alarms are logged against the managed objects, not the topology node. However, each topology node has an attribute, `topoNodeMOSet`, which contains a set of distinguished names (FDNs) that point to the managed objects configured for that device. The Alarm Service uses this list of managed objects to match incoming alarms to devices represented in the Viewer.

When a request is launched against a selected device in the Viewer, the topology node's list of managed objects (topoNodeMOSet) is loaded into the RCL system variable $pollFdnSet. Thus, if you want to use the RCL alarm-logging functions to change the fault status color for the device in the Viewer, you need to ensure that alarms are logged against one of the objects in $pollFdnSet. When the request is initially launched, the RCL variable $pollfdn is set to point to the first object in $pollFdnSet. Therefore, the easiest way to post alarms against the target device is to log alarms against $pollfdn. The RCL `alarm()` and `alarmStr()` functions automatically post alarms against $pollfdn. However, this will work only if you do not reset `$pollfdn` in the template to point to some other object. Notice, for example, that the SetInternetSystem sample condition resets $pollfdn to point to the RFC 1213 `internetSystem` group under the SNMP agent. If `alarmStr()` were then used to post alarms, they would not affect icon color because the Alarm Service would not be able to match that object to a topology node.

However, RCL provides another alarm-logging function, `alarmOi()`, which allows you to specify the managed object the alarm is to be posted against. Also, note that the `SetInternetSystem` condition saves the original value of $pollfdn in the variable `$save_pollfdn`. Thus, an alarm can be posted against the device by passing `$save_pollfdn` to `alarmOi()`. For example, we could post a minor alarm with the following condition:

```
$info = StrToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{3,minor,\"Device is up after being
down\",3,1}");
alarmOi($save_pollfdn,$info);
```

Figure 10-5 Using AlarmOi to Log a Minor Alarm

10.1.4.1 Using `alarmOi()` *to Clear Previous Alarms*

Also, note that if a critical alarm had been logged by our request before logging a minor alarm, using the condition in Figure 10-5, this minor alarm will not cause the icon to change to cyan unless we first log a "cleared" alarm to clear the previous critical alarm. The Alarm Service always sets the fault status color to the highest severity of uncleared alarms. No matter how many minor or

warning alarms are logged, the icon remains red if there is a single uncleared critical alarm against the device. We could clear the previous critical alarm with the following condition:

```
$info = StrToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{1,5,\"Device is up\",3,1}");
alarmOi($save_pollfdn,$info);
```

Note that, in the above example, the probableCause value has been set to 1 in the cleaar alarm — the same probableCause value used in the critical alarm. For an alarm to clear a previous alarm, it is necessary that the probableCause value of the clear alarm match the probableCause value of the alarm being cleared. If the alarm() or alarmStr() functions were used to log nerveCenterAlarms, the probableCause is automatically set to a value that matches the severity. For this reason, only the `alarmOi()` function can be used to log alarms that clear previous alarms.

For templates that subscribe for incoming event notifications, alarms can be targeted to the appropriate device by using the RCL $eventOi variable. $eventOi points to the managed object that is the source of the event. Before calling the alarm() or alarmStr() functions, you could set $pollfdn to point to the managed object that is the source of the event:

```
$pollfdn = $eventOi;
```

10.1.4.2 *Alarm-logging Tips*

We can summarize the alarm-logging considerations discussed here as follows:

- Event notifications must have a `perceivedSeverity` value to affect fault status color. Alarm Service uses the highest outstanding severity to determine fault status. If you want an icon to change to indicate a lower severity after your request has posted a higher severity alarm, your request should first clear the higher severity alarm.
- To affect fault status color, alarms must be logged to the alarm log monitored by the Alarm Service (by default, this is the log called `AlarmLog`). The RCL alarm-logging functions (alarm(), alarmStr(), and alarmOi()) can be used to log nerveCenterAlarms to the alarm log.

- If your request template resets $pollfdn to point to a different object, you may want to save the initial value of $pollfdn in another variable for use in alarm-logging. The functions `alarm()` and `alarmStr()` automatically post alarms against $pollfdn. If you have reset the value of $pollfdn, you can use `alarmOi()` to post alarms against a particular managed object.
- If you want to log an alarm to clear a previous alarm, you must use alarmOi() and set the `probableCause` value of the clear alarm to match the probableCause of the alarm it is clearing. If the previous alarm was logged using alarm() or alarmStr(), its probableCause value is the same as its severity.
- The managed object that is the source of an incoming event can be obtained using the RCL `$eventOi` variable. This can then be passed to `alarmOi()` to log a `nerveCenterAlarm` against that object.

For more information...

You may want to consult the "Alarm Service" and "SNMP Browser chapters in the *Solstice Enterprise Manager Reference Manual*, as well as the following chapters in this guide:

- Chapter 9, "Nerve Center Overview."
- Chapter 12, "Request Designer."
- Chapter 13, "Request Condition Language."
- Chapter 14, "RCL Functions."

10.2 Designing Request Templates

Before embarking on building a request template from scratch, use the Request Designer to examine the sample request templates supplied with the product. You may be able to use a template as is, modify a template, or, short of these labor-savers, use one or more of the conditions that are used in the sample templates. See the following subsection for a procedure for creating a template from an existing template.

If you find you need to create a request template, the essential steps are as follows:

1. **Design a state machine: draw a picture for yourself showing the states you want to monitor and the paths between those states. Make note of conditions that would cause movement from one state to another.**

2. **Invoke the Request Designer, as described below.**

The following steps all involve the use of the Request Designer.

3. **Create the states you need.**
   States are specific to each template, that is, you have to create new states for each template.

4. **Create the conditions you need.**
   You are supplied with a number of conditions. Conditions are reusable across all request templates. You may wish to develop a library of conditions that can be used in multiple templates.

5. **Create the transitions from one state to another.**
   Transitions are specific to each template and are executed in the order in which they appear in the template.

6. **Name the request template and enter a brief description of it.**

7. **Save the template.**

Note – You can save an incomplete template, to continue work on it at a later date, through the Request Designer's Template➤Export Current option. Use the Template➤Import option when you want to reload that template into the Request Designer.

With a template created, you can invoke the Requests window from the Viewer's Tools menu and start requests using that template against target managed objects.

The bulk of the work in building a new request template is in the design of the template and in the coding of the conditions for the template. You should design your template before you invoke the Request Designer, although you can use the application's graphical display as your drawing board. We recommend composing your conditions in the Request Designer, because the application tests the syntax of your condition code when you attempt to save it.

With your design and a set of conditions in place, the putting together of a request template is a simple matter of moving through menu selections. The Request Designer gives you both a text-based and a graphical method of creating templates. The choice of one or the other is a matter of personal preference.

10.3 Requests Based on Polling

The `IsSnmpSystemUp` template, discussed earlier, is an example of a request based on polling — periodically checking a managed resource for the current value of one of its attributes. A limitation of the `IsSnmpSystemUp` template is that it can only determine if an SNMP agent system is available on the occasions that it polls the system. But you may want to be notified if an SNMP device has been down momentarily and then become available. Another attribute in the SNMP RFC 1213 `internetSystem` group — `sysUpTime` — can be used to design a request template that does this. Building such a template will illustrate the process of creating templates based on polling.

`sysUpTime` measures the time, in hundredths of a second, since the last system restart. If this value has decreased since the last poll, we know that the system went down between polls. We can create a state in the template called "EverDown" to represent the situation where the device is currently up but was previously down. To make this visible in the Viewer, we can log an alarm with a `perceivedSeverity` of minor when a transition to the EverDown state occurs. However, this will only cause the Viewer icon color to "decay to cyan" if we first log a "cleared" alarm to clear the previous critical alarm. Fault status color is determined by the highest severity of uncleared alarms against a device.

The states for our template, and the corresponding Viewer color, might be the following:

- Ground — no color
- Poll — no color
- Up — no color
- Down — Red (critical alarm)
- EverDown — Cyan (minor alarm)

To ensure that the Viewer icon for the target object displays an appropriate color in response to a change in request state, we can add, as an action at each transition, a condition that calls an RCL alarm-logging function. For example,

to cause icons to turn cyan when the request enters the EverDown state, we can add a condition, `AlarmMinorOiEverDown`, consisting of the following statement:

```
IF ($everdown_alarm == FALSE)
{
$info = StrToAsn(“EM-NC-ASN1.NerveCenterAlarmInfo”,”{3,minor,\”Device is up after being
down\”,3,1}”);
alarmOi($save_pollfdn,$info);
}
$everdown_alarm = TRUE;
```

Because the SetInternetSystem condition resets $pollfdn to point to the internetSystem group, we need to use `$save_pollfdn` to log the alarm against the target device. The Boolean variable $everdown_alarm is used a latch, to ensure that the request does not flood the alarm log with duplicate alarms.

As in the IsSnmpSystemUp template, we can set the target of the poll to the SNMP RFC 1213 `internetSystem` group by using the SetInternetSystem condition to define a transition from Ground to Poll.

We want the request to transition from Poll or Up to EverDown if the value of `sysUpTime` decreases. This means we will need a variable to store the previous value of `sysUpTime`, to compare with the results of the next poll. This variable should be initialized to zero in the transition from the Ground to Poll state. For example, we might add a condition, `InitLastSysUpTime`, as an action at the transition from Ground to Poll; this condition might consist of this statements:

```
$last_sys_up_time = 0;
$everdown_alarm = FALSE;
$down_alarm = FALSE;
```

To define the transition from Up (or Poll) to EverDown, we might compose a condition, `sysUpTimeDecrease`, as follows:

```
$last_sys_up_time>sysUpTime;
```

We will also need to re-initialize the variable `$last_sys_up_time` with the current value of sysUpTime after each successful poll. For example, we might compose a condition, `getSysUpTime`, to do this:

```
$last_sys_up_time = sysUpTime;
```

We might call our template “IsSnmpSystemEverDown”; a possible state diagram for our template is pictured in Figure 10-6.

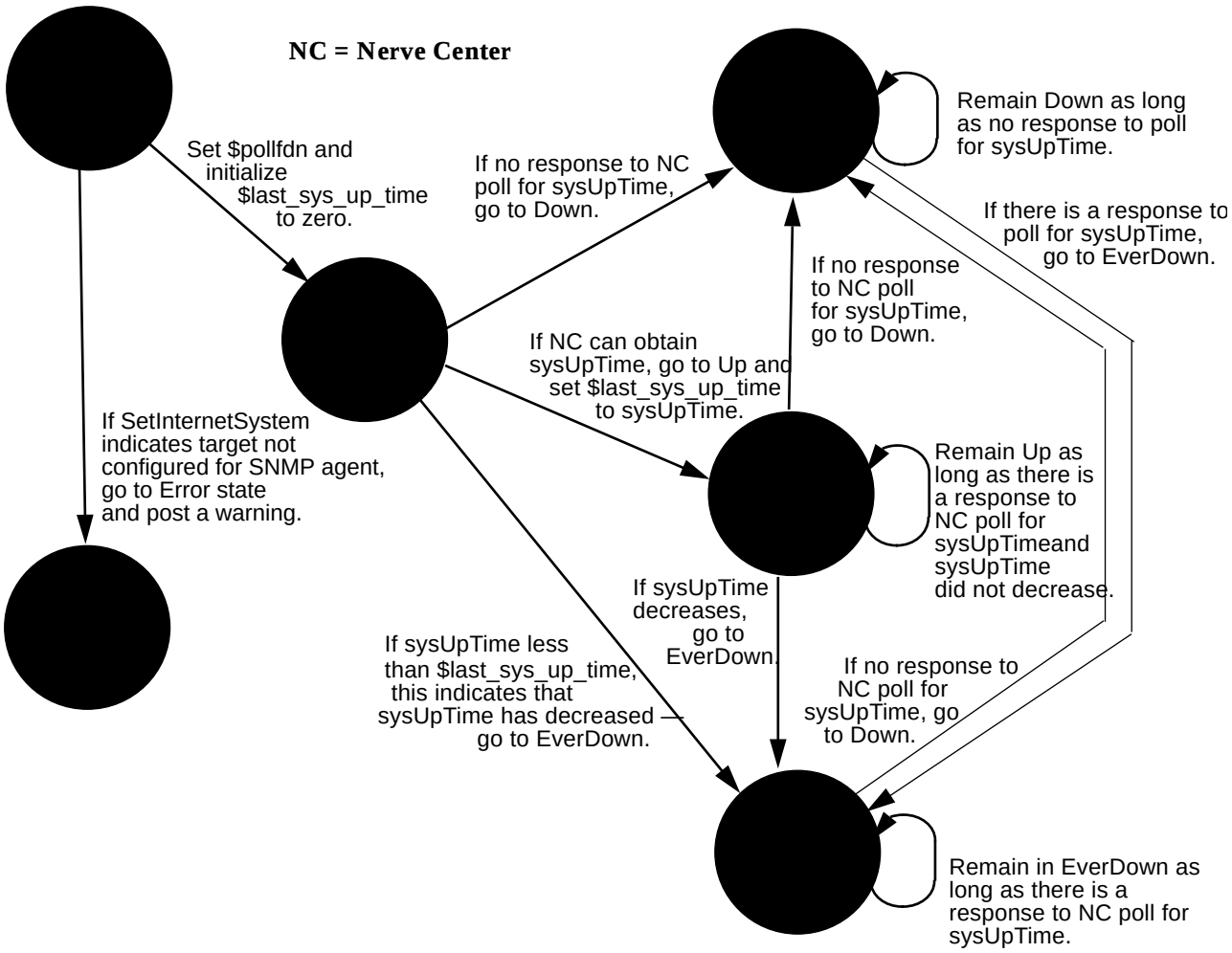


Figure 10-6 State Diagram of IsSnmpSystemEverDown Template

10.3.1 Adding States

To add a state, such as EverDown, to our template, invoke the Edit States window (by clicking on the main window States button) and do the following:

1. **Type in the name (“EverDown”) in the State field.**
2. **Describe what the state represents in the Description field** — for example, “Agent system is up but has been down.”
3. **Select a poll rate from the Poll Rate menu.**
   Note that the polling interval for poll rates (in seconds) can be changed, or new poll rates added, in the Poll Rates window (invoked by selecting the Edit➤Poll Rates menu option).
4. **Select a severity from the Severity menu.**
   Note that severity here is a value that is internal to the request. This does not cause an alarm to be logged, nor does it automatically cause a change in icon color in the Viewer when the request enters the state. Icon color is determined by alarms that are logged against a managed object. Control of icon color is discussed above in Section 10.1.4, “Controlling Fault Status Color in the Viewer.”
5. **Click on Add to add the new state to the template.**

Repeat this procedure for the other states to be added.

10.3.2 Adding Conditions

To add a condition, invoke the Conditions window by clicking on the Conditions button in the Request Designer main window. You create the condition code by simply typing in the text window. For example, a condition we will want to use in the IsSnmpSystemEverDown template tests whether sysUpTime has not decreased; this could be entered as follows:

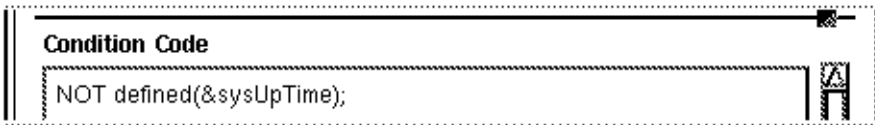


Figure 10-7 Entering Condition Code in the Request Designer

This RCL statement returns a value of true when (and only when) the expression “$last_sys_up_time > sysUpTime” is false.

To add the new condition to the MIS, do the following:

- Enter a name (“sysUpTimeNotDecrease”) for the condition in the Name field
- Describe what the condition does in the Description field
- Click on Add to store the new condition in the MIS.

Repeat this procedure for the other conditions to be added.

10.3.3 Adding Transitions

After you have created the states and conditions that you need for the template, you can begin to create transitions.

After you have created the Poll state, for example, do the following to create the first transition from the Ground to Poll state:

1. **Invoke the Transitions window in the Request Designer main window.**
2. **Select the** `Ground` **state as the From state.**
3. **Select the** `Poll` **state as the To state.**
4. **Select** `SetInternetSystem` **as the Condition to define this transition.**
5. **To select an Action to be taken at the transition, select either CONDITION, MAIL, or UNIXCMD.**
   Because `SetInternetSystem` is a “dummy” transition that never occurs, we do not select an action for this transition. Select <none>.
6. **Click on Add to add this transition.**

To add additional actions at a transition, do the following:

1. **Set the From, To, and Condition field settings so that they match the transition to which you wish to add an additional action.**
   If you are adding an action after just creating the transition, the settings will already be set properly for adding an additional action.
2. **Select the type of action from the Action menu**
   a. **If you selected MAIl as the type of action, fill in the Message and Address fields**
   b. **If you selected UNIXCMD as the type of action, fill in the Command and Arguments fields as required by the UNIX command.**

c. **If you selected CONDITION as the type of action, select the condition from the condition Name menu.**

3. **Click on Add to add the new action to the list of actions that will be performed at that transition.**

Note that actions at a transition are executed in the order they appear in the template textual display (which is the order you entered them in the template). If you want to delete one of the actions at a transition, you can only delete the last action listed in the that transition. By deleting the actions from the bottom up, as needed, you can re-enter them to arrange them in the desired order.

Repeat this procedure for each of the transitions in the template. For example, we need to do error-checking to determine whether SetInternetSystem indicates a configuration error. If an error occurs, we transition to the Error state; if no error occurs, we transition to the Poll state. The `InitLastSysUpTime` condition can be called as an action in the transition from Ground to Poll, thus completing the request's initialization.

After we have completed the initialization phase, we might create the three transitions out of the Poll state shown below for our `IsSnmpSystemEverDown` template:

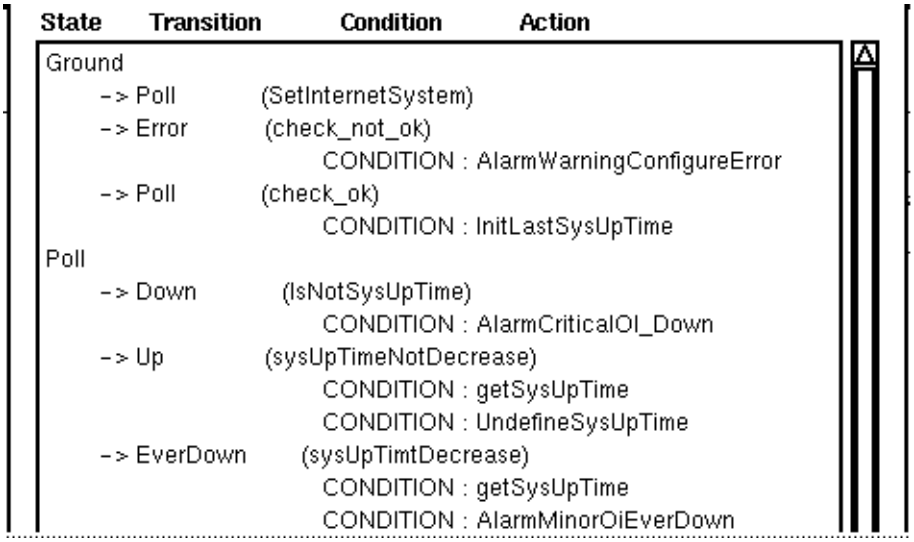


Figure 10-8 Order of Transitions in IsSnmpSystemEverDown

The transition to the Down state is the first transition out of Poll because we want IsNotSysUpTime to be checked first, to determine if the request will be able to retrieve the sysUpTime value. If this condition evaluates to false, the request knows that sysUpTime has been retrieved, and it can then determine if it has decreased. If it has not decreased, the request transitions to the Up state. If it has decreased, the request transitions to the EverDown state.

If you enter the transitions out of a certain state in the wrong order in a template, you can change the order by invoking the Order Transitions window. The Order Transitions window is accessed by clicking on the Order Transitions... button in the Transitions window.

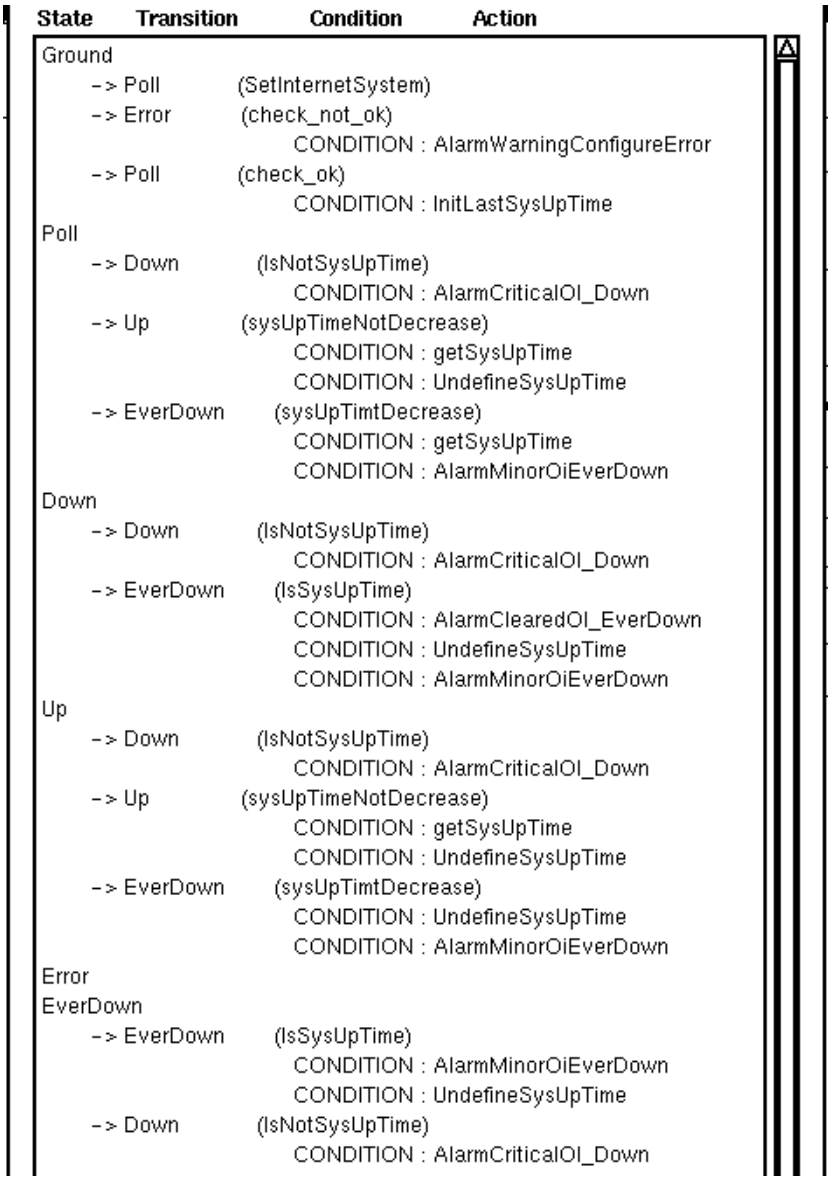


Figure 10-9 isSnmpSystemEverDown Template

The condition UndefineSysUpTime contains the following RCL statement:

```
undefine(&sysUpTime);
```

This condition needs to be called after each successful poll to remove the last sysUpTime value from the memory space where the request is running. This forces the Nerve Center to retrieve the current `sysUpTime` value from the agent system at the *next* poll. But before removing sysUpTime from memory, we call the `getSysUpTime` condition, which stores the retrieved `sysUpTime` value in the variable `$last_sys_up_time` for comparison with the value of sysUpTime at the next poll.

Figure 10-9 shows the completed IsSnmpSystemEverDown template as it appears in the Request Designer textual display: Select the Template➤Save As option to save the completed template.

10.4 Polling RPC Agents

A limitation of our IsSnmpSystemEverDown template is that it cannot distinguish between a situation where the lack of response to a poll for `sysUpTime` is due to the SNMP daemon being down and a situation where the lack of response is due to the unavailability of the machine on which the daemon is installed. The SnmpPingBackoffReachable sample template overcomes this limitation by using the `ping` RPC proxy agent to poll for reachability. Discussing this template will illustrate the design of request templates that do direct polling of RPC agents.

The SnmpPingBackoffReachable request begins by polling every 30 seconds for the SNMP system for the sysUpTime attribute. If there is no response to the poll, the request backs off the poll rate to 60 seconds. If there is still no response to the poll, the request attempts to poll the device for reachability using the RPC ping proxy agent. If there is a response to the ping, the request knows that the machine is up but the SNMP daemon is down, and a major alarm indicating this is logged.

However, if there is no response to the poll for reachability, the request knows that the machine is unavailable and logs a major alarm indicating this. The request continues to poll for reachability. When a response is received, a

warning alarm is logged, indicating that the device is up after having been down. The request then transitions back to polling for the SNMP sysUpTime attribute. A state machine diagram for this request is shown in Figure 10-10.

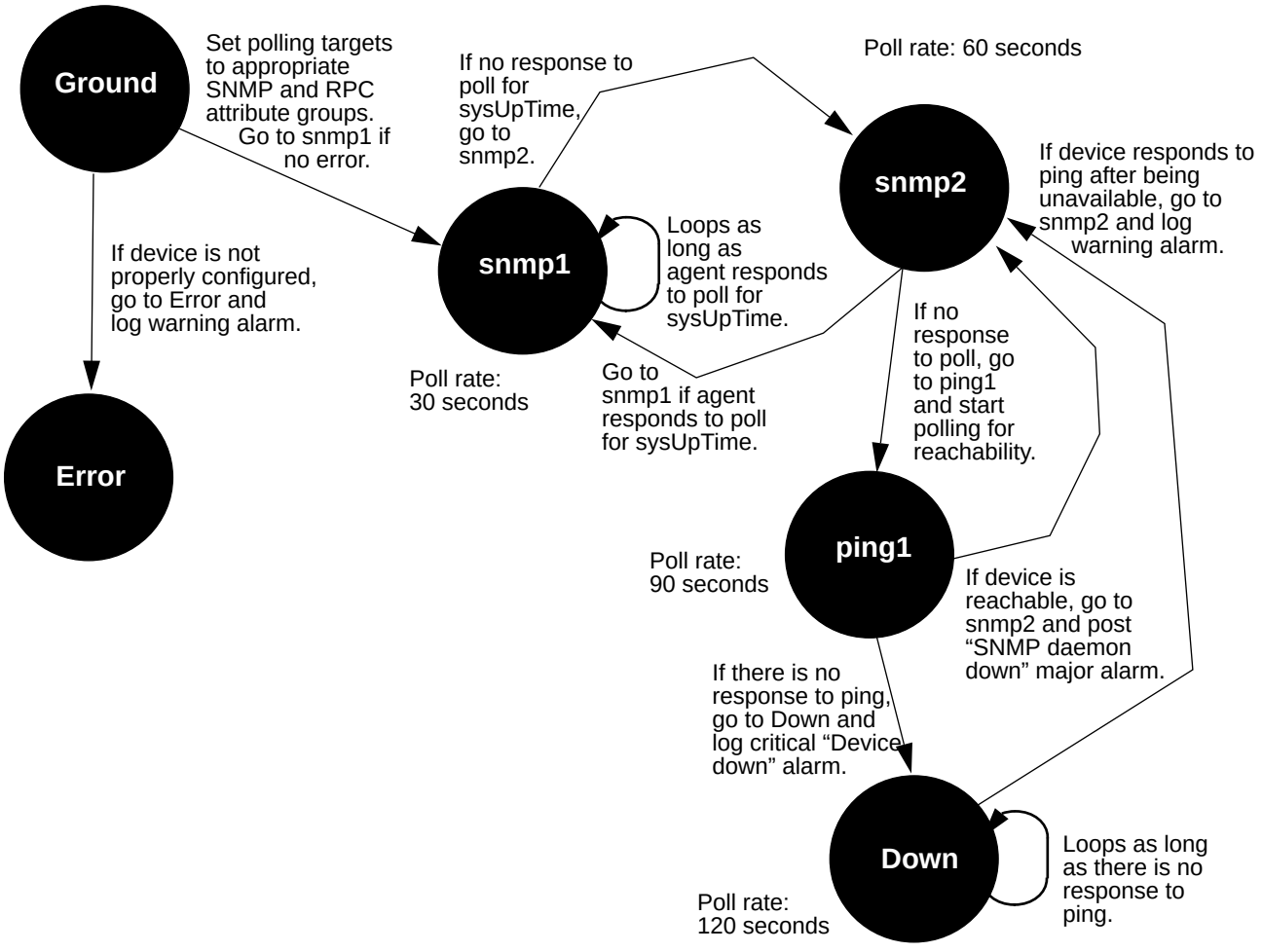


Figure 10-10 State Diagram of SnmpPingBackoffReachable Request

10.4.1 *Targeting the RPC* `ping-reach` *Group*

The SnmpPingBackoffReachable template polls for both the SNMP `sysUpTime` attribute and the `reachable` attribute supported by the RPC ping proxy agent. To do this, the request must extract both the cmipsnmpProxyAgent object and the RPC proxy table object from $pollFdnSet for the device against which the user has launched the request. Just as SNMP attribute groups are "contained" under the cmipsnmpProxyAgent object, all RPC agent attribute groups configured for a device are "contained" under the RPC proxy table object for that device. The request should check to ensure that the target device is configured to support both SNMP and RPC. In this case, the SnmpPingBackoffReachable request transitions to an Error state if it determines that the device is not configured to support the request.

The first transition in the template uses the following condition to obtain the RPC proxy table object from $pollFdnSet:

```
$save_pollfdn = $pollfdn;
$num = numElements(&$pollFdnSet);
$count = 1;
while( $count <= $num )
{
   $numstr = AsnToStr($count,TRUE);
   $dn = Extract(&$pollFdnSet,$numstr);
   $dn1 = Extract(&$dn,"distinguisedName");
   $dnstr = AsnToStr($dn1,TRUE);
   $res = AnyStr($dnstr,"RPC");
   if ($res == TRUE);
   {
      $rpc_mo = appendRdn($dn,"/agentId=\"ping-reach\"");
      $count = $num + 1;
      print($rpc_mo);
   }
   $count = $count+1;
}
false;
```

Figure 10-11 `get_rpc_mo` Condition

The initial setting of $pollfdn is saved in $save_pollfdn. $save_pollfdn can then be used to target alarms against this device. Alarms must be targeted at one of the objects in $pollFdnSet in order for the Alarm Service to match it to a device icon in the Viewer.

A WHILE loop is used to locate the RPC proxy table object by finding a match for "RPC" on one of the FDNs in $pollFdnSet. An `appendRdn()` operation is used to set `$rpc_mo` to point to the `ping-reach` attribute group "contained" under the RPC proxy table. This is accomplished by appending `/agentId="ping-reach"` to the RPC proxy table FDN.

The condition ends with "false;" to ensure that the transition defined by this condition is not actually executed, and the request then proceeds to evaluate the next condition, `get_snmp_mo`, which is similar to the SetInternetSystem condition discussed in Section 10.1.3.1, "Setting the Target Managed Object." This condition sets $snmp_mo to point to the `internetSystem` group object under the cmipsnmpProxyAgent object (which represents the SNMP agent).

The third transition out of the Ground state checks for errors:

```
$num < 2 OR NOT defined(&$snmp_mo) OR NOT defined(&$rpc_mo);
```

This condition will evaluate to true if $pollFdnSet had one or fewer objects (i.e., not both SNMP and RPC) or the internetSystem group or RPC ping-reach group are not defined for the target device. If so, the request transitions to the Error state and an appropriate warning alarm is posted. If there is no error, the request transitions to the snmp1 state in the transition defined by the initialize_variables condition:

```
$ping_alarm = FALSE;
$snmp_alarm = FALSE;
$pollfdn = $snmp_mo;
true;
```

The $pollfdn is set to the SNMP internetSystem group to poll for the sysUpTime attribute. "true;" ensures that this transition will occur when it is evaluated.

10.4.2 Correlating Information from Multiple Polls

The SnmpPingBackoffReachable uses polls for the sysUpTime attribute in the snmp_up condition:

```
defined(&sysUpTime);
```

As long as this condition evaluates to true, the request loops in the snmp1 state. After each poll, the `UndefineSysUpTime` condition is called to force the next `defined()` call to access the remote agent. If there is no response to the 30 second poll for sysUpTime, the request transitions to the snmp2 state which backs off the poll rate to 60 seconds, in case a longer timeout is needed in waiting for a response from the agent. If there is now a response from the agent, the request transitions back to snmp1 and in the transition executes the is_snmp_backup condition:

```
if ($ping_alarm == TRUE)
{
$info = StrToAsn("EM-NC-
ASN1.NerveCenterAlarmInfo","{4,warning,\"SNMP Daemon is now
responding\",3,1}");
alarmOi($save_pollfdn,$info);
}
$ping_alarm = FALSE;
$snmp_alarm = FALSE;
```

The IF construct here is used to distinguish between the situation where the machine has previously failed to respond to a ping and the situation where it has not failed a poll for reachability. This warning alarm will be logged only in the situation where it has not previously failed a response to a poll for reachability; that is, the request had previously determined that failure of response to a sysUpTime poll was due to a failure of the SNMP daemon and not due to the unreachability of the machine.

If there is still no response to the poll for sysUpTime, the request transitions to the ping1 state to begin polling for reachability, using the RPC ping proxy agent. In this transition the $pollfdn is set to point to the ping-reach attribute group with the `set_ping_pollfdn` condition:

```
$pollfdn = $rpc_mo;
```

The request tests for reachability with the `ping_up` condition:

```
reachable == 1;
```

If this condition evaluates to true, the request knows that the previous absence of a response to the previous poll for sysUpTime was not due to the unavailability of the machine, but indicates a failure of the SNMP daemon. The request thus transitions back to snmp1 and executes the is_ping_alarm condition as an action in the transition:

```
if ($ping_alarm != TRUE)
{
$info = StrToAsn("EM-NC-
ASN1.NerveCenterAlarmInfo","{2,major,\"SNMP daemon is not
responding\",3,1}");
alarmOi($save_pollfdn,$info);
}
$ping_alarm = TRUE;
```

This condition logs a nerveCenterAlarm with a severity of major against the device the request was launched against (indicated by $save_pollfdn), but only if an "SNMP daemon is not responding" alarm has not already been logged. Setting the $ping_alarm variable to true ensures that if the SNMP device does start responding to a sysUpTime poll, an alarm will be logged indicating that the SNMP daemon is up after having been down.

If the device does not respond to the ping, the `ping_down` condition will evaluate to true:

```
reachable == 0;
```

If so, the request knows that the absence of a response to the original sysUpTime poll was due to the unavailability of the machine, and thus the request transitions to the Down state. The request executes the `deviceDownCriticalAlarm` condition as an action at this transition:

```
alarmStr(1,"Device Not Responding to Ping");
```

The request loops in the Down state as long as the device does not respond to polls, which are generated at two-minute intervals. If the device responds to a ping, the request transitions back to the snmp2 state and issues a warning alarm in the ping_back_up condition.

10.5 Requests Based on Event Subscription

In addition to responding to requests from managers, agents typically have the ability to detect conditions and generate messages — called *event notifications* — on their own initiative. Event *subscription* is a facility that forwards specified event notifications to a request as soon as they arrive at the MIS. Rather than actively polling devices in the network, the request waits for the arrival of specified event notifications, and then takes appropriate action when this happens.

A request can "subscribe" to receive any type of event notification that is known to the MIS. Subscriptions can request all events of a certain type, events generated by a specified object, or all events that satisfy a CMIS filter. (For information on CMIS filters and the types of event notifications that are defined in Solstice EM by default, refer to the *Solstice Enterprise Manager Reference Manual* appendices.)

When designing requests based on event subscription, you will want to correlate this with your use of other Solstice EM event-handling features:

- Event logging and Alarm Service monitoring of alarm logs
  Most types of incoming event notification are, by default, logged to the `AlarmLog`; this log is, by default, used by the Alarm Service to determine the fault status of devices. The fault status of the device — indicated by icon color in the Viewer — is set to the highest severity of outstanding (uncleared) alarms against that device. However, if you are, for example, using requests to determine when alarms are posted in response to the arrival of internetAlarms, you might not want internetAlarms to be

automatically logged to the `AlarmLog`. The Log Manager is used to define which events are logged to specified logs. The Log Manager and Alarm Service are discussed in the *Solstice Enterprise Manager Reference Manual*.

- Mapping of SNMP traps to CMIP event notifications
  The Solstice EM SNMP trap daemon (`em_snmp-trap`) converts incoming SNMP traps into CMIP event notifications for forwarding to the MIS. The type of event the trap is mapped to depends upon the way you have configured the trap daemon's mapping capability. (SNMP trap-mapping is described in Chapter 8, "Mapping SNMP Traps to CMIP Event Notifications.") If you are designing a subscription template to listen for SNMP traps, the event types your request should subscribe for will depend upon the trap-to-event mapping implemented by the trap daemon.

10.5.1 *Example: Subscribing for Enterprise-Specific SNMP Traps*

An example of an event subscription request would be a request that subscribes to receive all enterprise-specific SNMP traps.

You could design an event-subscription request that uses the Nerve Center alarm-logging function to generate more meaningful alarms. For example, suppose that the enterprise-specific traps generated by certain devices on your network have specific trap values that are to be interpreted as indicated in Table 10-1.

Table 10-1 Enterprise Specific Traps Example

Specific Trap Number	Description	Desired action
1	CPU Failure	Critical alarm
2	Power Supply Failure	Critical alarm
3	Fan Failure	Critical alarm
4	Overheating	Warning alarm
5	Realtime Clock Failure	Ignore
6	Network Connection Failure	Minor alarm

Your request could subscribe for enterprise-specific traps and then use the RCL alarm-logging functions to log nerveCenterAlarms with the appropriate severities in response to incoming traps. We would thus be using nerveCenterAlarms to drive fault status indication in the Viewer for incoming enterprise-specific alarms. This means that we want to route incoming enterprise-specific traps to our subscription request before they become alarms that affect Viewer icon color.

For purposes of this example, let us suppose that you have configured the SNMP trap daemon to convert enterprise-specific traps to internetAlarms in accordance with the ISO-Internet Management Coexistence (IIMC) standard. (The IIMC standard defines the use of the CMIP protocol for integrated management of TCP/IP networks that are managed using SNMP. The trap daemon maps enterprise-specific traps to IIMC internetAlarms by default if no mapping is provided for enterprise-specific traps in its trap-mapping file.) Thus, you might have configured the trap daemon to map SNMP generic trap types to event notifications as follows:

- coldStart traps — coldStartTrap notifications
- warmStart traps — warmStartTrap notifications
- linkDown traps — linkDownTrap notifications
- linkUp traps — linkUpTrap notifications
- authenticationFailure traps — authenticationFailureTrap notifications
- egpNeighborLoss traps — egpNeighborLossTrap notifications
- enterpriseSpecific traps — internetAlarms

All of these alarm types are logged to the AlarmLog by default, and thus affect icon status color in the Viewer. However, in this example we are assuming that you want to control fault status indication for enterprise-specific traps with a request template rather than relying on automatic tracking of internetAlarms by the Alarm Service. Thus, we want to remove internetAlarms from the AlarmLog to eliminate duplication of alarms. To do this, you could use the Log Manager to alter the log discriminator for the AlarmLog to filter out internetAlarms. (An example is described in Chapter 3, “Managing Devices.” Also, you may want to refer to the “Log Manager” chapter in the *Solstice Enterprise Manager Reference Manual*.)

10.5.1.1 Initiating the Event Subscription

Because, in this example, the trap daemon is mapping incoming enterprise-specific traps to internetAlarms, we will want our request template to listen for incoming internetAlarms.

A subscription request typically initiates the subscription in the transition out of the Ground state. For example, we might define a condition, called "SnmpTrapSubscription" that has the following condition code:

```
$itindx=Subscribe("internetAlarm");
$itType = NameToOid("internetAlarm");
false;
```

Figure 10-12 SnmpTrapSubscription Condition

The variable `$itindx` receives a value of `-1` if an error occurred which prevented the subscription from being implemented. Accordingly, you could define a condition to check if such an error occurs, and then transition to a "Dead" state if it does. The following is an example:

```
$itindx < 0;
```

Figure 10-13 IsSubscriptionError Condition

The `IsSubscriptionError` condition is used to define the transition from Ground to Dead. The use of "false" in the last line of SnmpTrapSubscription conditions ensures that this error-checking transition will be evaluated. A third transition out of the Ground state is defined by the `IsNotSubscriptionError` condition:

```
NOT ($itindx < 0);
```

After subscribing for internetAlarms and checking for subscription error, the request transitions from Ground to a state where it listens for incoming internetAlarms, if no error has occurred. Accordingly, you might want to create a state in the template called "Waiting," to represent this situation.

10.5.1.2 Listening for Incoming Events

In the Waiting state the request checks for the arrival of incoming internetAlarms by testing the following condition:

```
$eventType == $itType;
```

Figure 10-14 receivedTrap Condition

$eventType is a system variable that contains the type of the current event notification. If this condition evaluates to true, an internetAlarm has arrived and we will want this to cause a transition to another state — which we might call the “Problem” state — where the request examines the internetAlarm in detail and takes appropriate action, depending on the nature of the trap. A possible state diagram for our SNMP trap subscription template is pictured in Figure 10-15.

The `examineTrap` condition illustrated in Figure 10-16 is an example of how the request could interpret enterprise-specific traps in the way suggested by the example in Table 10-1. Alarms are posted with the appropriate severity and the cause of the trap is passed in the `additionalText` field of the `nerveCenterAlarm`.

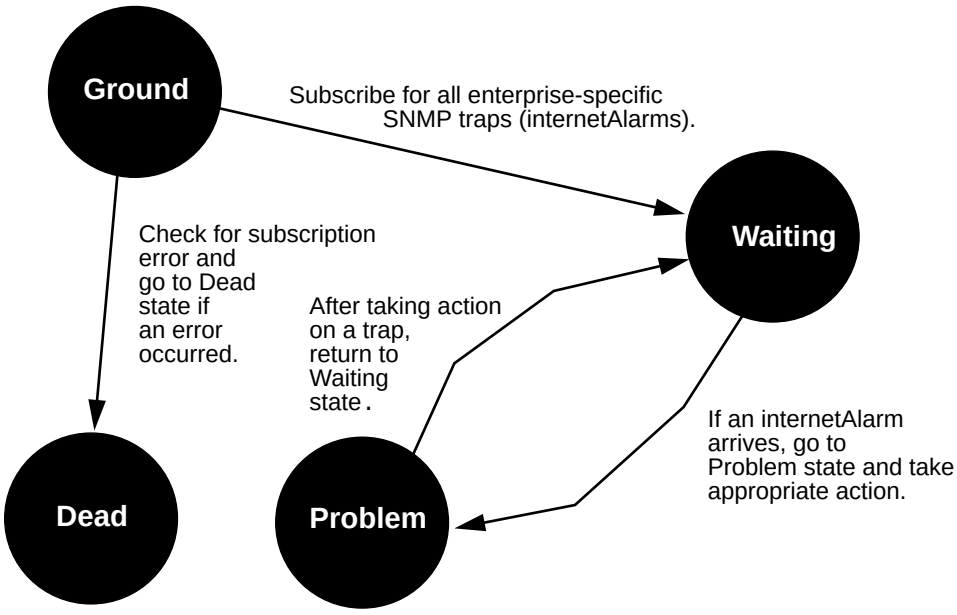


Figure 10-15 State Diagram for `IsEnterpriseSpecificTrap` Template

```
$gnum = TrapGenericType($eventInfo);
$snum = TrapSpecificType($eventInfo);
$pollfdn = $eventOi;
IF ($gnum == 6)
{
   IF ($snum == 1)
   {
      alarmStr(1,”CPU Failure”);
   }
   ELSE
   {
       IF ($snum == 2)
       {
          alarmStr(1,”Power Supply Failure”);
       }
       ELSE
       {
          IF ($snum == 3)
          {
             alarmStr(1,”Fan Failure”);
          }
          ELSE
          {
             IF ($snum == 4)
             {
               alarmStr(3,”Overheating”);
             }
             ELSE
             {
                IF ($snum == 6)
                {
                   alarmStr(4,”Network Connection Failure”);
                }
             }
          }
       }
   }
}
```

Figure 10-16 examineTrap Condition

Note that $pollfdn is set to $eventOi. $eventOi is a system variable that indicates the managed object that is the source of the event. We set $pollfdn to point to this object because $pollfdn determines the managed object that will have a `nerveCenterAlarm` posted against it by the alarmStr() function. The text string passed to the `alarmStr()` function is passed in the `additionalText` field of the alarm, which can be viewed in the Alarm Manager.

Our completed `IsEnterpriseSpecificTrap` template is shown in Figure 10-17.

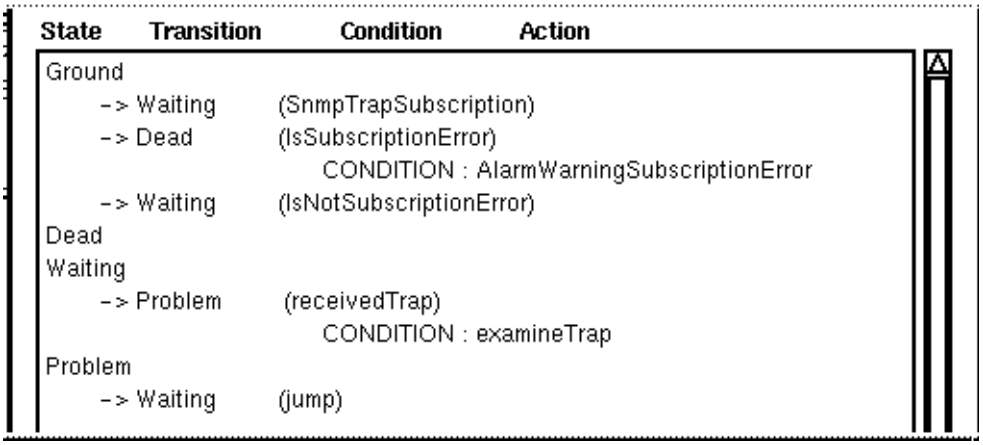


Figure 10-17 SNMP Trap Subscription Template

10.6 Debugging Your Templates

There are several faciliities available to you in debugging templates:

- The Request Designer does RCL syntax checking when you attempt to save a newly created, or modified, condition. However, the Request Dpesigner will not catch possible runtime errors.
- The Request Monitor tool (accessible from the Viewer's Tools menu) lists the requests currently executing in the Nerve Center. If you select a request and click on the Examine button, the Request Examine window is invoked. This window displays the values of variables in the request as it is executing. (Refer to the "Viewer" chapter in the *Solstice Enterprise Manager Reference Manual* for information on Request Monitor.)

The Solstice EM `em_debug` utility also provides facilities that are useful in debugging templates.

10.6.1 Nerve Center Debugging Agents

As a part of `em_debug`, there are a number of Nerve Center debugging "agents" which track aspects of Nerve Center operation and display messages in the shell where they are invoked.

- `nc_state` — Traces transitions from state to state, indicating the current state and the condition used to transition out of a state.
- `nc_poll` — Traces the enabling and disabling of polling in states.
- `nc_event` — Provides information on all event notifications that have been received.
- `nce_error` — Provides information on Nerve Center runtime errors.

For example, you can activate the `nc_state` agent by entering the following command:

```
hostname% em_debug -c "on nc_state"
```

In reporting on state transitions in a running request, nc_state refers to states by number. States are numbered by order of appearance in the left-most column in the Request Designer textual display.

You should invoke the debugging agents before launching the request in the Viewer.

The Nerve Center debugging agents report on the activities of any request running in the MIS. If you have multiple requests running, it may be difficult to isolate which request is the cause of a message that is displayed. For this reason, it is recommended that you only have the request running which you are trying to debug when using the NC debugging agents.

10.6.1.1 Activating RCL Print Statements

Request Condition Language provides a `print()` function which you can use in conditions to help you in debugging templates — by printing current values of variables, for example. You can use the following `em_debug` command to activate RCL `print()` statements:

```
hostname% em_debug -c “on misc_stdout”
```

The RCL `print()` statements will be displayed in the shell where this command was invoked.

As with the NC debugging agents, the misc_stdout agent turns on print statements for all requests running in the MIS that contain the `print()` function. If you follow a practice of removing print() statements from templates after new conditions have been debugged, you can use `print()` statements to debug new templates even while other requests are running in the MIS. Only messages from the request being debugged are then displayed.

Keep in mind that the RCL `print()` function always returns a value of true. If a `print()` statement is the last statement in a condition that defines a transition, that transition will always occur. Accordingly, when debugging a template, you may want to avoid `print()` statements in conditions that define transitions, and restrict them to conditions that are used as actions after a transition.

Note – An RCL `print()` statement will only be executed if it is in a condition that is evaluated. If a state is never “woken up” by either a poll for an attribute value or the arrival of an incoming event, the conditions defining the transitions out of that state will never be evaluated. Also, the conditions that define transitions out of a state are evaluated in the order they appear in the template. If a prior condition has evaluted to true, and the request transitions out of the state, the subsequent transitions in that state are not evaluated.

10.6.2 Turning Off Debug Agents

You can turn off a particular debugging agent by entering the following commend:

```
% em_debug -c “off <agent-name>”
```

If you want to turn off all debugging, you can use the `em_debug` wildcard feature, as follows:

```
% em_debug -c “off *”
```

10

Building Templates for SunNet Manager Event Requests

11

Overview	*page 11-1*
Nerve Center's SNM Event Request Capability	*page 11-4*
SNM Alarms	*page 11-6*
Building SNM Event Request Templates	*page 11-7*

11.1 Overview

Solstice Enterprise Manager is shipped with a suite of agents developed for the Site/SunNet/Domain Manager (SNM) network management system. These agents communicate with a network manager, such as Solstice EM, using Remote Procedure Call (RPC) protocol within an Internet (TCP/IP) network environment. These RPC agents have the ability to poll managed resources to check for predefined thresholds and send an event notification — called an *SNM event* — to a specified management station. This polling activity is initiated by a one-shot message from a management station — called an *SNM event request*. The SNM event request defines the threshold and polling interval for the agent's polling activity. The agent thus acts as a *proxy* for the manager. Polling activity is offloaded from the management station to the RPC proxy agents, which may be distributed to various sites around your network. For example, a certain machine (either a PC running Solaris for x86 or a SPARC workstation running SunOs 4.x or Solaris 2.x) — called a *proxy host* — may contain the proxy agents for polling of resources in a particular subnet.

The Solstice EM Nerve Center has the ability to initiate SNM event requests. This enables EM to offload the polling of the managed resource from the MIS. If the threshold defined in the event request obtains on the managed resource, the RPC agent sends an SNM event to the SNM Event Dispatcher (`na.event`) (by default, this is sent to the management station that initiated the request). This information is forwarded to the EM MIS by EM's SNM Event Forwarder (`em_snmfwd`).

As illustrated in Figure 11-1, RPC proxy agents use Remote Procedure Call (RPC) protocol (over TCP/IP) to communicate with a management station. However, a RPC proxy agent may use a different management protocol in gathering information from other agents. In the example in Figure 11-1, SNM's Simple Network Management Protocol (SNMP) proxy agent (`na.snmp`) is used to manage devices that support the SNMP protocol.

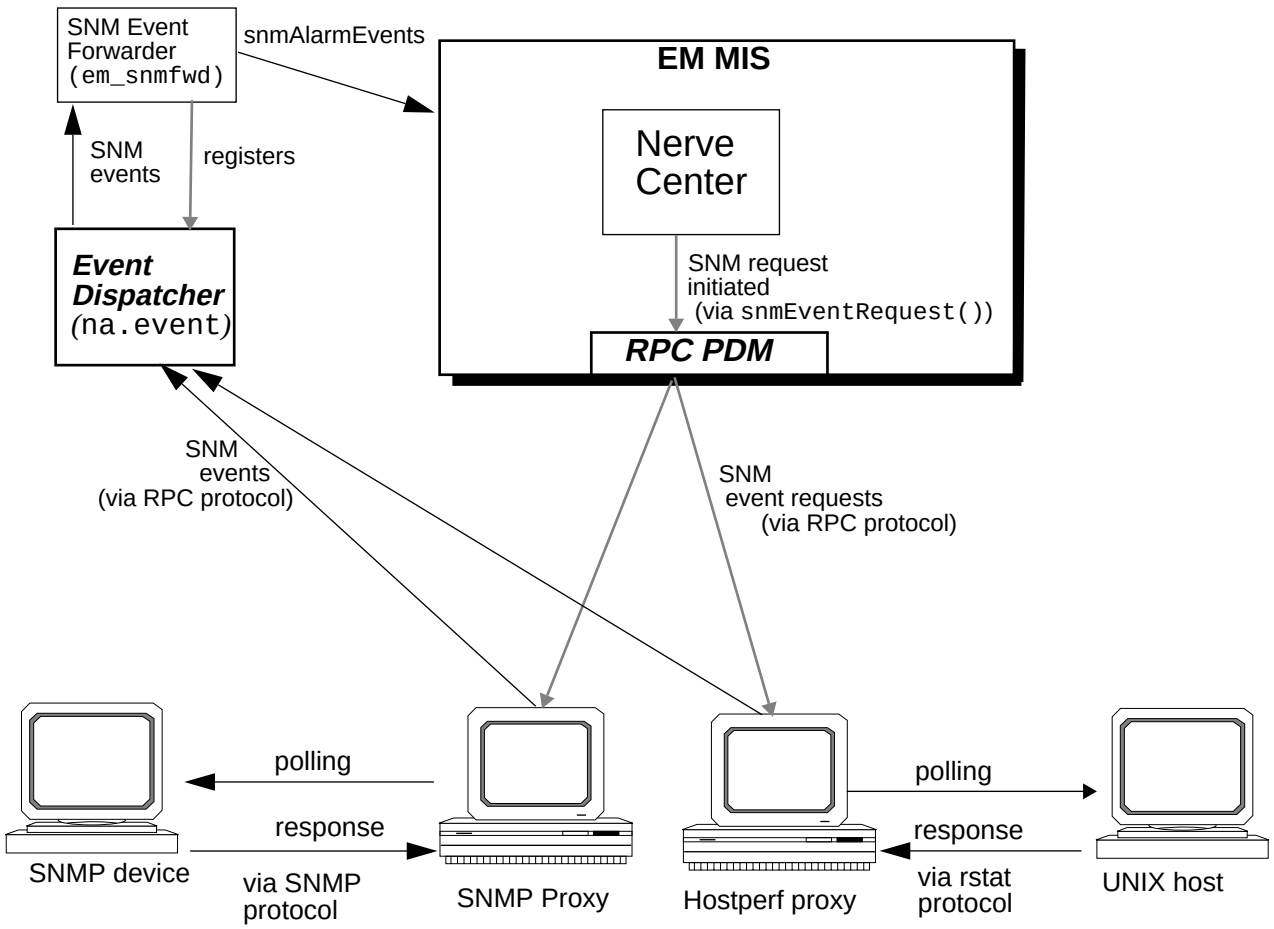


Figure 11-1 Using SNM Event Requests with Solstice EM

For information on the installation of RCP proxy agents, refer to the *Solstice Enterprise Manager Installation Guide*.

Note – SNM events that are received by SunNet Manager Consoles managing segments of your network can also be forwarded to the EM MIS using Cooperative Consoles. This type of distributed management scenario is described in Chapter 5, “Using Cooperative Consoles with Enterprise Manager.”

For general information on using SunNet Manager RPC agents with Solstice EM, refer to Chapter 4, “Device Management Using RPC Agents.”

11.2 Nerve Center’s SNM Event Request Capability

The Nerve Center module in the MIS contains the request-handling capabilities of EM. Nerve Center requests are based on request templates, which are built using the Request Designer application. A key building block in request templates are request *conditions* — sets of instructions defined using the EM Request Condition Language (RCL). RCL provides two built-in functions, `snmEventRequest()` and snm`KillRequest()`, for starting and stopping SNM event requests. For general guidance in building request templates, consult Chapter 10, “Building Request Templates.” For information on the Request Designer and Request Condition Language, refer to Part 3 of the *Solstice Enterprise Manager Reference Manual*.

SNM event requests can be launched from the EM management station using the request-handling capabilities of the EM Nerve Center. Request templates built using the EM Request Condition Language (RCL) can initiate SNM event requests via the RCL `snmEventRequest()` function. When SNM event requests are launched at target managed objects, the Nerve Center communicates the request to the appropriate SNM agent or proxy through the RPC Protocol Driver Module (PDM) in the MIS.

When the `snmEventRequest()` function initiates a request, the following information is passed to the target SNM agent or proxy:

- The agent attribute
  For example, the `mempct` attribute, supported by the `hostmem` agent, reports the percentage of network memory in use on a machine running SunOS 4.x. A request might use this attribute to generate an SNM event if the network memory usage on a router is greater than 80%.

- The agent attribute group
  For example, the `load_stats` group, supported by the `cpustat` agent, reports load statistics for a particular CPU in a multi-processor machine.
- The relation used to define the threshold
  Relations such as Equal To, Greater Than, Not Equal To, can be used to define situations that generate SNM events if they occur.
- The threshold value to test for
  For example, if the threshold value is 1 and the relation is Not Equal To, then **Not Equal To 1** is the threshold that will generate an SNM event if it occurs for the specified attribute.
- The SNM priority of an alarm generated if the threshold obtains
  The possible priorities for SNM events are High, Medium, or Low. These correspond to `perceivedSeverity` values of EM alarms as indicated in Table 11-1 below.
- Polling interval (in seconds)
  The delay between polls of the target object by the SNM agent or proxy.
- The number of times the device should be polled before terminating the request
  This can be unlimited, or a finite number of polls can be specified. (The number 0 is used to indicate that polling should continue indefinitely.)
- A specific resource to target within the agent system
  For example, a specific file system can be checked for its percent of capacity in use via the `diskInfo` agent. A request could be defined to generate an SNM event if the `capacity` attribute value is greater than 90% on the target file system.

Once the Nerve Center has initiated the SNM request, polling of the managed resource at the specified intervals is handled by the SNM proxy rather than the Solstice EM Nerve Center, thus minimizing network traffic and the polling work required of the Nerve Center.

When an SNM agent or proxy first receives a request, two agent processes are started: one is a parent process and one is a child process to handle the request. Subsequent requests sent to the same agent will cause the agent to start additional child processes.

Information on the attributes and attribute groups supported by SNM agents and proxy agents can be found in the *Site/SunNet/Domain Manager Reference Manual*.

11.3 SNM Alarms

When a critical threshold defined in an SNM request is detected by the SNM agent, a response — called an *event* in SNM terminology — is sent via RPC protocol to the SNM Event Dispatcher (`na.event`). The SNM Event Forwarder daemon (`em_snmfwd`) registers with the SNM Event Dispatcher to receive incoming SNM events. SNM events received by `em_snmfwd` contain the following information:

- Name of the target system where the managed resource resides
- Name of the system which sent the event
- The pertinent agent attribute, and the threshold which obtained, thus causing the event
- Priority of the event
- RPC number of the agent

If the sending agent is a proxy agent, the target system name and the agent system name will be distinct.

The SNM Event Forwarder uses the SNM event to build an `snmAlarmEvent`, which will be sent to the EM MIS. The Event Forwarder maps SNM event severities to the `perceivedSeverity` values used by the Alarm Service in the manner indicated in Table 11-1.

Table 11-1 Mapping of SNM Event Severities

SNM Event Severity	perceivedSeverity Value	Default Icon Color
Low	Minor	Cyan
Medium	Major	Orange
High	Critical	Red

The attributes in the `snmAlarmEvent` include the following:

- `perceivedSeverity` — This is mapped to SNM priorities as indicated in Table 11-1.
- `managedObjectInstance` — This represents the target element within the agent system.

- `probableCause` — This indicates the threshold that was defined in the SNM request; the event was generated because this threshold obtained.
- `additionalText` — This contains the name of the RPC agent and the threshold that generated the event.
- `notificationIdentifier` — This a timestamp of the moment when the MIS sent the SNM event request; this enables the MIS to identify the request that is responsible for the event.

For the structure of snmAlarmEvents, refer to the "Standard Event Notifications" appendix to the *Solstice Enterprise Manager Reference Manual*.

Because snmAlarmEvents are, by default, not logged to the AlarmLog, they are not monitored by the Alarm Service and therefore do not affect fault status indication (icon color) in the Viewer. By default, only alarms logged to the AlarmLog affect fault status color in the Viewer. The Alarm Service is a module in the MIS that monitors the alarm log and uses the highest severity of outstanding (uncleared) alarms to determine the fault status color for the device. For information about the Alarm Service, refer to the "Alarm Service" chapter in the *Solstice Enterprise Manager Reference Manual*.

However, a request that listens for incoming snmAlarmEvents can use the RCL alarm-logging functions to post appropriate nerveCenterAlarms to the AlarmLog. The RCL subscription functions enable a request to listen for specified types of events. Thus, you will want to design your SNM event request templates to listen for incoming snmAlarmEvents from SNM agents and take appropriate action.

11.4 Building SNM Event Request Templates

An example of a Nerve Center request template that initiates an SNM event request is the `DeviceReachablePing` template, shipped with Solstice EM. Examining this template may give you some ideas for building other SNM event request templates.

When a DeviceReachablePing request is launched against a target host, an SNM event request is sent to the ping proxy agent with a polling interval of 30 seconds and a threshold of reachable Not Equal To true. A high priority SNM event is generated by the ping proxy agent if it finds the target device not reachable when it polls. As indicated in Table 11-1, the SNM Event Forwarder translates the high priority SNM event into an snmAlarmEvent with a `perceivedSeverity` of critical. The DeviceReachablePing request listens for

incoming snmAlarmEvents from the target device and posts a nerveCenterAlarm with a `perceivedSeverity` of critical if an SNM event is received.

While it is listening for incoming SNM events, the `DeviceReachablePing` request counts the elapsed time since any previous “Device Down” event, and if the elapsed time is greater than the timeout used by the ping proxy agent in polling the device, the DeviceReachablePing request assumes the device is up and posts a minor alarm to indicate the device is up after having been down.

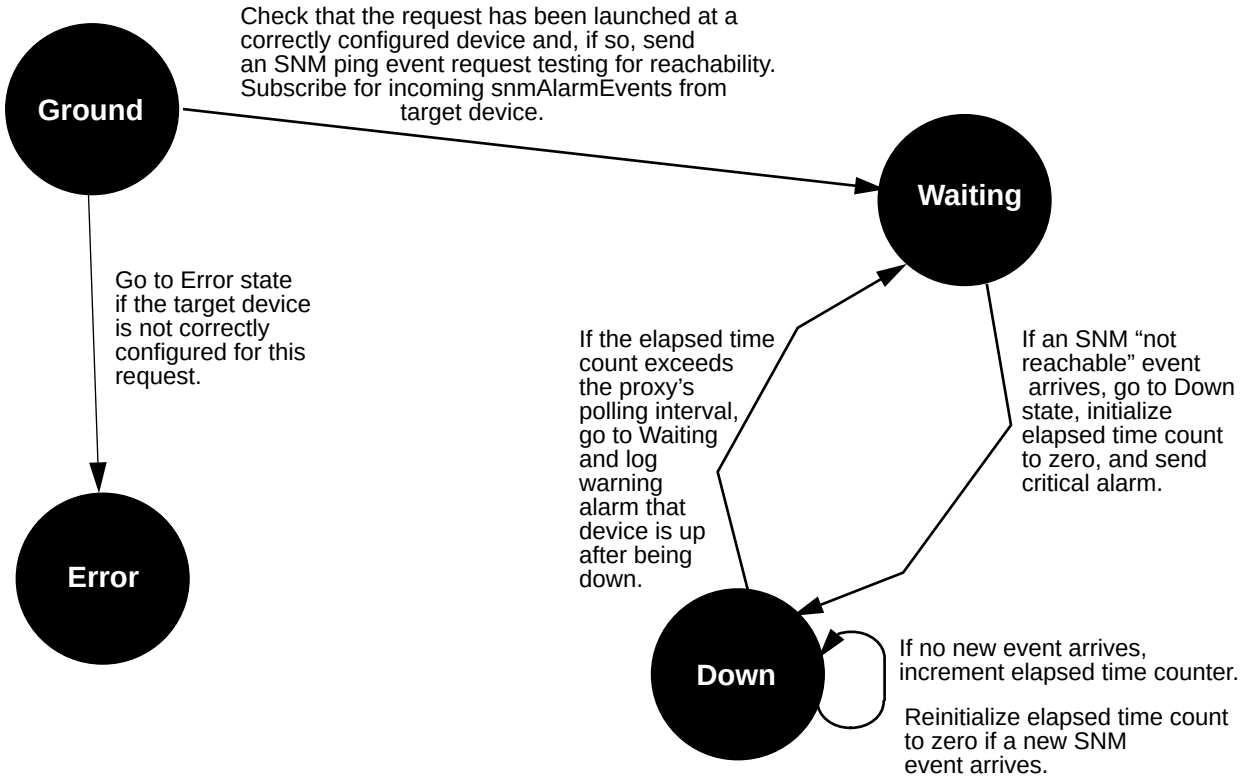


Figure 11-2 State Machine Diagram for `DeviceReachablePing` Template

The transition from the Ground state to the Waiting state is where the request's initialization is accomplished:

- The target device is checked to determine if it is correctly configured for a ping request. If the target device does not support the request, the request transitions to the Error state and an appropriate warning alarm is logged.
- The RCL `subscribeOI()` function is used to subscribe for incoming snmAlarmEvents from the target device.
- The RCL `snmEventRequest()` function is used to send the SNM event request to the ping proxy agent.

Each of these tasks is carried out by a separate condition defining a transition from the Ground state to the Waiting state. The first of these transitions is defined by the `get_rpcAgent_name` condition:

```
$num = numElements(&$pollFdnSet);
$count = 1;
while( $count <= $num))
{
    $numstr = AsnToStr($count,TRUE);
    $dn = Extract(&$pollFdnSet,$numstr);
    $dn1 = Extract(&$dn,"distinguishedName");
    $dnstr = AsnToStr($dn1,TRUE);
    $res = AnyStr($dnstr,"RPC");
    if ($res == TRUE)
    {
        $dn2 = Extract(&$dn1,"3");
        $dn3 = Extract(&$dn2,"1");
        $hostname = Extract(&$dn3,"attributeValue");
        $rpc_dn = appendRdn($dn,"/agentId=\"ping-reach\"");
        $count = $num+1;
    }
    $count = $count+1;
}
false;
```

Figure 11-3 get_rpcAgent_name Condition

Using the RCL `numElements()` function, the first statement in the condition determines how many managed objects are configured for this device. This information is passed to the request in the $pollFdnSet variable when the request is launched against a target device in the Viewer. The condition then

uses a WHILE loop to examine the distinguished name (FDN) pointing to each such object to determine if the device is manageable via RPC. If the device is manageable by RPC, the RPC proxy table for the device (which “contains” under it the various RPC agent attribute groups supported by that device) will be represented in the $pollFdnSet.

The Boolean variable $res is set to true if the device does support RPC, false otherwise. This condition is followed in the template by a transition defined by the `check_for_rpc` condition. If $res is false, that condition causes a transition to the Error state.

The get_rpcAgent_name condition also extracts from the RPC FDN the hostname of the device, which will be used in building the SNM event request. The RCL appendRdn() function is used to point `$rpc_dn` to the ping agent reach group contained under the RPC proxy table. This will be passed to the RCL `snmEventRequest()` function when initiating the SNM event request.

Note that the get_rpcAgent_name condition ends with a line that says “false;”. This is to ensure that this condition does not cause a transition to the Waiting state. If this condition did cause a transition to the Waiting state, the conditions initiating the SNM event request and subscribing for incoming SNM events would never be executed. The conditions defining transitions are executed by Nerve Center in the order they occur in the template. The conditions in the later transitions out of the Ground state would not be executed by Nerve Center if any of the earlier conditions evaluate to true. If a condition defining one of these transitions evaluates to true, the request transitions to the Waiting state. Thus, if `check_for_rpc` evaluates to true, the request transitions to the Error state and the conditions initiating the SNM event request and subscribing for SNM alarms are never evaluated.

11.4.1 Subscribing for SNM Events

The subscription for snmAlarmEvents occurs in the following condition:

```
subscribeOi(“snmAlarmEvent”,””,$dn);
false;
```

Figure 11-4 `subscribe_snmAlarmEvent` Condition

The subscribeOi() function is used to subscribe for events from a specified object. Note that `$dn` — the RPC proxy table for the target device, not the FDN pointing to the ping reach group (`$rpc_dn`), is the object that is the target of the subscription. For RPC requests, the RPC proxy table FDN contained in `$pollFdnSet` must be used for both event subscriptions and logging of alarms against the device.

As with the get_rpcAgent_name condition, the subscribe_snmAlarmEvent condition ends with "false;" to ensure that the request does not leave the Ground state after evaluating this condition but proceeds to the next transition in the Ground state.

11.4.2 Sending an SNM `ping` *Event Request*

After subscribing for snmAlarmEvents from the target device, the `DeviceReachablePing` request sends the SNM event request to the ping proxy agent. This is accomplished in the `send_ping_reach` condition:

```
$tmp = "{agentHost \"";
$request_timeout = 30;
$tmp = StrCat($tmp,$hostname);
$s1 = "\",agentProgram 100115, agentVersion 10, timeout 30,interval 10,group
\"reach\",threshold {\"reachable\",21,1,\"0\",high}}";
$tmp = StrCat($tmp,$s1);
$handle = 0;
print($tmp);
snmEventRequest($rpc_dn,$tmp,&$handle);
true;
```

Figure 11-5 `send_ping_reach` Condition

The SNM event request parameters are passed to the snmEventRequest() function as the string $tmp. The hostname, which was extracted in the `get_rpcAgent_name` condition, is concatenated with the other parameters. If the RPC `proxyhost` setting for $hostname is configured as `localhost`, the request is sent to the ping proxy agent on the MIS system. However, polling by SNM agents can be offloaded to other machines if the managed resource is configured with a `proxyhost` other than localhost. (This can be configured in the Discover Properties window, when doing discovery of RPC-manageable devices on TCP/IP networks, or it can be configured manually using OCT.)

The event request passes the address of $handle to Nerve Center. This variable can be passed to `snmKillRequest()` function to kill the request. Note that handle must be initialized before calling `snmEventRequest()`.

The parameters passed in the event request string are as follows:

- `agentHost` <*hostname*> — <*hostname*> was obtained from $pollFdnSet in the `get_rpcAgent_name` condition. This is the target device for the SNM event request.
- `agentProgram 100115` — The RPC number of the ping proxy agent.
- `agentVersion 10` — This is the software version number. This is contained in the entry for the agent in the `/etc/initd.conf` file. For example, 10 is the version number for `na.snmp` in the following inetd.conf entry:

```
na.snmp/10  tli rpc/udp  wait  root  /opt/SUNWconn/snm/agents/na.snmp  na.snmp
```

- `timeout 30` — This is the length of time the ping proxy agent will wait for a response from the device before sending an alarm.
- `interval 10` — The ping proxy agent polls the target device every 10 seconds.
- `\"reach\"` — The name of the attribute group used in this request.
- threshold { <*threshold*> } — The name “threshold” introduces a set of values that define the threshold that the agent is to check for:
  - `\"reachable\"` — The name of the attribute whose value is checked.
  - `21` — The data type of the operands of the relational operator.
  - `1` — The relational operator. A value of 1 indicates the operator is Equal To.
  - `\"0\" – “0”` indicates false in this case.
  - `high` — The priority to assign to the SNM event generated if the threshold obtains.

Thus, the ping proxy agent is instructed to check for reachability Equal To false and generate an SNM event notification if this should occur.

11.4.3 Waiting for a Response to the Event Request

After the DeviceReachablePing request subscribes for snmAlarmEvents from the target device and sends the SNM event request to the ping proxy agent, the request transitions to the Waiting state. The request “sleeps” until it is “woken up” by the arrival of an snmAlarmEvent. This happens when the `is_snmAlarmEvent` condition evaluates to true:

```
$messType == 0;
```

A `$messType` of 0 indicates that the request was woken up by the arrival of an event. The arrival of an snmAlarmEvent indicates that the target device is not reachable. The request then transitions to the Down state and executes two conditions as actions in the transition. One of these actions logs a nerveCenterAlarm:

```
alarmStr(1,”Device Not Responding to Ping”);
```

This alarm is logged against the device indicated by the request’s $pollfdn value. When the request is first launched, this is set by Nerve Center to point to the first object in $pollFdnSet. This critical alarm will cause the icon of the target device to turn red in the Viewer. The string passed to the `alarmStr()` function appears in the `additionalText` field for that alarm in the Alarm Manager.

The other action in the transition from the Waiting to the Down state initializes a counter:

```
$time_counter = 0;
```

At this point the request knows that the device is down. But it would also be useful to be notified if the device comes back up. The request can assume that the device is back up if it stops receiving “Device Down” events from the ping proxy agent for a length of time that is longer than the timeout that the ping agent is using in waiting for responses from the target device. The request has set this timeout value to 30 seconds in the SNM event request. Therefore, the DeviceReachablePing request counts the time elapsed after each incoming

"Device unreachable" event, and when it stops receiving such events for a period longer than the request timeout being used by the ping agent, the request assumes the device is back up.

After the request transitions to the Down state, it loops back to that state so long as the following condition evaluates to true:

```
$messType == 0;
```

Figure 11-6 another_event Condition

Each time the request loops back from the Down to Down state due to the arrival of a new SNM event notification from the ping proxy agent, the time counter is reinitialized to zero.

Note that the polling interval is every 20 seconds in the Down state. If no new SNM event arrives after 20 seconds, the another_event condition will evaluate to false and the request will then evaluate the following condition:

```
$fake = topoNode;
$time_counter = $time_counter + 10;
$time_counter > $request_timeout;
```

Figure 11-7 wakeup_count Condition

The purpose of the first statement "`$fake = topoNode;`" is to retrieve some attribute (it may be irrelevant to the purposes of the request, as in this example) in order to force the request to be "woken up." If the request is not woken up, this condition would not be evaluated.

The wakeup_count condition increments the time counter and then checks to determine if the time elapsed since the last SNM event is greater than the ping proxy request timeout. If it is not, this condition will evaluate to false and will not cause a transition back to the Waiting state; the request then continues to loop in the Down state. If this condition does evaluate to true, the request assumes that the ping proxy agent is no longer sending "Not reachable" event

notifications because the device is back up. This causes the request to transition back to the Waiting state, and in the transition a minor alarm is logged by the `deviceBackUpWarningAlarm` condition:

```
alarmStr(4,"Device is up after being down");
```

This is a minor alarm. This will only turn the icon cyan, however, after a user clears the previous critical alarm in the Alarm Manager. If you wanted to implment an automatic "decay to cyan" feature, to automatically change the icon to cyan when a device becomes available after being unreachable, you could modify the DeviceReachablePing template to issue a "cleared" alarm before logging the minor alarm. The following condition would send a "cleared" alarm to clear the previous critical alarm:

```
alarm(5)
```

If the request did not clear the previous critical alarm, the icon would remain red because the Alarm Service sets fault status color to the highest severity of uncleared alarms. An outstanding critical alarm always takes precedence over alarms of lesser severity. The minor alarm only causes the icon to "decay to cyan" if the previous critical alarm has been cleared.

Request Designer 12

Overview	*page 12-1*
Starting the Request Designer	*page 12-3*
Using the Request Designer	*page 12-3*
Edit States Window	*page 12-9*
Transitions Window	*page 12-10*
Conditions Window	*page 12-16*
Poll Rates Window	*page 12-17*
Severities Window	*page 12-18*
Graphical State Diagram Display	*page 12-18*
em_ncimport and em_ncexport Utilities	*page 12-22*

12.1 Overview

Requests are the series of activities through which the Solstice EM *Nerve Center* polls for the *attributes* of managed objects or receives *notifications* from the agents of managed objects, or both. A request is typically initiated when a *request template is launched at a target object.* The Request Designer is an application that allows you to create request templates.

States, poll rates, and conditions are the building blocks of request templates. Each template is made up of multiple states, with, potentially, multiple transitions between those states. A request state may be thought of as an internal representation of a state (such as Up or Down) of a network resource.

A *condition* is a set of instructions written in the *Request Condition Language* (RCL). Conditions can play two roles in *requests*:

- A single condition can be used to define when a request will undergo a *transition* from one *state* to another (or loop back to the same state). You must have exactly one condition associated with each transition. Where more than one transition out of a given state exists in a template, each defined by a distinct condition, the *Nerve Center* evaluates the conditions in the order they are entered in the *request template*.
- A second role of a condition is as an *action*, taken in response to a transition. A condition is one of three types of action, the others being the sending of mail and the invocation of a Unix command. Multiple conditions can be invoked as actions resulting from a single transition. When multiple actions result from a given transition, Nerve Center executes them in the order entered in the request template.

As part of a request template, you can enter RCL statements that cause an event to be treated as an alarm and to be logged to an alarm log. The Request Condition Language is described in Chapter 13, “Request Condition Language.”

The alarm logging activity of Nerve Center affects the Alarm Manager, which displays a summary of alarms in one or more logs, and the Log Manager, which displays log records for events you have chosen to log. Both the Alarm Manager and Log Manager are described in the *Solstice Enterprise Manager Reference Manual*. Nerve Center alarm logging also has an impact upon the color of icons in the Viewer. The MIS Alarm Service module monitors the alarm logs and updates the color of icons in the Viewer to reflect the severity of alarms logged against the managed objects represented by those icons. The default mapping of color to severity is described in Table 12-3. Nerve Center controls this mapping and you can change it through the Request Designer’s Edit➤Severities option, described in Section 12.7, “Poll Rates Window.”

12.2 Starting the Request Designer

To use the Request Designer, a Solstice EM MIS must be running and the Request Designer must be able to communicate with it.

You can start the Request Designer by selecting the Request Designer icon in the Application Launcher window. Also, the application is brought up if you click the Create or Modify buttons in the Viewer's Requests window. (See the "Viewer" chapter in the *Solstice Enterprise Manager Reference Manual* for a description of the Requests window.)

You can also invoke the Request Designer, and have it connect to an MIS, by using the following command line format:

```
host% em_reqedit [ -host <hostname> ]
```

The <*hostname*> option is used to specify the name of the machine where the MIS is running. If you start the Request Designer from the command line, and you are a logged on as a non-root user, you receive a Login window if password authentication has been activated for EM. To proceed, enter your password and click OK. Your access to Request Designer functions depends upon the permissions granted to you through the Solstice EM Access Manager.

The Request Designer offers two modes of interaction — through a text-based window or through a graphical display, called a "State Diagram" window. Upon invoking the Request Designer, the application comes up in its text mode. In the following subsections, we first describe the use of the text-based window. Much of this information applies to the graphical display as well. See Section 12.9, "Graphical State Diagram Display for instructions on the use of the graphical display.

12.3 Using the Request Designer

The following sections explain the use of the Request Designer.

12.3.1 Main Window

The Request Designer main window consists of the following components:

- A menu bar that offers File, Edit, and View menus

- A display area, showing the textual description of the current request template
- An area near the bottom with the States, Transitions, and Conditions buttons

Each of these areas is described in detail in the following sections.

12.3.1.1 Display Area

The Request Designer has a read-only text display area beneath the menu bar. The name of the current template appears in the footer of the window. The text display area has the column headings for State, Transition, Condition, and Action. As you add states, transitions, conditions, and actions in building a template, the names of these components are aligned in the appropriate column.

The display area contains the text for the currently selected request template. Contents of the area change in response to File➤Open..., File➤Delete... and File➤New commands. The File➤Open... option allows you to open (and thereby display) an existing template. Invocation of File➤New clears the display area.

The display area records the states and transitions you create or modify using the States and Transitions windows.

12.3.1.2 States, Transitions, and Conditions Buttons

These buttons invoke the windows of the same names. The States, Transitions, and Conditions windows are described starting with Section 12.4, “Edit States Window.”

12.3.1.3 Text-display Message Area

The Request Designer has a message area along the bottom edge of the main window. In this area, the Request Designer displays the name of the current template. Error messages are also displayed here.

12.3.2 File Menu

The options in the File menu are as follows:

New

Select the New option to have a “fresh” request template window displayed. If you select New when you have unsaved changes to a template, the Request Designer asks you if you want to save those changes before presenting you with a new, empty template. The message area displays a current template name of “NoName.”

Open...

Select Open... to receive the Open Template window. This window lists available templates by name and description. To open a template, select the template name and description then select Open. Or simply double-click on a template name and description. To delete a template (which removes the template object from the MIS), select the template name, then select Delete.

Note – The Request Designer does not allow you to change a request template that is in use. The Request Designer determines whether a template is in use at the point where you attempt to save the template.

Delete...

The Delete... option also invokes the Open Template window. To delete a template (which removes the template object from the MIS), select the template name, then select Delete.

Save

Select Save to save the current template. The Request Designer saves the template under the name used when it was opened. To save a new template, or to save a previously opened template under a new name, use the Save As option. The Save option does not allow you to save partial or incomplete templates. Use the File➤Export Current... option to save incomplete templates.

Save As...

Select Save As to save the current request template under a new name. The Request Designer saves a template (and stores the template in the MIS) by its name. Enter the new name in the Name field and select OK to save the template. If you attempt to save the template under a name that is in use, the program gives you the option of overwriting the existing template or of

selecting another name. The Save As option does not allow you to save partial or incomplete templates. Use the File➤Export Current... option to save incomplete templates.

Import

Select Import to invoke the Import window (shown in Figure 12-1). This window allows you to select which type of component you want to import from a previously exported ASCII file — Templates, Conditions, or Poll Rates. You can select the directory and the specific file as the source of the import. The action of the Import option depends upon how the imported file was previously saved:

- If the imported file was previously saved using the Export option, the imported components are loaded into the MIS.
- If the imported file was previously saved using the Export Current option, the template is loaded into the Request Designer but not into the MIS.

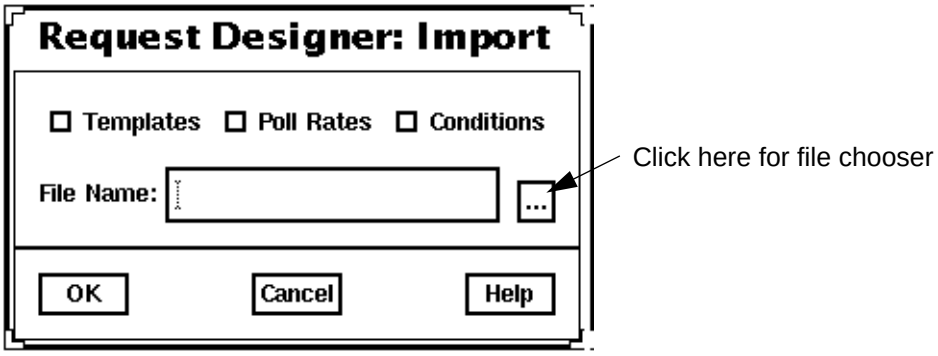


Figure 12-1 Import Window

Export...

Select Export to invoke the Export Customized window. This window allows you to select Templates, Conditions, and Poll Rates to export to an ASCII file. You can select the directory and filename to which the selected items are to be exported. Depending on your selection of component type from the buttons in the top of the window, the existing components of that type are displayed for your selection. In the example in Figure 12-2 the template IsSnmpSystemUp has been selected for export to a file named `snmp.template`.

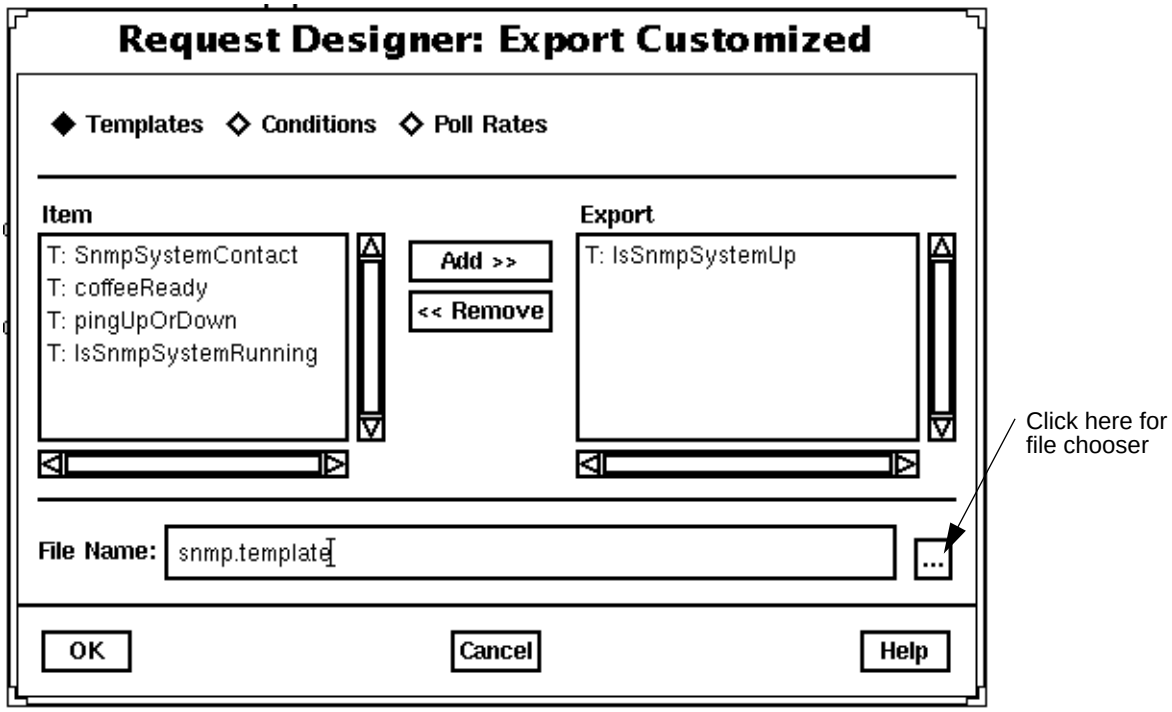


Figure 12-2 Export Customized Window

Export Current...

Allows you to save the contents of the Request Designer main window into an ASCII text file. You can use this option to save incomplete or partial templates in order to continue work on them later. You can reload the exported template using the Import option.

Note – The Request Designer Export and Import functions can also be accessed through the `em_ncimport` and `em_ncexport` command-line utilities. Refer to Section 12.10, “em_ncimport and em_ncexport Utilities,” for more information.

Exit

Select Exit to leave the Request Designer. Any unsaved changes to a template are lost. Use the Export Current option to save incomplete or partial templates before exiting.

12.3.3 Edit Menu

The Edit menu offers the following options, which invoke the corresponding windows:

- States — see Section 12.4, “Edit States Window”
- Transitions — see Section 12.5, “Transitions Window”
- Conditions — see Section 12.6, “Conditions Window”
- Poll Rates — see Section 12.7, “Poll Rates Window”
- Severities — see Section 12.8, “Severities Window”

The foundation-building work of the Request Designer is performed in the States, Transitions, and Conditions Windows. These windows can also be invoked by selecting the appropriate button at the bottom of the main window.

You must create states before creating transitions. States and transitions are specific to a given request template; conditions can be used across multiple request templates.

Severities and poll rates are associated with states. You may find that the poll rates already defined are more than adequate for your needs.

12.3.4 View Menu

The options in the View menu control the way in which the template is displayed. The Request Designer offers two modes of interaction — through a text-based window or through a graphical display, called a “State Diagram” window. The Request Designer is in text mode when it is initially invoked.

Graphical

Select this option if you want to view the template solely in the graphical State Diagram display.

Text

Selects text-only display of the template. Invoking this option closes the graphical State Diagram window if it is in use.

Both

Select this option if you want to use both the text and graphical displays of the same template. The two modes are not exclusive; changes made in one are reflected immediately in the other. Even if you prefer to work in the text-based display, you might find it useful to have the graphical display open, to see a picture of the template as it emerges.

12.4 Edit States Window

Select the States button at the bottom of the main window (or the Edit➤States... menu option) to display the Edit States window.

You can create any number of states for a given request template. However, a maximum of nine states (including Ground) can be displayed in the Request Designer's graphical display. If you have more than nine states, the Request Designer displays the most recently entered eight states, plus Ground.

The components of the Edit States window are described as follows:

Tabular Display

This is a read-only area that is updated after you perform an Add, Delete, or Modify operation. It is also updated when you invoke File➤Open... and open a request template. The display area lists the states that are in the currently displayed template.

For a given state, the display area lists the state name and the name of the severity and the poll rate associated with that state.

State Field

Displays the name of a currently selected state, if any, from an open request template. If you want to create a new state, enter a name in this field. Names must be of 64 or fewer alphanumeric characters and must *not* contain whitespace characters.

Description Field

The contents of this field describe a state. This field is updated when you make a selection in the States window display area. Descriptions are strings of 256 or fewer characters. Whitespace *is* allowed in descriptions.

Poll Rate Button

The Poll Rate button provides a menu of available poll rates. Poll rates supplied with Solstice EM are shown in Table 12-2. You can create or modify a poll rate in the Edit➤Poll Rates window.

Severity Button

The Severity button provides a menu of available severities. Severities supplied with Solstice EM are shown in Table 12-3.

Note – "Severities" attached to *states* are internal to the Request Designer application. They do not cause alarms to be logged nor do they cause changes in icon color in the Viewer. The severity value attached to a Nerve Center alarm is determined by the value passed in the <*severity*> parameter of an RCL alarm logging function, such as `alarm()`, `alarm_oi()`, or `alarm_str()`.

In the menu of severities, select a severity you want for the selected state. You can customize the colors associated with severities by invoking the Edit➤Severities option.

If you select a state in the tabular display and values such as severity or poll rate, you can then alter the state in the template by clicking on the Modify action button.

12.5 Transitions Window

Select the Transitions button at the bottom of the Request Designer main window (or the Edit➤Transitions... menu option) to display the Transitions window (shown in Figure 12-4). All operations relating to transitions are performed in this window.

Order Conditions

Select Order Conditions to display the Order Conditions window (shown in Figure 12-3). This window allows you to change the order of transitions from a specified state within the current template. To select the state whose transitions you wish to reorder, select a state from pulldown menu for the Select State field.

Each transition is associated with a condition that determines when that transition will occur. The order of these conditions in the template determines the order in which the Nerve Center evaluates them. After a state has been selected from the Select State menu, the transitions out of that state are listed in their current order. Click on the Move Up or Move Down buttons to move the position of the selected transition.

Changes made in this window are shown immediately in the text and graphics template display areas.

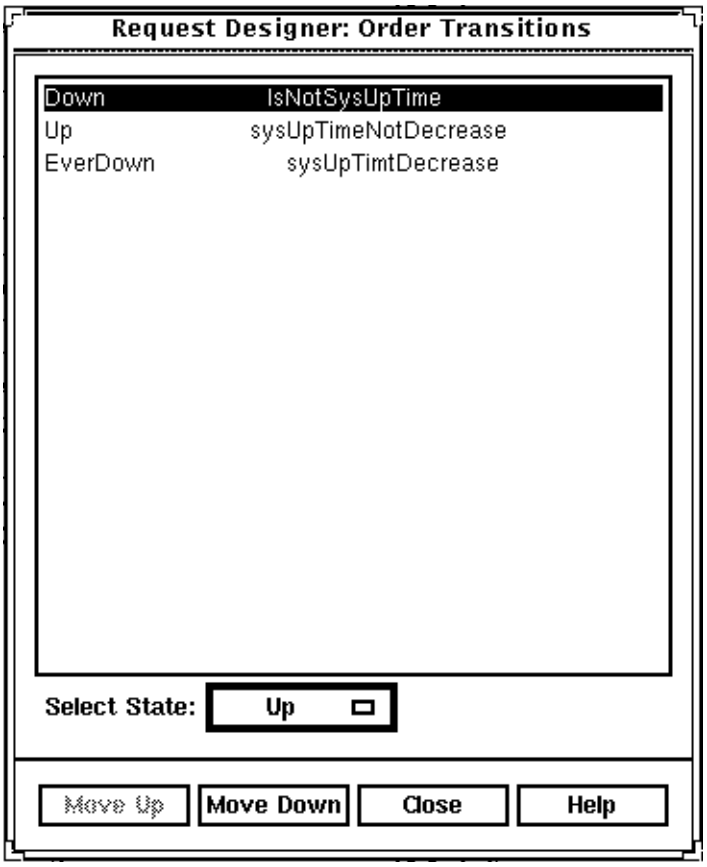


Figure 12-3 Order Transitions Window

Display Area

This is a read-only area that lists all transitions leaving the currently selected "From" state.

From

"From" here refers to the state *from* which a request is making a transition. The From menu shows the available states, that is, states that you have created. When you select an available state, all transitions leaving that state are displayed in the display area above.

To

The To menu allows you to select the state *to* which a request is making a transition. This may be the same state as the From state (i.e. a transition may loop back to the same state). The To menu shows the available states, that is, states that you have created.

Condition

The Condition menu allows you to select a condition to be the test of whether to move from one state to another. A single condition is used to define when a transition will occur. If there are multiple transitions out of a given state to a variety of other states, each such transition requires a condition to define when it occurs. You can define multiple transitions between the same two states — for example, from Up to Down — each defined by a different condition.

The Condition menu shows the available conditions, that is, conditions that exist in the MIS.

Use the Conditions window, described in Section 12.6, “Conditions Window to create a new or modify an existing condition.

Action

The Action menu allows you to select a set of actions that will be taken when a transition occurs.The Action menu options are described in Table 12-1.

Table 12-1 Action Menu Items

Action	Description
<none>	No action taken.
UNIXCMD	The name of command with any required parameters. For example, for `netstat -rn`, you enter `netstat` in the Command field and `-rn` in the Arguments field.
MAIL	An electronic mail address and message. For example, `verma@halcyon` in the Address field and `CPU usage exceeded 90%` in the Message field. By default, the mail that results from an action has a subject “Problem with Node.”
CONDITION	The name of a condition as you created and saved it in the Request Designer (which saves it into the MIS).

If you select CONDITION, the Name field is activated. From the Name menu, you can choose a condition from the menu of conditions. This menu lists all the conditions that exist in the MIS.

Variables that have been introduced in conditions can also be used for the UNIXCMD or MAIL items in the Action menu.

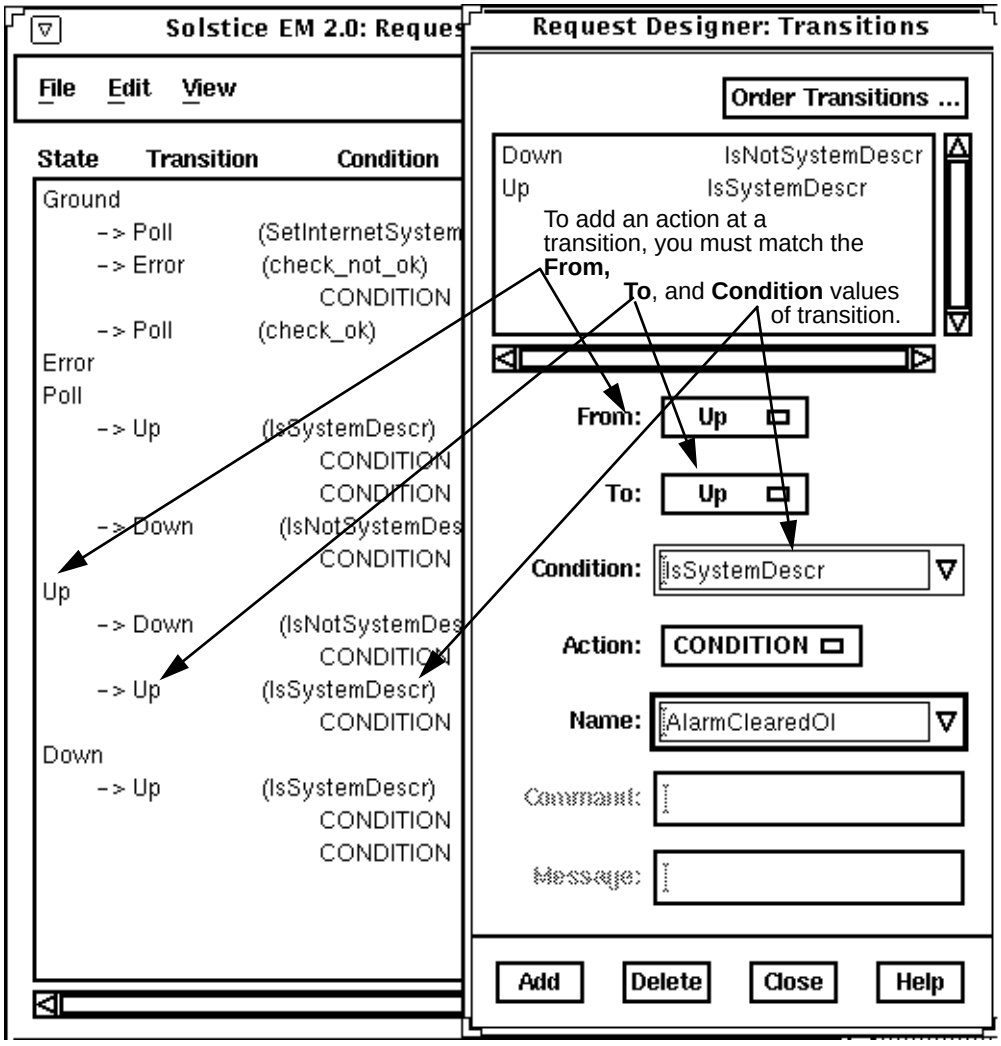


Figure 12-4 Example: Adding AlarmClearedOi as an Action at Up-to-Up Transition

Command and Arguments

These fields are active if you select an action of UNIXCMD. Enter a command as described in Table 12-1.

Address and Message

These fields are active if you select an action of MAIL. Enter a message as described in Table 12-1.

Name

This field is active if you select an action of CONDITION. The Name menu lists the available conditions.

Add

Select Add to add the transition. You can add multiple actions that are to be executed at the same transition. To add a new action to an existing transition, the Transitions window must match the target transition on the From, To, and Condition fields, as shown in the example in Figure 12-4. In that example, the condition AlarmCleared is being added as an action at the transition from Up to Up.

The actions in a transition are invoked by the request in the order they appear in the template. If you want to change the order of actions in an existing transition, you need to delete the actions, then add them in the desired order.

Delete

You can delete one action at a time. To delete an action that is a condition, select the condition from the condition menu and select Delete. To delete an action of UNIXCMD or MAIL, specify both arguments (Command and Arguments for UNIXCMD, Address and Message for MAIL) exactly as you entered them when you created the action.

Note – To delete an entire transition, specify an Action of <none>, then select the From, To, and Condition entries for the transition you want to delete. Then, select Delete.

Close

Dismisses the Transitions window.

12.6 Conditions Window

Select the Edit➤Conditions option (or the Conditions… button at the bottom of the main window) to invoke the Conditions window. Use the Conditions window to create new or modify existing conditions.

The Conditions window displays a list of available conditions. To modify an existing condition or to create a new condition based on an existing condition, activate the condition you want to change. (A single click on an item in the available conditions list displays that condition's name and description; a double-click, in addition to name and condition, displays the condition code.)

If you change the condition name, select Add after making the changes you want to the description and the condition code. If you change the description or code and leave the name unchanged, select Modify.

Note – You cannot delete or modify a condition that is used in a template displayed in the Request Designer's Open Template window, described under Section 12.3.2, "File Menu."

The Conditions window has a sash, a small box divided along the diagonal, located near the right side of the window. The sash allows you to use your mouse to change the proportions between the "available conditions" and the "condition code" displays within the Conditions window.

Error Checking

After you select Add or Modify, the Request Designer performs a syntax check on the condition code. If the code meets the syntactical requirements of the RCL, the Request Designer adds or modifies your condition. If the code does not meet RCL requirements, the Request Designer displays an error pop up window and prevents saving the condition.

If your condition code includes references to agent attributes, the Request Designer checks to determine if that attribute has been defined in a GDMO document loaded into the MIS. If the attribute has not been defined in the MIS, you receive an error message and the Request Designer does not save the condition.

Select the Delete button to delete the currently displayed condition.

12.7 Poll Rates Window

Select Edit➤Poll Rates to display the Poll Rates window. Use the Poll Rates window to create new or to modify existing poll rates. A poll rate specifies the interval at which the Nerve Center checks on a managed object for a given state. A single poll rate is associated with each state. A poll rate consists of a name (alphanumeric characters only, no whitespace) and a number, the number being the number of seconds that must elapse before the Nerve Center makes a poll request.

Poll rates supplied with the product are listed in Table 12-2.

Table 12-2 Poll Rates

Name	**Interval (secs.)**
Fast	60
default_rate	300
VerySlow	21600
Slow	3600
Medium	900
Moderate	300
Pollping	10
Poll	20

12.8 Severities Window

Select Edit➤Severities to receive the Severities window, which, by default, offers you the list of severities shown on Table 12-3.

Table 12-3 Severities

Integer	Name	Default Color
0	Indeterminate	Blue
1	Critical	Red
2	Major	Orange
3	Minor	Cyan
4	Warning	Yellow
5	Normal	Green

A severity describes the degree of importance you attach to a network resource entering a state. The Severities window allows you to change the color that is used to represent a severity. To make a change, select the severity and type in the desired color in the Color field; then click on Modify.

A severity is made up of three items: a name, a number, and a color. For example, “Warning 4 yellow” is one of the supplied severities. The Nerve Center’s mapping of colors to severities (shown in Table 12-3) controls the use of color in the Viewer and Alarm Manager to represent the severity of alarms logged against managed objects.

Note – You cannot change the name or the numeric value of the severities, nor can you add or delete severities. Only integer values in the range 0 to 5 are valid severity values.

12.9 Graphical State Diagram Display

In the Request Designer main window, select View➤Graphical (or View➤Both) to receive the State Diagram window shown in Figure 12-5. In this figure, you start with the single state, “Ground,” which is the required starting point for all request templates.

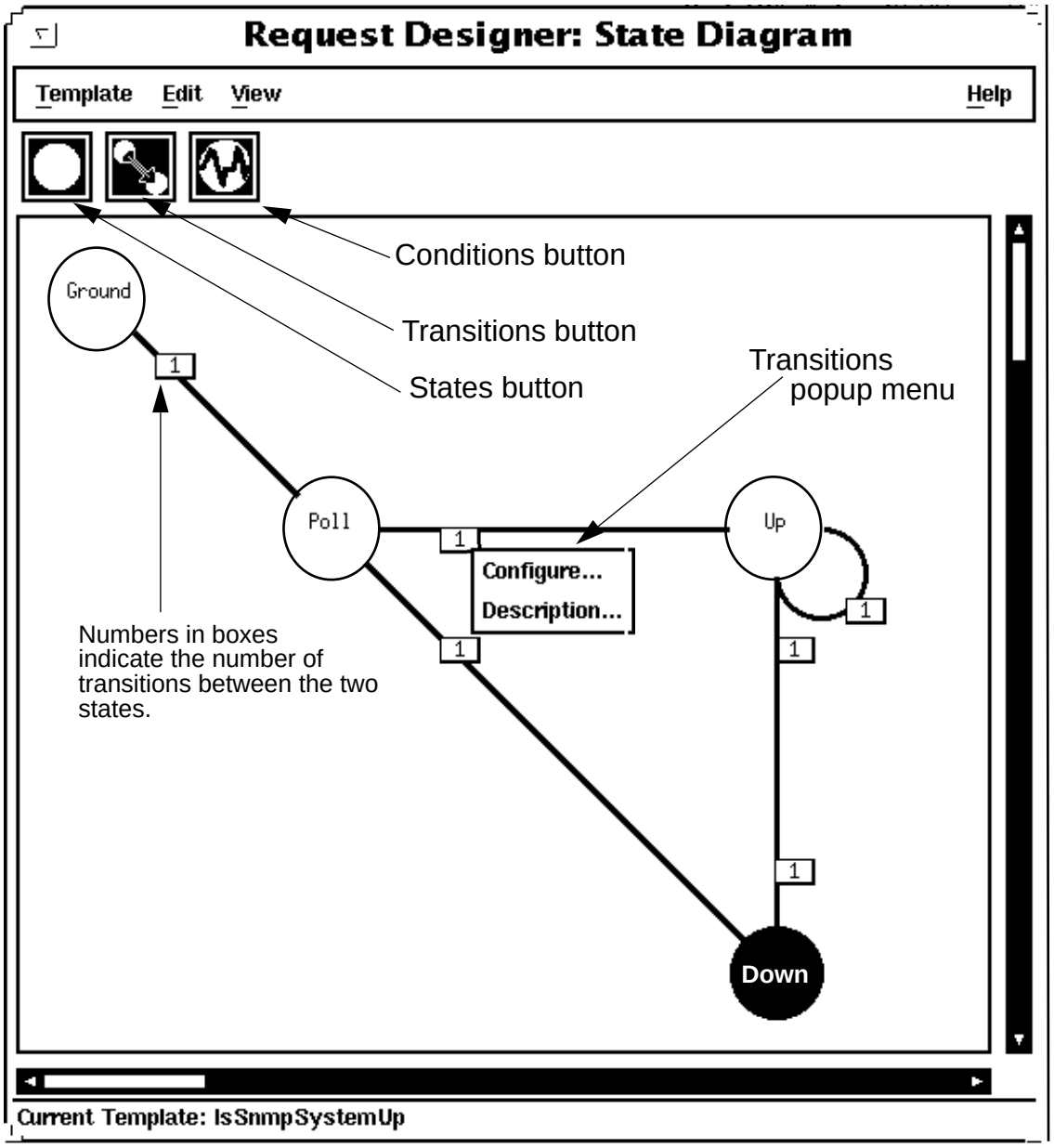


Figure 12-5 Graphical State Diagram Display

The menus at the top of this window are identical to those in the text-based display and are described starting in Section 12.3.2, “File Menu.”

12.9.1 Creating a Template Through the State Diagram Display

The graphical display icons are pointed out in Figure 12-5. These icons correspond to (from left to right) the States, Transitions, and Conditions buttons in the text-based display.

To use the graphical display to create a request template, perform the following steps:

1. **Select the States (leftmost) icon.**
   A new, unconfigured state displays in the graphical display. This state has a name of “NoName.” At the same time, the Configure States window is displayed. This allows you to enter the same data as is accepted by the States window invoked from the text-based display. See the description of that window in Section 12.4, “Edit States Window.”

2. **Enter a name and description and select a poll rate and severity for the new state. Select Add.**
   The name just entered appears in the circle for the new state; the color specified in the state’s severity is also displayed.

Note – At this point, you have the option of creating additional states — first adding a state (circle), then configuring that state — or making a transition from the Ground state to your new state. In these instructions, we proceed as if you are making a transition before making additional states.

---

3. **Select the circle for the Ground state (the “from” state), use the middle mouse button to extend your selection to the circle for the new state (the “to” state), and then select the Transitions (center) icon.**
   A line appears between the two states with a small box containing “0” on the line. We call this box the “transition-count box”. At the same time, you receive the Transitions window, which is identical to the Transitions window invoked from the text-based display. See the description of that window in Section 12.5, “Transitions Window.” When you bring up the Transitions window from the graphics display, the names of the two states connected by the transition are displayed in the From and To fields.

You can also obtain the Transitions window by pressing right in the transition-count box in the newly made transition line and selecting Configure in the new transition's menu.

Note – When configuring states and transitions, do your work in indivisible pairs: create a state, configure that state; create a transition, configure that transition.

4. **To make additional states and transitions, repeat Step 1 through Step 3.**

5. **Invoke File➤Save As to save the new template.**

You can create any number of states for a given template. However, you can display a maximum of nine (including Ground) in the Request Designer's graphical display.

The graphical display has a message area at the bottom of the window that is analogous to the text-display message area, described in Section 12.3.1.3, "Text-display Message Area."

The numbers in boxes indicate the number of transitions that have been created between a pair of states.

12.9.2 Other Tasks in the Graphical Display

To delete a state, select Configure from the menu for the state icon to be deleted. In the Configure States window, select Delete, then select OK in the dialog box that subsequently appears.

To delete a transition, press right in the transition-count box for that transition. Select Configure. You receive the Transitions window. In this window, select Delete, then select OK in the dialog box that subsequently appears.

To delete the action for a transition (not the entire transition), press right in the transition-count box for that transition. Select Configure. You receive the Transitions window. In this window, select the action you want to delete, enter any arguments (such as Command or Address), and select Delete. If you specify an action of <none>, all actions are deleted.

To obtain a description of a state or transition, press right in the state icon or in the transition-count box for the transition. Select Description. You receive a read-only State or Transition Information window.

12.10 em_ncimport *and* em_ncexport *Utilities*

Templates, conditions, poll rates, and severities can be saved to an ASCII text file using the em_ncexport command-line utility. Templates, conditions, poll rates, and severities previously saved to an ASCII text file can be loaded into an MIS using the em_ncimport command-line utility. This facility is useful for:

- Replicating templates, or selected template components, from one MIS to another
- Storing several versions of the same template, or an entire Nerve Center template data

Syntax:

```
em_ncexport [-host <hostname>] [-file <filename>] [-help]
[-t <template_name>] [-c <condition_name>]
[-p <poll_rate_name>] [-s <severity_name>] [-minimize]

em_ncimport [-host <hostname>] [-file <filename>] [-t] [-
help]
[-p] [-s] [-c] [-v]
```

Options

`-help`
Displays help text.

`-host` <*hostname*>
<*hosthame*> is the name of machine with the target MIS for import or export. Default is `localhost`.

`-file` <*filename*>
<*filename*> is name of file to export to or import from. Default is `stdin` for import, `stdout` for export.

`-t` [<*template_name*>]
<*template_name*> is the name of the template to export. By default, all poll rates, severities, and conditions that are associated with the template are exported. Multiple template names can be listed if the list of names is surrounded by double quotes. If no name is specified, all templates are exported. For example:

```
em_ncexport -t “IsSnmpSystemUp PingUpOrDown RouterIfStatus”
```

Individual templates cannot be selected for import. The `-t` option imports all templates in the selected file.

`-c` [<*condition_name*>]
<*condition_name*> is the name of the condition to export. Multiple condition names can be listed if the list of names is surrounded by double quotes. If no name is specified, all conditions are exported.

Individual conditions cannot be selected for import. The `-c` option imports all conditions in the selected file.

`-p` [<*poll_rate_name*>]
<*poll_rate_name*> is the name of the poll rate to export. Multiple poll names can be listed if the list of names is surrounded by double quotes. If no name is specified, all polls are exported.

Individual polls cannot be selected for import. The `-p` option imports all poll rates from the selected file.

`-s` [<*severity_name*>]
<*severity_name*> is the name of the severity to export. Multiple severity names can be listed if the list of names is surrounded by double quotes. If no name is specified, all severities are exported.

Individual severities cannot be selected for import. The `-s` option imports all severities from the specified file.

`-v`

Turns on verbose mode. Warnings are printed in addition to errors. For example, if a condition is a duplicate of one already in the MIS, this generates a warning. This option is not supported for export.

`-minimize`

If specified, only the template is exported, and not the conditions, poll rates, or severities associated with it. This option is suppored only for `em_ncexport`.

Note – For `em_ncexport`, at least one of the following options must be specified: `-t`, `-c`, `-p`, `-s`. `em_ncimport` can be used without any options being specified. If no options are specified, `em_ncimport` imports all components in the specified file.

Examples

- Templates can also be piped to a printer. In the following example, the template `IfUP` from the remote MIS on machine `bar` is sent to the printer:

```
% em_ncexport -host bar -t IfUp -minimize | lp
```

- Export all conditions from the default MIS to file `myconditions`:

```
% em_ncexport -f myconditions -c
```

- Export template `IfUp` and all templates related to it (that is, referenced by it) to file `foo`:

```
% em_ncexport -f foo -t IfUp -r
```

- Import all contents of the file `mytemplates` to the default MIS:

```
% em_ncimport -f mytemplates -t -c -p -s
```

- Import only templates from the file `templatelib`:

```
% em_ncimport -f templatelib -t
```

- Export all conditions from the machine `bighost` to the machine `host2`:

```
% em_ncexport -host bighost -c | em_ncexport -host host2 -c
```

Request Condition Language 13

Types of Operands	*page 13-2*
Constants	*page 13-3*
Variables in a Condition	*page 13-3*
Data Types	*page 13-5*
System Variables	*page 13-6*
Message Types	*page 13-12*
Attributes	*page 13-13*
Operators	*page 13-15*
Control Structures	*page 13-18*
Timestamp Arithmetic	*page 13-23*
Error Checking	*page 13-24*

Request Condition Language (RCL) is a script language used to build *conditions*. Using RCL you can build up a library of conditions that can be deployed as building blocks in the construction of request templates in the Request Designer.

A *condition* is a sequence of one or more statements written in the Solstice EM RCL. With the exception of compound expressions built using IF, IF ELSE, FOR EACH, and WHILE constructs, each RCL statement must end with a semicolon.

13

There are two possible roles that a condition can play in a template:

- A single condition is used to define when a transition from one state to another in a template will occur.
- Conditions can also function as *actions* that are executed as the result of a transition. Multiple conditions can be specified as actions for the same transition and they will be executed in the order listed in the transition.

When a condition is used to define when a transition will occur, the sequence of statements must evaluate as true or false. That is, the last statement must return a result that can be treated as false (null) or true (not null). If the condition returns a value of true, this causes the transition to occur.

However, when a condition is used to define an action that occurs as the result of a transition, the condition's return value is ignored.

This chapter covers the operands, variables, attributes, and other components that make up RCL. RCL also provides a library of built-in functions that can be used in building templates. The built-in functions are described, in alphabetical order, in Chapter 14, "RCL Functions."

13.1 Types of Operands

The operators or built-in named operators that are currently implemented support primitive data types.

The RCL has three basic types of operands:

- Constants
- Variables
- Attributes

Each operand has a type and a value. The *type* is represented internally as an `Asn1Type`. For example, an integer operand has the type `INTEGER`. The *value* of an operand is represented internally as an `Asn1Value`. A built-in operator assigns types and values to variables dynamically. Type declarations are not

required for variables. For example, the variable `$counter` is initially defined as an integer type when the following assignment is encountered in a condition:

```
$counter = 0;
```

13.2 Constants

Table 13-1 shows the list of constants represented in the RCL syntax.

Table 13-1 Types of Request Condition Language Constants

Constant Type	**Values (examples)**
BOOLEAN	true false
INTEGER	10 0 57 0x3e 0X580
REAL	10.73
OCTET STRING	“Hello” “How are you?” “Embedded\\’Quote”
OBJECT IDENTIFIER	{1 2 3 1 }
GeneralizedTime	"19931106210627.3" (YYYYMMDDHHMMSS.S)

13.3 Variables in a Condition

Variables can be used to store temporary information. For example:

```
$last_sys_up_time = sysUpTime;
```

In this RCL statement, `$last_sys_up_time` is a user variable to store the value obtained from the SNMP `sysUpTime` attribute.

Variables have the scope of the request template. Every request that uses a template thereby has all the variables named in the template. Each request has its own storage area, the `StackFrame`. All variables are assigned to storage

locations in the `StackFrame`. Thus, when a variable is defined once in the request template, instances of it are created in each request's `StackFrame`. Values are retained throughout the life of the request.

A condition expression can make use of two classes of variables:

- **System Variables** — These are always present in each request. Their values are set by the Solstice EM MIS. System variables are listed in Table 13-2.
- **User-defined Variables** — These comprise all user-deployed variables other than system variables. They **cannot** duplicate the name of a system variable. They are not declared. Whenever a variable is assigned, it is automatically assigned a type corresponding to the result of the expression assigned to it. If a variable has never been assigned, it is said to be undefined. There is a function, `defined()`, (described in Section 14.9, "Defined") to test whether a variable has been defined.

13.3.1 Variable Names

The name of a variable begins with $ (dollar sign) followed by one or more alphanumeric characters or _ (underbar).

Note – Case is significant in variable names.

13.3.2 Scope of Variables

Each request that implements a request template has a complete set of the variables defined in all of the conditions used anywhere in that template. The values of those variables are local to the request. That is, in each request, variables have values that are independent of their values in any other request. The values of variables within a request persist as long as the request lives.

The scope of a variable name is the template in which it occurs. That is, a variable that is set by one condition within a template can be used by any other condition in the same template. However, variables are defined when a condition in which it occurs is evaluated. A variable defined in a condition that has not yet been evaluated in a request, is not available for conditions that occur earlier in that request template.

13.4 Data Types

Because the Request Designer functions in a CMIP and an SNMP framework, `Asn1Values` and `Asn1Types` are used throughout the RCL. Any variable, attribute, or constant carries the `Asn1Type` along with it. For example, in the expression:

```
$int_val = 10;
```

`$int_val` is a variable and is assigned the value 10. It is automatically assigned the `Asn1Type INTEGER`. Also, attributes have type information associated with them.

Because typing is dynamic, a variable's type is defined as the currently assigned type. Thus `$int_val` can be changed to type `REAL` as follows:

```
$int_val = 10.0;
```

Variables can be assigned arbitrary `Asn1Values` of arbitrary `Asn1Type`. Thus it is possible to add new operators to the language that deal with any type other than those listed in Table 13-1 with which most operators deal. An example of such an operator is `TrapSpecificType`, which takes as input an operand of type `InternetActionInfo`.

13.5 System Variables

The names and types of the available system variables are shown in Table 13-2.

Table 13-2 System Variables Available to a Condition

Name	Type	Description
$eventOC	OID	Object class of last/current event
$eventOI	ObjectInstance	Object instance of event
$eventInfo	eventInfo	eventInfo of eventType
$eventTime	GeneralizedTime	Actual time the event was generated
$eventType	OID	OID of the event type of the last event
$messType	INTEGER	Type of current message (refer to Table 13-4 for `$messType` values)
$multipleInstance	BOOLEAN	For SNMP polls only. True if the polled object has multiple instances, False otherwise.
$pollOC	ObjectClass	Object class last polled or being currently polled
$pollfdn	ObjectInstance	Object instance being polled
$pollFdnSet	SET OF ObjectInsance	Set of distinguished names pointing to the managed object instances configured for the target device. Assigned when request launched against a selected element in the Viewer.
$pollTime	GeneralizedTime	Delay until the first poll is sent and the time between successive polls, in seconds
$severity	INTEGER	Severity level of current state

The following paragraphs contain examples illustrating the use of system variables.

13.5.1 $pollFdnSet

When a request is launched against a device selected in the Viewer, $pollFdnSet is assigned a set of fully distinguished names (FDNs) which denote the managed objects that have been configured for the device (for example, when Discover is run to populate the MIS). A “managed object” is

the internal representation in the MIS of the agent — for example, a cmipsnmpProxyAgent object, which represents an SNMP agent system. The order of the FDNs in `$pollFdnSet` depends upon the order in which they were added to the MIS. Individual FDNs can be extracted from $pollFdnSet using the RCL `extract()` function. The following is an example of a condition that extracts the RPC proxy table FDN from $pollFdnSet in order to set the $pollfdn to `ping-reach` — the `reach` attribute group of the RPC ping agent.

```
$num = NumElements(&$pollFdnSet);
$count = 1;
WHILE ($count <= $num)
{
   $numstr = AsnToStr($count,TRUE);
   $dn = Extract(&$pollFdnSet,$numstr);
   $dn1 = Extract(&$dn,”distinguishedName”);
   $dnstr = AsnToStr($dn1,TRUE);
   $result = AnyStr($dnStr,”RPC”);
   IF ($result == TRUE)
   {
      $dn2 = Extract(&$dn1,”3”);
      $dn3 = Extract(&$dn2,”1”);
      $Hostname = Extract(&$dn3,”attributeValue”);
      $count = $num + 1;
   }
   $count = count+1;
}
$pollfdn = appendRdn($dn,”/agentId=\”ping-reach\””);
```

13.5.2 $pollfdn

The `$pollfdn` variable represents the object that is the target of the request. In the following example, `$pollfdn` is used to pass the request's target managed object instance to the `subscribeOi()` function to subscribe for SNMP event notifications generated by that object.

```
$itindx=subscribeOi(“internetAlarm”,””,$pollfdn);
```

The $pollfdn is based on the object's Fully Distinguished Name — the absolute path to the object through the Management Information Tree (MIT). When a request is launched in the Viewer, the $pollfdn is initially set to the first managed object in $pollFdnSet. In the following example, a request launched against the router bigguy has its $pollfdn set to the cmipsnmpProxyAgent:

```
/systemId=name:"gatoloco"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2
4]/cmipsnmpProxyAgentId="bigguy"
```

However, this $pollfdn could be changed to point to particular MIBs "contained" under the SNMP agent. In the following example, the appendRdn() function is used to change the $pollfdn to point to the snmp-mibII object:.

```
$tmp = "/InternetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}";
$pollfdn = appendRdn($pollfdn,$tmp);
```

After the append operation, the $pollfdn is the following:

```
/systemId=name:"gatoloco"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2
4]/cmipsnmpProxyAgentId="bigguy"/InternetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}
```

Note – If a condition resets $pollfdn to point to a different managed object, the change does not take effect until after a transition to a different state. Therefore, if a condition resets $pollfdn to point to a different object, your design the request to transition to a different state before trying to poll the new object.

Chapter 10, "Building Request Templates," describes sample templates that change the target of polling during execution.

13.5.3 $eventOI

`$eventOI` indicates the managed object instance that was the source of an event notification that "woke up" the request. In the following example, the `OiToOiName()` function is used to convert the `$eventOI` to a string.

```
$name = OiToOiName($eventOI);
```

13.5.4 $eventInfo

`$eventInfo` is the current event notification. This will be a sequence of ASN.1 values of the attributes comprising the event. The attributes that comprise `$eventInfo` depend upon its event type (the value of `$eventType`). The definition of the event type specifies the *required* attributes for that type and optional attributes, if any. The required attributes for a communicationsAlarm, for example, are probableCause and perceivedSeverity. If the attribute has an assigned name, this tag can be used to extract the ASN.1 value of that attribute from $eventInfo, as in the following example:

```
$cause = Extract(&$eventInfo,"probableCause");
```

The attributes that comprise a given event notification depend upon the definition of its event type. The definitions used by the MIS are contained in the pertinent ASN.1 and GDMO documents. The event types known to the MIS by default are described in Chapter 15, "Adding New Event Types."

In the following example, the value of `$eventInfo` is set to contain a nerveCenterAlarm:

```
$eventInfo = strToAsn("EM-NC-
ASN1:NerveCenterAlarmInfo","{1,critical,\"Device Down\",3,1}";
```

The first argument passed to `strToAsn1()` is text that refers to the event type — NerveCenterAlarmInfo. This type is defined in the ASN.1 document `/opt/SUNWconn/em/etc/asn1/nc.asni`. The definition specifies that an event of type NerveCenterAlarmInfo is a sequence of ASN.1 attributes. A nerveCenterAlarm is defined as including four required attributes —

probableCause, perceivedSeverity, mosiSeverity, mosiStateID — and an optional fourth attribute, additionalText. The permissible values, and standard interpretation, of perceivedSeverity values is indicated in Table 13-3.

Table 13-3 perceivedSeverity Values

Severity Name	Value	Default Color
Indeterminate	0	Blue
Critical	1	Red
Major	2	Orange
Minor	3	Cyan
Warning	4	Yellow
Clear	5	No color

Thus, in the example above, `$eventInfo` is defined as a `nerveCenterAlarm` with a `perceivedSeverity` of critical, a `mosiSeverity` value of 3, a probableCause value of 1, and a `mosiStateID` value of 1. The `strToAsn()` function is used to convert the string constant to the sequence of ASN.1 values required by the event type definition.

13.5.5 *$eventType*

`$eventType` is the Object Identifier for the type of the event that "woke up" the request. The following example shows a condition used to define a transition from one state to another. The transition will take place if communicationsAlarm is the event type.

```
$comm = NameToOid("communicationsAlarm");
$eventType == $comm;
```

In the next example, an `IF` statement tests whether an event is an internetAlarm and `sendEvent()` is called to post the event to the alarm log if it is.

```
$itType=NameToOid(“internetAlarm”);
IF ($eventType == $itType)
{sendEvent(“internetClass”,$pollfdn,”internetAlarm”,$eventInfo);}
```

The event types known to the MIS by default are the following:

- objectCreation
- objectDeletion
- attributeValueChange
- relationshipChange
- stateChange
- communicationsAlarm
- environmentalAlarm
- equipmentAlarm
- integrityViolation
- operationalViolation
- physicalViolation
- processingErrorAlarm
- qualityofServiceAlarm
- securityServiceOrMechanismViolation
- timeDomainViolation
- internetAlarm
- snmAlarmEvent
- snmAlarmTrap
- nerveCenterAlarm
- coldStartTrap
- warmStartTrap
- linkDownTrap
- linkUpTrap
- linkDownTrap
- egpNeighborLossTrap
- authenticationFailureTrap
- enterpriseSpecificTrap

For more information on these event types refer to the ”Standard Event Notifications” appendix in the *Solstice Enterprise Manager Reference Manual.*

13.6 Message Types

When an event or poll response is received, the variable `$messType` is set in the request. A condition can check the value of `$messType`. In a state machine that is both poll- and event-based, `$messType` will indicate the current message received, either an `EVENT_REPORT_REQ`, `GET_RES`, or errors. As an example of the use of `$messType`, the following is a condition that defines a transition. The probableCause value is extracted from an equipmentAlarm and a transition occurs if an event notification has been received ($messType == 0) and it has a probableCause value indicating equipmentMalFunction.

```
$cause = Extract($eventInfo,"probableCause");
$messType == 0 AND $cause == 15;
```

A CMIP event notification has a `$messType` value of 0 (EVENT_REPORT_REQ) because it is generated by the agent on its own initiative; it is not a response to a request generated by a management station.

Another use of $messType is to check for errors in the request. If errors occur during polls, the state machine can be designed to transition to a "dead" state.

The possible values of $messType are specified in Table 13-4.

Table 13-4 Values of $messType

Message	No.	Description
EVENT_REPORT_REQ	0	Request
EVENT_REPORT_RES	7	Response
GET_RES	8	Get Response obtained
SET_RES	9	Set Response obtained
ACTION_RESPONSE	10	A response to an M-ACTION has been received.
NO_SUCH_OC	14	Object class being polled does not exist
NO_SUCH_OI	15	Object Instance being polled does not exist
ACCESS_DENIED	16	Operation not performed due to security problem
SYNC_NOT_SUPP	17	Synchronization not supported
INVALID_FILTER	18	Filter parameter invalid
GET_LIST_ERR	21	One or more attribute values not read because access was denied or attribute was not recognized
PROCESS_FAILURE	24	General failure in processing
INVALID_SCOPE	32	Value of scope parameter invalid
INVALID_OI	33	Invalid OI specified
CLASS_INST_CONFL	35	Specified OI not of specified class
COMPLEX_LIMIT	36	Operation not performed due to complex parameter supplied
MISTYPED_OP	37	One of the parameters supplied has not been agreed for use on association
INVALID_OPERATION	38	Invalid operation requested
OP_CANCELLED	41	Operation cancelled by M-Cancel-Get

13.7 Attributes

In addition to constants, variables, operators, and built-in functions, an expression in the RCL can refer to an *attribute* of a managed object. An attribute has both a name (determined in the GDMO description of the object in which it occurs) and a value. When an attribute is used in a condition, it is

assigned storage in the same way that storage is allocated for a variable. The scope of an attribute is the request template in which it occurs. Each request has its own copy of the attribute, independent of any other request.

Each attribute has a *type*. The type is determined by the object's description, as recorded in the MetaData Repository (MDR). When a condition refers to an attribute, the Nerve Center queries the MDR and assigns the attribute's type accordingly.

An attribute within a request reflects the current condition of some attribute of a managed object. To obtain current information, the Nerve Center schedules polls for all attributes that are referred to in the conditions that must be tested for the current state of each request. The Nerve Center also schedules polls for the attributes referred to in the actions for transitions leading from the current state. That is, for every request, the Nerve Center tracks its current state, and for that state, schedules a poll for every attribute that must be tested to determine whether there will be a transition from that state. When the response to a poll arrives, the attribute's value in the request is updated before the conditions are evaluated.

The value of an attribute within a request is set only by the mechanism just described; you *cannot* use the = operator to assign a value to an attribute. Therefore an attribute name cannot appear on the left side of an assignment.

Like a variable, an attribute is "declared" automatically when used in a condition. It becomes "defined" when a notification assigns a value to it.

13.7.1 Syntax of Attribute Names

The name of an attribute can be written in either of two forms:

&<*label_name*>	<*label_name*> is the name as it occurs in the relevant GDMO description of a managed object class. & is to be used if an attribute or variable is passed as an argument in such functions as extract(), define(), and undefine(). The ampersand is used to pass the address of the variable or attribute in the Stack Frame.
"<*doc_name*>":<*label_name*>	A label <*label_name*> may be preceded by the name of the GDMO document in which the label occurs. The document name is enclosed in double quotes. The document name and the label name are separated by a colon.

For example, the attribute `sysContact` specified in the document IIMCRFC1213-MIB, is written

```
"IIMCRFC1213-MIB":sysContact
```

13.8 Operators

The RCL uses the same operators as C (for example, = for assignment, == for a test of equality, * for multiply, and so on). In addition, there are built-in named operators whose syntax resembles the syntax of functions in the C programming language.

Operators are arithmetical, logical, and relational. Each operator or built-in named operator specifies the input argument and types that it can handle.

The following operator symbols are supported by the RCL.

Operation Type	Operator	Description
Assignment	=	Assigns the value or attribute to the right of the operator to the name to the left
Arithmetic	+ – - * / % | ^ ~	plus minus (preceded and followed by blanks) negative (no blanks following) multiply divide modulus Bitwise inclusive OR Bitwise exclusive OR Bitwise NOT
Relational	< <= > >=	less than less than or equal greater than greater than or equal
Equality	== !=	equal not equal

Operation Type	Operator	Description
Logical	AND OR NOT	and or not
Address	&	"Address of" is used in the argument of defined or undefine to permit inquiry about the status of a variable name without referring to its value. Addresses are basically indices into the StackFrame — the location where the variable or attribute is stored.

The assignment operator can be used to assign values or types to variables.

Note – You cannot use = to assign or set values of attributes.

The arithmetic operators are defined for INTEGERS and REAL data types. Modulus is defined only for INTEGERS.

Note – Because the hyphen within an attribute name could be confused with a minus sign, a minus sign *must* be surrounded by blanks. For example:

```
10 - 5;    This is correct
10-5;      This is a syntax error
```

- There are no implicit type conversions. That is, 5/2 yields an integer result, so if you expect the result to be 2.5, you need 5.0/2.0 (as in C).
- The relational and equality operators accept not only integer or real arguments, but can also be used to compare arbitrary Asn1Values (in the same way as the CMIS Filter constructs).
- Statements built up using the logical operators are completely evaluated. That is, each operand of a complex expression, built using the AND, OR, and NOT operators, is evaluated. Thus, "short circuiting" is not implemented, that is, evaluation of the component expressions does not stop even if the value of the complex expression is already known from the evaluation of initial components. Logical operators operate only on Boolean values, Integers, and Reals. The names of the logical operators are *not* case sensitive.

13.8.1 Logical Operators

The following example illustrates the use of an `OR` statement to define a condition for a transition. First, the user variables `$ncType` and `$itType` are defined as nerveCenterAlarm and internetAlarm (respectively) in the following condition, which also subscribes to receive SNMP traps. This condition might be used to initialize the template in the Ground state.

```
$ncindx=Subscribe("nerveCenterAlarm");
$itindx=subscribeOi("internetAlarm","",$pollfdn);
$ncType=NameToOid("nerveCenterAlarm");
$itType=NameToOid("internetAlarm"); true;
```

In the following example, the `OR` operator is used in a condition that forces a transition if the system variable `$eventType` indicates that either an `internetAlarm` or a `nerveCenterAlarm` has been received:

```
$eventType == $ncType OR $eventType == $itType;
```

13.8.2 Bitwise Operators

The bitwise operators numeric AND, inclusive OR, and exclusive OR perform binary operations on numeric operands and generate numeric results. For example,

```
20|24
```

compares the binary numbers

```
20 = 00010100
24 = 00011000
```

and generates a binary number with a 1 bit wherever either (or both) of the operands has a 1 bit. The resulting value is:

```
28 = 00011100
```

13.8.3 Precedence and Associativity

The precedence and associativity of operators are summarized in Table 13-5.

Table 13-5 Precedence of Operators

Operator	Name	Associativity
High		
=	Assign	right
OR	Or	left
AND	And	left
| ^ &	Bitwise numeric OR Bitwise numeric XOR Bitwise numeric AND	left
= !=	Equal Not equal	**left**
< <= > >=	Less than Less than or equal Greater than Greater than or equal	left
+ -	Plus Minus	left
* / %	Multiply Divide Modulus	left
&	Address	right
~ - NOT	Bitwise NOT (numeric) Negative (numeric) Negation (logical)	right
Low		

Parentheses force precedence in the usual way.

13.9 Control Structures

RCL supports four constructs that can be used to build control structures within a condition: `IF`, `IF ELSE`, `WHILE`, and `FOREACH`. These four constructs are used to control the conditions under which a block of RCL statements are

to be executed. An RCL statement block consists of zero or more RCL statements, each terminated with a semicolon. Also, a statement block must be preceded by a left curly brace and followed by a right curly brace.

13.9.1 IF Constructs

Syntax:
IF (<*boolean_expression*>)
{<*statement_block*>}

<*boolean expression*> must be an RCL expression that evaluates as either true or false. <*statement_block*> consists of zero or more RCL statements, each terminated with a semicolon. The block of statements must be surrounded by curly braces, as shown above. The RCL statements contained in <*statement_block*> are executed if <*boolean_expression*> evaluates to true. For example:

```
IF ($eventOi = $pingFdn)
{$ping_response_count = ping_response_count+1;}
```

13.9.2 IF ELSE Constructs

Syntax:
IF (<*boolean_expression*>)
{<*statement_block1*>}
ELSE
{<*statement_block2*>}

<*boolean expression*> must be an RCL expression that evaluates as either true or false. <*statement_block1*> and <*statement_block2*> each consists of zero or more RCL statements, each terminated with a semicolon. Each block of statements must be surrounded by curly braces, as shown above. The RCL statements contained in <*statement_block1*> are executed if and only if <*boolean_expression*> evaluates to true.

The block of statements comprised in <*statement_block2*> — the ELSE construct — are executed if and only if <*boolean_expression*> in the preceding IF statement evaluated to *false*. For example:

```
$FdnStr = AsnToStr($dn,TRUE);
$result = AnyStr($FdnStr,"RPC");
IF ($result == TRUE)
{
  print($FdnStr);
  $count = $num + 1;
}
ELSE
{
$count = $count+1;
}
```

13.9.3 WHILE Constructs

Syntax:
WHILE (<*boolean_expression*>)
{<*statement_block*>}

<*boolean_expression*> must be an RCL expression that evaluates to either true or false. <*statement_block*> consists of zero or more RCL statements, each terminated with a semicolon. The statements comprised in <*statement_block*> are executed if <*boolean_expression*> evaluates to true. After the statements in <*statement_block*> have been executed, <*boolean_expression*> is evaluated once again. So long as <*boolean_expression*> remains true, the statements in <*statement_block*> continue to be executed in a repetitive cycle.

The following is an example of a condition that uses a WHILE loop to extract the RPC proxy table FDN from $pollFdnSet in order to set the $pollfdn to `ping-reach` — the `reach` attribute group of the RPC ping agent.

```
$num = NumElements(&$pollFdnSet);
$count = 1;
WHILE ($count <= $num)
{
   $numstr = AsnToStr($count,TRUE);
   $dn = Extract(&$pollFdnSet,$numstr);
   $dn1 = Extract(&$dn,"distinguishedName");
   $dnstr = AsnToStr($dn1,TRUE);
   $result = AnyStr($dnStr,"RPC");
   IF ($result == TRUE)
   {
      $dn2 = Extract(&$dn1,"3");
      $dn3 = Extract(&$dn2,"1");
      $Hostname = Extract(&$dn3,"attributeValue");
      $count = $num + 1;
   }
   $count = count+1;
}
$pollfdn = appendRdn($dn,"/agentId=\"ping-reach\"");
```

13.9.4 FOREACH Constructs

Syntax:
`Foreach name in (`<*list_expression*>`)`
`{`<*statement_block*>`}`

<*list_expression*> must be of type `SEQUENCE OF` or `SET OF`. The block of statements comprised in <*statement_block*> is executed once for each element of the set or sequence, using `name` as a variable to represent the current element in each cycle. The variable `name` is automatically assigned the appropriate type for each element that it represents. If <*list_expression*> is not of type SEQUENCE OF or SET OF, <*statement_block*> is executed exactly once with

`name` assigned the entire value of <*list_expression*>. The end of the block of statements in <*statement_block*> is marked by the final curly brace. For example:

```
foreach $var in (collectionInfoList)
{
   print($var);
}
```

The RCL `FOREACH` construct is similar to the UNIX Shell `Foreach` construct.

13.9.5 Nested Constructs

A statement block in an `IF`, `IF ELSE`, `WHILE`, or `FOREACH` construct can contain additional constructs. For example, an `ELSE` construct could contain another `IF ELSE` construct, such as the following:

```
IF (<boolean_expr1>) {<RCL_statement1>}
ELSE
{IF (<boolean_expr2>) {<RCL_statement2>} ELSE {<RCL_statement3>}}
```

Similarly, a `WHILE` or `FOREACH` construct might contain an `IF ELSE` construct within its statement block.

The following is an example of an IF ELSE construct used to log nerveCenterAlarms in response to enterprise-specific traps:

```
$snum = TrapSpecificType($eventInfo);
$pollfdn = $eventOi;
IF ($snum == 1)
{
   $tmp = “CPU Failure”;
   alarmStr(1,$tmp);
}
ELSE
{
   IF ($snum == 2)
   {
      $tmp = “Fan Failure”;
      alarmStr(1,$tmp);
   }
   ELSE
   {
      IF ($snum == 3)
      {
          $tmp = “Power Supply Failure”;
           alarmStr(1,$tmp);
      }
      ELSE
      {
         IF ($snum == 4)
         {
            $tmp = “Excessive Temperature”;
            alarmStr(3,$tmp);
         }
      }
    }
}
```

13.10 Timestamp Arithmetic

Timestamps are of the type `GeneralizedTime`. The system variables `$eventTime` and `$pollTime` are of that type.

The following operators can accept Timestamp arguments or return Timestamp results

- <*Timestamp1*> = <*Timestamp2*> + <*integer*>
- <*Timestamp1*> = <*Timestamp2*> + <*real*>
- <*real*> = <*Timestamp1*> - <*Timestamp2*> (result in milliseconds)
- <*Timestamp1*> = <*Timestamp2*> - <*integer*>
- <*boolean*> = <*Timestamp1*> > <*Timestamp2*>
- <*boolean*> = <*Timestamp1*> >= <*Timestamp2*>
- <*boolean*> = <*Timestamp1*> <= <*Timestamp2*>
- <*boolean*> = <*Timestamp1*> < <*Timestamp2*>
- <*boolean*> = <*Timestamp1*> == <*Timestamp2*>
- <*boolean*> = <*Timestamp1*> != <*Timestamp2*>

In the following example, a difference greater than six seconds between the current system time on the MIS machine and the time when an event was generated on a remote machine is used to define a condition for a transition.

```
$curtime = getTimeStamp();
($eventTime - $curtime) > 600;
```

In this case the transition would occur if the statement evaluates to True.

13.11 Error Checking

The Request Designer application checks and catches lexical and syntactic errors at the time you try to save the condition's definition. If it finds an error, the Request Designer displays an error dialog, and the offending condition is not saved. If there are any references to attributes in conditions, the Request Designer checks to determine if the attribute is known to the MIS. An attribute is not known to the MIS if it is not referred to in a GDMO document that has been loaded into the MIS. If an attribute is not known to the MIS, Request Designer displays an error dialog if you try to save the condition, and the condition is not saved.

If the condition survives the lexical and syntactic check, the Request Designer engine compiles the condition's definition. No further checking occurs at compile time. Only runtime type checking is implemented.

When a condition is executed within a particular request, each function or operator checks the type of each argument it receives. (The RCL does not provide type casting, so you cannot coerce types.) If the type is invalid, the

operator returns an error. For example, if a built-in function expects an OCTET STRING and it is passed an INTEGER, it causes the condition to return FALSE.

Warning – A condition that is syntactically valid but contains an error detected only at runtime acts in the same way as a valid condition that returns FALSE.

The em_debug utility provides facilities for debugging request templates. For information on template debugging, refer to Chapter 10, “Building Request Templates.”

13

RCL Functions 14

The Request Condition Language (RCL) built-in functions are described in this chapter in alphabetical order.

Built-in functions take the form of a function call with a set of arguments. `List` denotes an Asn1Value whose type is SET, SEQUENCE, SET OF, or SEQUENCE OF. <*Var*> indicates a variable name. <*Attr*> indicates an attribute name.

The names of RCL functions are not case-sensitive.

14.1 Summary of RCL Built-in Functions.

Alarm Log Functions

- `Alarm` — Generates a nerveCenterAlarm with indicated severity.
- `AlarmOi` — Takes severity, object instance, and event arguments.
- `AlarmStr` — Generates a nerveCenterAlarm with indicated severity and additionalText.
- `SendEvent` — Logs an event notification. Allows you to log event notifications of types other than `nerveCenterAlarm`.

String-Handling Functions

- `InitialStr` — True if string matches initial portion of a string.
- `FinalStr` — True if string matches end portion of a string.
- `AnyStr` — True if string appears anywhere in a string.
- `StrCat` — Builds string by concatenation.

Value Check Functions

- `Defined` — True if variable or attribute has a value.
- `Undefine` — Sets variable or attribute to have no value.

Name Conversion Functions

- `NameToOid` — Returns an Object IDentifier from a name string.
- `OiNameToOi` — Object instance from quoted name string.
- `OiToOiName` — Returns the name of an object instance.
- `AddressStrToAddress` — Dot address string to Internet address string.
- `NameToAddress` — IP address from a host name string.
- `AppendRdn` — Constructs an object instance from an object instance and RDN string.

Action Functions

- `Unixcmd` — Executes a specified UNIX command.
- `Mail` — Sends an e-mail message.

ASN.1 Conversion Functions

- `AsnToStr` — Builds string representation of an ASN.1 value.
- `StrToAsn` — Builds ASN.1 values from strings.

SunNet Manager RPC Request Functions

- `SnmEventRequest` — Issues request to an SNM RPC agent.
- `SnmKillRequest` — Kills a previously issued SNM request.

Debugging Function

- Print — Prints values if `em_debug misc_stdout` option turned on.

Constructed-Type Handling Functions

Parameters may be passed to constructed-type handling functions. Each parameter is an expression which in turn may contain function calls.

- `Extract` — Returns value of a component in a list.
- `Include` — Used to construct an ASN.1 value of types SET or SEQUENCE.
- `IsList` — True if variable or attribute is a list.
- `NumElements` — Returns number of elements in a list.
- `IsChoice` — True if a variable or attribute is a choice.

Time Functions

- `GetTimeStamp` — Retrieves the current time of the MIS machine.

Event-Handling Functions

- `Set` — Performs an M-SET on a specified object.
- `Subscribe` — Subscribes to event type specified in string.
- `SubscribeOi` — Subscribe to events for specified object.
- `SubscribeFilter` — Subscribes for events that match a specified CMIS filter.
- `TrapSpecificType` — Returns number of `SpecificType` of a trap.
- `TrapGenericType` — Returns number of `GenericType` of a trap.
- `SendAction` — Sends an M-ACTION to a specified object.
- `SendTrap` — Sends a trap to destination IP address.
- `UnSubscribe` — Can be used to terminate a previously invoked event subscription.

14.2 AddressStrToAddress

Syntax:
`AdressStrToAdress(<addrStr>);`

where <*addrStr*> is of type `OCTET STRING.`

Return Value: `unsigned long.` Returns the value of the `inet_addr()` system call.

Takes a string containing an address in dot format and returns an unsigned long integer containing an internet address. For example:.

```
$saddr=AddressStrToAddress(“129.144.44.36”);
```

14.3 Alarm

Syntax:
`alarm(<perceivedSeverity>);`

where <*perceivedSeverity*> is of type `INTEGER` (in the range 0–5).

Return Value: None.

The MIS Alarm Service (which monitors the alarm logs) causes the Viewer to change the color of the icon for the object instance associated with the alarm (the object indicated by the `$pollfdn` system variable), based on the value of <*perceivedSeverity*>. The value must be in the range 0–5. For example, the `AlarmCritical` sample condition provided with Solstice EM contains the following:.

```
alarm(1);
```

This statement will post a `nerveCenterAlarm` with severity `critical`. The valid severities and their associated icon colors are listed in Table 14-1.

Table 14-1 Valid Alarm Severities

Severity Value	Severity Name	Default Icon Status Color
0	Indeterminate	Blue
1	Critical	Red
2	Major	Orange
3	Minor	Cyan
4	Warning	Yellow
5	Cleared	No color

14.3.1 Alarm Logging and Viewer Fault Status

The `alarm()` function allows you to generate a nerveCenterAlarm which is, by default, logged to the AlarmLog. Alarms logged to the alarm log can be viewed and cleared in the Alarm Manager.

The AlarmLog is also, by default, monitored by the Alarm Service. The Alarm Service is a module in the MIS that controls the fault status color in the Viewer. Fault status is an attribute of topology nodes, which are represented by icons in the Viewer. Each topology node has an attribute `topoNodeMOSet`, which points to a set of managed object instances (MOIs), representing the agents configured for the particular device.

The Alarm Service associates an alarm posted to the AlarmLog with a topology node if and only if that alarm is posted against one of the managed objects in the `topoNodeMOSet` for that topology node. The Alarm Service tracks the `perceivedSeverity` values of the alarms that are posted against each topology node. The highest `perceivedSeverity` value of uncleared alarms determines the fault status of the device. Thus, if a critical alarm is logged aganst router `bigguy`, the router icon, by default, turns red. If several minor alarms are then posted against `bigguy`, these do not cause the router icon to turn cyan unless the critical alarm has been cleared. Once the critical alarm is cleared, the presence of uncleared minor alarms causes a change in color to cyan.

When a request is launched at a target device in the Viewer, the `$pollFdnSet` RCL system variable for that request points to the managed objects that are comprised in the topoNodeMOSet for the selected topology node. The `$pollfdn` system variable is also initially set to point to the first managed object listed in $pollFdnSet.

The `alarm()` function posts a nerveCenterAlarm against the managed object that the `$pollfdn` variable points to at the time when the `alarm()` function is called. If you have reset the `$pollfdn` variable to point to an object other than one of those comprised in `$pollFdnSet` in your request, you should either reset `$pollfdn` to an appropriate managed object before calling `alarm()` or else use the `alarmOi()` function, which enables you to specify the managed object against which the alarm is to be posted.

The `alarm()` and `alarmStr()` functions post nerveCenterAlarms that have a `probableCause` value equal to the `perceivedSeverity` value. For example, if your request uses `alarm()` to post a minor alarm, `probableCause` is set to 3. Alarm Service uses the `probableCause` value of nerveCenterAlarms to match a “clear” alarm to the previous nerveCenterAlarm it is clearing. For example, if your request has used

```
alarm(1);
```

to post a critical alarm, your request must post a nerveCenterAlarm with a `probableCause` of 1 and a `perceivedSeverity` of 5 (clear) to clear this alarm. Because `alarm()` and `alarmStr()` set `probableCause` to equal

perceivedSeverity, requests cannot use alarm() or alarmStr() to clear a previous nerveCenterAlarm. To post an alarm that clears a previous nerveCenterAlarm, your request must use the alarmOi() function.

14.4 AlarmOi

Syntax:
alarm(<*oi*>, <*perceivedSeverity*> | <*event-notification*>);

where <oi> is of type Object Instance. The second argument is either <*perceivedSeverity*> or <*event-notification*>. <*perceivedSeverity*> is of type INTEGER (in the range 0–5).

Return Value: None.

The Viewer icon will change color based on the severity value passed in <*perceivedSeverity*> (as indicated in Table 14-1). For example, the statement

```
alarmOi($pollfdn, 1);
```

will cause the icon representing the target of the current request to turn red.

Note – For an alarm posted against $pollfdn to cause a change in icon color in the Viewer, $pollfdn must point to one of the managed objects configured for the device. The $pollFdnSet variable is initially set to point to these objects when the request is launched against a selected device in the Viewer, and $pollfdn is initially set to point to the first object in $pollFdnSet. If your request template changes the value of $pollfdn to point to an object other than those in $pollFdnSet, the alarm may not affect icon color. For the <*oi*> parameter, you must use a variable that points to one of the objects in $pollFdnset if the alarm is to affect the Viewer fault status of the target device. For more information, see Section 14.3.1, “Alarm Logging and Viewer Fault Status” above or the “Alarm Service” chapter in the *Solstice Enterprise Manager Reference Manual*.

If <*event-notification*> is passed as an argument to `alarmOi()`, the event notification attribute values are used to build a `nerveCenterAlarm`. For example:

```
$event=strToAsn("EM-NC-ASN1.NerveCenterAlarmInfo", "{3,minor, 3, 1}";
alarmOi($pollfdn, $event);
```

If the system variable `$eventInfo` is passed as the second argument, `$eventInfo` will have been defined only if the request has subscribed for events. In that case, `$eventInfo` is set to the current event notification.

The `alarmOi()` function can be used to clear previous nerveCenterAlarms posted by a request. To post a "clear" alarm, `alarmOi()` must set the `probableCause` value to equal the `probableCause` of the nerveCenterAlarm it is clearing. The `alarm()` and `alarmStr()` functions automatically set the `probableCause` value of a nerveCenterAlarm to equal the `perceivedSeverity` value. Thus, to use `alarmOi()` to clear a previous critical alarm, you could use the following::

```
$event=strToAsn("EM-NC-ASN1.NerveCenterAlarmInfo", "{1,cleared, 3,
1}";
alarmOi($pollfdn, $event);
```

In this example, `probableCause` is set to 1 to match the `probableCause` of the previous critical alarm.

14.5 AlarmStr

Syntax:
`alarmStr(`<*perceivedSeverity*>`,` <*additionalText*>`);`

where <*perceivedSeverity*> is of type `INTEGER` (in the range 0–5) and <*additionalText*> is an RCL variable of any type.

Return Value: None.

The Viewer icon will change color based on the severity value passed in <*perceivedSeverity*> (as indicated in Table 14-1).

When this function is called, a nerveCenterAlarm will be generated with severity set to <*perceivedSeverity*> and `objectInstance` set to `$pollfdn`. The <*additionalText*> string will be passed as the `additionalText` attribute, which can be viewed in the Alarm Manager application. For example:

```
alarmStr(1,"Over 80% of network memory capacity in use");
```

In the following example the <*additionalText*> argument is used to pass the FDN of the managed object in an alarm with a severity of critical:

```
alarmStr(1,$pollfdn);
```

Note – An alarm posted using the `alarmStr()` function uses the value of `$pollfdn` to determine the managed object that is the target of the alarm. If the alarm is to affect Viewer icon color, `$pollfdn` must point to one of the managed objects that have been configured for that device. The `$pollFdnSet` variable is initially set to point to the managed objects configured for a device when the request is launched against a target device in the Viewer. `$pollfdn` is initially set to refer to the first of these objects. If your template resets the value of `$pollfdn`, you will need to either reset it to point to one of the objects in `$pollFdnSet` before calling `alarmStr()` or use the `alarmOi()` function, which allows you to specify the managed object that is the target of the alarm. Refer to Section 14.3.1, "Alarm Logging and Viewer Fault Status."

The `alarmStr()` function cannot be used to post nerveCenterAlarms that clear previous nerveCenterAlarms. For an explanation of how to clear previous alarms, refer to the entries for the `alarm()` and `alarmOi()` functions.

14.6 AnyStr

Syntax:
`anystr( <firststr>, <secondstr>);`
where <*firststr*> and <*secondstr*> are of type `OCTET STRING`.

Return Value: `BOOLEAN`

Checks whether <secondstr> appears anywhere in <*firststr*>. (A string constant is enclosed in double quotes.) In the following example $anywhere is a Boolean variable that will be assigned the value true if "Agent" occurs anywhere in the string that is the value of $hostdescr.)

```
$anywhere=anystr($hostdescr,"Agent");
```

14.7 AppendRdn

Syntax:
appendRdn(<oi>, <stringRdn>);
where <*oi*> is of type Object Instance and <*stringRdn*> is of type OCTET STRING.

Return Value: Object Instance.

Used to specify a new object instance (OI) from a supplied OI and a relative distinguished name (RDN) string. An RDN consists of a naming attribute and a value connected by the identity sign (=). Examples of naming attributes are systemId, networkId, internetClassId, and agentId. An RDN identifies an object uniquely relative to a superior object that "contains" it.

This built-in function can be used to set $pollfdn (the object that is the current target of the poll), which is of type Object Instance.

The following example illustrates this use. Let us suppose that you want to design a template that retrieves SNMP attribute values from an SNMP agent. The IsSnmpSystemUp template, shipped with Solstice EM, is an example of such a template. IsSnmpSystemUp polls the agent system for its system description in order to verify that the SNMP daemon is running. This requires that the template set the $pollfdn to point to the internetSystem group of the SNMP agent. To set the $pollfdn to point to the internetSystem group, the SetInternetSystem condition must first locate the cmipsnmpProxyAgent distinguished name (FDN). The cmipsnmpProxyAgent is the object in the MIS that represents the agent on the system being managed. The various groups in the SNMP agent are represented by objects "contained" in the cmipsnmpProxyAgent object. These "containment" relationships are reflected in the path to the object specified in the FDN. In the IsSnmpSystemUp template, the appendRdn() function is used to construct an FDN that points to the internetSystem group object.

When a template is launched against a device selected in the Viewer, `$pollfdn` is initially set to the first FDN in `$pollFdnSet`, which is the set of FDNs identifying the managed objects that have been configured for the target device. However, if you have instructed Discover to search for RPC agents when populating the runtime database in the MIS, the `$pollFdnSet` for a target device may contain FDNs for RPC agents as well as the SNMP agent. Depending upon the order in which Discover found the agents on the devices in your network, the cmipsnmpProxyAgent FDN may or may not be the first FDN in $pollFdnSet.

The SetInternetSystem sample condition checks the initial `$pollfdn` to determine if it is an RPC agent FDN, and, if it is, the condition then searches the FDNs in the `$pollFdnSet` to find the FDN for the cmipsnmpProxyAgent object

```
$dnstr = AsnToStr($pollfdn,true);
$check = AnyStr($dnstr,"RPC");
if ($check == TRUE)
{
   $num = NumElements(&$pollFdnSet);
   $count = 1;
   while ($count <= $num)
   {
      $numstr = AsnToStr($count,TRUE);
      $dn = Extract(&$pollFdnSet,$numstr);
      $dn1 = Extract(&$dn,"distinguishedName");
      $dnstr = AsnToStr($dn1,TRUE);
      $res = AnyStr($dnstr,"cmipsnmpProxyAgentId");
      if ($res == TRUE)
      {
           $tmp = "/internetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0);
           $pollfdn = AppendRdn($dn,$tmp);
           $count = $num+1;
      }
      else
      {
           $count = $count+1;
      }
   }
}
else
{
   $tmp = "/internetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0);
   $pollfdn = AppendRdn($dn,$tmp);
}
true;
```

Figure 14-1 Sample SetInternetSystem Condition

Once the SetInternetSystem condition has located the cmipsnmpProxyAgent, it then uses the `appendRdn()` function to form an FDN that points to the internetSystem group contained in that SNMP agent. For example, let us suppose that an IsSnmpSystemUp request launched against the SNMP host `bigguy` has its `$pollfdn` set to the following `cmipsnmpProxyAgent` FDN:

```
/systemId=name:”gatoloco”/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2 4 1
0}/cmipsnmpProxyAgentId=”bigguy”
```

The `SetInternetSystem` sample condition then resets the value of `$pollfdn` to point to the RFC 1213 internet`System` group object via an appendRdn statement:.

```
$tmp = “/internetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0);
$pollfdn = AppendRdn($dn,$tmp);
```

The affect of this `appendRdn` operation on our request launched against the host `bigguy` is to change the value of `$pollfdn` to the following:

```
/systemId=name:”gatoloco”/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2 4 1
0}/cmipsnmpProxyAgentId=”bigguy”/InternetClassId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6 1 2 1 1 0}
```

This function only appends a single RDN. Although in some contexts a string may contain several names separated by a slash (/), if such a string is given as the second argument to `appendRdn`, only the first RDN is appended

Another use for the `extract()` function is to pull out attribute values from events. For example, if `$eventInfo` is an enterpriseSpecificTrap, `extract()` can be used to get the specific type of the trap as follows:

```
$spec_trap_type = Extract(&$eventInfo,”probableCause”);
```

14.8 AsnToStr

Syntax:
`AsnToStr( <asn1_value>, <fTranslate>);`
where <*asn1_value*> is of type `Asn1Value` and <*fTranslate*> is of type BOOLEAN.

Return Value: `OCTET STRING`.

This function can be used to build an equivalent string representation of an `ASN1value`. It takes two arguments:

<*asn1_value*> is the value whose string representation you want.

<*fTranslate*> controls the choice of format for an object name. If <*fTranslate*> is `TRUE`, the resulting string OIDs are represented by equivalent string names. Otherwise, the resulting string OIDs are represented in curly brace form (for example, as {1 2 3 1} rather than its equivalent string form). For example:

```
$destIp=NameToAddress($host);
$ipStr=asnToStr($destIp,FALSE);
```

14.9 Defined

Syntax:
`defined(&<Var>);`
where <*Var*> is a variable; or

`defined(&<Attr>);`
where <*Attr*> is an attribute.

Return Value: `BOOLEAN`.

Checks whether the variable <*Var*> or the attribute <*Attr*> has a value. Returns `TRUE` if so, `FALSE` if the name or attribute has not been assigned a value or is not a valid name. A valid name is any of the assigned names of system variables (refer to Table 13-2 on page 13-6), a user-defined variable local to the request, or any of the names or attributes occurring in the request template. For

example, the IsSystemDesc sample condition, shown below, will return true if the device has responded to a get request for its SNMP sysDescr attribute value.

```
defined(&sysDescr);
```

Figure 14-2 IsSystemDesc Sample Condition

However, if this condition is used to test repetitively for availability of the SNMP device, an undefine should be executed before each subsequent test. This is necessary since the attribute value will remain defined in the request if a previous define returned successful.

14.10 Extract

Syntax:
Extract(&<*listvalue*>, <*subName*>);
where <*listvalue*> *is of type* LIST *and* <*subName*> *is of type* OCTET STRING.

Return Value: Asn1Value.

From <*listvalue*>, returns the value of the subcomponent identified by the string <*subName*>. <*subName*> specifies the name of the type of the component. Returns a null Asn1Value if the first parameter <*listvalue*> is not a LIST whose type is a SEQUENCE or SET. In the following example,

```
$ele1=Extract(&collectionInfoList,"1");
```

$ele1 will e assigned the first set from collectionInfoList, which is a SET OF SEQUENCE.

In the following example, the ASN1 value for one attribute in the sequence of attributes in a communicationsAlarm is extracted using the tag probableCause that identifies that attribute:

```
$cause = Extract(&$eventInfo, "probableCause");
```

However, a single `extract()` call will only extract from the first layer of components. For example, suppose that you want to extract oldAttributeValue from an attributeValueChange event notification in the system variable $eventInfo. The event contains an ASN.1 attributeValueChangeDefinition, which is itself a construct containing both the old and new attribute values. (The structure of attributeValueChange events is described in Chapter 15, "Adding New Event Types.") To get the oldAttributeValue, you could first assign

```
$attrChange = extract(&$eventInfo,"attrValueChangeDefinition");
```

`attributeValueChangeDefinition` to a variable `$attrChange` and then call `Extract()` again to pull `oldAttributeValue` from that variable:

```
$oldAttrVal = extract(&$attrChange,"oldAttrValue");
```

14.11 FinalStr

Syntax:
`finalstr(`<*firststr*>, <*secondstr*>`);`
where <*firststr*> and <*secondstr*> are of type `OCTET STRING.`

Return Value: `BOOLEAN.`

Checks whether `secondstr` appears in the end of <*firststr*>. (A string constant is enclosed in double quotes. In the following example `$at_end_of_str` is a Boolean variable that will be assigned the value true if "IPX" occurs at the end of the string that is the value of `$hostdescr`.)

```
$at_end_of_str = finalstr($hostdescr,"IPX");
```

14.12 FirstStr

Syntax:
`FirstStr(` <*first-string*>, <*delimiter-string*>`);`
where <*string1*> and <*string2*> are of type `OCTET STRING.`

Return Value: `OCTET STRING.`

Returns the first string delimited by <*delimiter-string*>. For example:

```
$x = “gatoloco dokusan columbine”;
$machine1 = firststr($x, “ ”);
```

`$machine1` will have “`gatoloco`” as its value.

14.13 GetTimeStamp

Syntax:
`getTimeStamp`(<*cur_time*>);
where <*cur_time*> is of type `GeneralizedTime` in the format YYYYMMDDHHMMSS.

Retrieves the current time of the host where the MIS is running. For example:

```
$curtime = getTimeStamp();
if (($curtime - $eventTime) > 600) { print($curtime);}
else {print($eventTime);}
```

14.14 Include

Syntax:
`include(&`<*list-var*>`,` <*value*>`);`
where <*list-var*> is a list variable of ASN.1 type SET or SEQUENCE. <*value*> contains the value to be included in the list.

Return Value: Boolean. True if the include operation was successful; false if not successful.

To add values to a list using `include()`, the list variable should be initialized first. For example:

```
$mylist = StrToAsn("SET","");
$ifint = 1;
$ifStr = "ifIndex";
include(&$mylist,$pollfdn);
include(&$mylist,$ifint);
include(&$mylist,$ifStr);
```

14.15 InitialStr

Syntax:
`initialstr(` <*firststr*>, <*secondstr*>`);`
where <*firststr*> and <*secondstr*> are of type `OCTET STRING`.

Return Value: `BOOLEAN`.

Returns True if `<secondstr>` appears initially in `<firststr>`. (A string constant is enclosed in double quotes.)Returns FALSE otherwise. In the following example, `$at_start_of_str` is a Boolean variable that will be assigned the value true if "Sun" occurs at the beginning of the string that is the value of `$hostdescr`.

```
$at_start_of_str = initialstr($hostdescr,"Sun");
```

14.16 IsChoice

Syntax:
`IsChoice(&`<*Var*>`);`
where <Var> is a variable; or

`IsChoice(&`<*Attr*>`);`
where <*Attr*> is an attribute.

Return Value: `BOOLEAN`.

Returns TRUE if the variable or attribute is a `choice`. Returns FALSE otherwise. For example:

```
$choice = IsChoice(&accessControlInfo);
```

14.17 IsList

Syntax:
`IsList(&<`*Var*`>);`
where <Var> is a variable; or

`IsList(&<`*Attr*`>);`
where <*Attr*> is an attribute.

Return Value: `BOOLEAN`.

Returns TRUE if the variable or attribute is a LIST. Returns FALSE otherwise. For example:

```
$listVar = IsList(&collectionInfoList);
```

14.18 Mail

Syntax:
`Mail(<`*addr*`>,<`*message-text*`>);`
where <addr> and <message-text> are both of type OCTET STRING.

Return Value: None.

Sends an e-mail message. This is the equivalent of the MAIL action. <*addr*> is a string containing the e-mail address. <*message-text*> is a string containing the message text. RCL variables can be interspersed in the string; for example:

```
Mail("netMgr@Eng","linkDown trap from $pollfdn");
```

14.19 NameToAddress

Syntax:
`NameToAddress( <hostname>);`
where <hostname> is of type `OCTET STRING.`

Return Value: `OCTET STRING.`

Returns the IP address of the host whose name is <*HostName*>. In the following example, the NameToAddress function is used to set up an IP address for a SendTrap operation.

```
$destIpAddr = NameToAddress($host);
SendTrap($destIpAddr,$eventType,$eventInfo);
```

14.20 NameToOid

Syntax:
`NameToOid( <Name>);`
where <Name> is of type `OCTET STRING.`

Return Value: `OBJECT IDENTIFIER.`

Returns the OID (Object IDentifier) of the object whose name is <*Name*>. This may be any name in the MIT that has been assigned an OID, such as an attribute, a name binding, and so on. In the following example, the OID corresponding to `internetAlarm` is assigned to a user variable `$ncType`.

```
$ncType=NameToOid("internetAlarm");
```

14.21 NumElements

Syntax:
`NumElements(&<Var>);`
where <Var> is a variable that is a `LIST`*; or*

`NumElements(&<Attr>);`
where <*Attr*> is an attribute that is a `LIST`.

Return Value: `INTEGER`.

Returns the number of elements contained in a variable or an attribute that is a `LIST`. Returns 0 if the variable or attribute is not a `LIST`. This function can be used, for example, to check how many managed objects are listed in the `$pollFdnSet` — the set of FDNs pointing to the managed objects that have been configured for the device that the request has been launched against. This following statement illustrates this use:

```
$numFdns = NumElements(&$pollFdnSet);
```

14.22 OiNameToOi

Syntax:
`OiNameToOi(`<*name*>`);`
where <*name*> *is of type* `OCTET STRING.`

Return Value: `Object Instance.`

Returns the object instance for the object whose name is supplied as quoted string <*Name*>. In the following example, an ObjectInstance will be returned from the distinguished name string constant.

```
$oi = OiNameToOi("/systemId=name:\"bigguy\"/topNodeDBId=NULL/topoNodeId=5");
```

14.23 OiToOiName

Syntax:
`OiToOiName(` <*instValue*>`);`
where <*instValue*> *is of type* `ObjectInstance.`

Return Value: `OCTET STRING.`

Returns the name of the object instance <*inst*> in the format the server uses. This is the inverse of the `OiNameToOi` function.

```
$nm = OiToOiName($oi);
```

14.24 Print

Syntax:
print(<*Var*>);
print(<*AttrName*>);
print(<*Constant*>);

Return Value: BOOLEAN. Returns TRUE.

This function is used to print the value of the supplied variable, attribute, or constant. This function will print messages only if the following `em_debug` command has been invoked in a shell:

```
%em_debug -c "on misc_stdout"
```

For more information on running the `em_debug` utility, refer to the "Building Request Templates" chapter in the *Solstice Enterprise Manager Administration Guide*."

14.25 SendAction

Syntax:
`<Var> = SendAction(` <*dest_oi*>, <*ActionInfo*>, &<*result*>`);`
where <*dest_oi*> is of type `ObjectInstance` (an ASN.1 value) and <*ActionInfo*> is an `OCTET STRING` constant. The address of <*result*> is also passed. The variable <*result*> needs to be initialized before being passed to `SendAction()`. <*Var*> is a BOOLEAN variable.

Return Value: `BOOLEAN.`

The return value <*Var*> will be true if the action request is issued, otherwise false. However, even if <*Var*> is true, this does not mean the action request was successful. To determine whether the action request was successful, you will need to check the `$messType` system variable.

Sends an `M-ACTION` to the destination object instance <*dest_oi*>, using <*ActionInfo*>. The second argument is a string in the following form:.

```
"{ <actionName>, <actionArgs>}"
```

For example, suppose there is an object

```
/systemId=titleist/counterObject=4
```

which has an action `incrementCounter` defined in the appropriate GDMO document. This action takes as its argument a name of a counter — for example, “alarm_counter” might be such a name. The following condition sends a CMIS M-ACTION to `counterObject=4` with the parameter “alarm_counter”:

```
$dn=OiNameToOi(”/systemId=name:\”titleist\”/counterObject=4”);
$result=0;
$ret = SendAction($dn,”{incrementCounter,”alarm_counter”}”,&$result);
```

You may then wish to transition to another state to check for a response; for example:

```
if ($ret AND ($messType == 10)) {print($result);}
```

Another example:

```
$dn=OiNameToOi(“/systemId=name:\”bigguy\”/topoNodeDBId=NULL”);
$result = 0;
$ret = SendAction($dn,”{topoNodeGetByType,\”Host\”},&$result);
```

After issuing this SendAction request, you can then check for a `$messType` of 10 in a separate condition:

```
if ($ret AND ($messType == 10)) {print($result);}
```

RCL functions can also be used to extract information from <*result*>.

14.26 SendEvent

Syntax:
`sendEvent(`<*oc_name*>`,`<*oi*>`,`<*eventTypeName*>`,`<*event-notification*>`);`

where <*oc*> is of type `OCTET STRING`, <*oi*> is of type `objectInstance`, and <*eventTypeName*> is of type OCTET STRING.
<*event-notification*> is the event to be logged, comprised of ASN.1 attributes. The attributes must be those that would be appropriate for the type of event notification specified by <*eventTypeName*>. (The structure of the default EM event types is described in Chapter 15, “Adding New Event Types.”)
<*oc_name*> is a string that specifies the object class of which <*oi*> is an instance.

Return Value: None.

`sendEvent()` has a function similar to the other alarm logging functions, such as `alarm()` and `alarmOi()`, but `sendEvent()` can be used to log events other than nerveCenterAlarms. For example, internetAlarms or CMIP event notifications defined by the ISO/ITU X.733 standard, such as a `communicationsAlarm`, can be sent to the alarm log using the `sendEvent()` function. In the following example a `communicationsAlarm` is logged using the `sendEvent()` function.

```
$event=strToAsn(“Notification-ASN1Module.AlarmInfo”, “{4, major}”;
sendEvent(“system”,$pollfdn,”communicationsAlarm”,$event);
```

This example generates a communicationsAlarm with a `probableCause` value of 4 and a `perceivedSeverity` of major.

14.27 SendTrap

Syntax:
`SendTrap(` <*dest*>`,` <*type*>`,` <*info*>`);`
where <*dest*> is an `OCTET STRING` constant, <*type*> is an event type, and <*info*> is of type `InternetAlarmInfo`.

Return Value: `BOOLEAN`.

Sends a trap to the destination IP address <*dest*>, using <*type*> and <*info*>.
The appropriate <*type*> can be obtained from `$eventType`.
The appropriate <*info*> can be obtained from `$eventInfo`.

Returns TRUE is successful, FALSE otherwise. For example:

```
$IpAddr = NameToAddress($host);
SendTrap($IpAddr,$eventType,$eventInfo);
```

14.28 Set

Syntax:
<Var> = Set(<*oi*>, <*modList*>);
where <*oi*> *is of type* ObjectInstance. The argument <*modList*> is a modification list whose structure is defined in nc.asn1 as EM-NC-ASN1.ModificationList. <*Var*> is a BOOLEAN variable.

Return Value: BOOLEAN.

Performs an M-SET on the object instance specified by <*oi*>. Set returns TRUE if the SET request has been issued.

The following example is a condition that does an M-SET on the topoNodeState attribute of a topology node.

```
$dn=strToAsn("CMIP-
1.ObjectInstance","distinguishedName:{{{systemID,\"titleist\"}},{{topoNodeDBId,NULL}},
{{topoNodeId,0}}}");
$arg = strToAsn("[12] IMPLICIT SET OF SEQUENCE { modifyOperator [2] IMPLICIT ModifyOperator
DEFAULT replace, attributeId AttributeId, attributeValue ANY DEFINED BY attributeId }",
"{{replace,topoNodeState,5}}");
Set($dn,$arg);
```

After invoking Set, you can check for success of the operation by checking the $messType system variable. If $messType has a value equal to 9, the set was successful. Otherwise, $messType will be set to a different value if the Set was unsuccessful. For example, if the set was directed at an invalid object instance, $messType would be set to 15, indicating no such object instance. The statement that checks for $messType value should be evaluated in a different condition from the one that issues the Set.

Another example:

```
$dn = OiNameToOi("systemId=name:\"solpuppy\"/topoNodeDBId=NULL");
$mlist = strToAsn("EM-NC-ASN1.ModificationList","{{attributeId topoNodeDisplayStatus,
{\"Down\",5}}}");
$ret = set($dn,$mlist);
```

After the M-SET is executed, you may want to transition to another state to check for a response in another condition, such as the following:

```
if ($ret AND ($messType == 9))  {print($ret);}
```

14.29 *SnmEventRequest*

Syntax:
`snmEventRequest(<oi>,` <*EventRequest*>, &<*result*>`);`
where <*oi*> is of type ObjectInstance, <*EventRequest*> is of either of type `OCTET STRING` or of type `Asn1Value.`<*result*> is a variable that has already been initialized.

Return Value: `BOOLEAN`. The return value is true if the action request was issued, otherwise false.

This function is used to issue a SunNet Manager event request to an SNM agent or proxy via RPC protocol.

The response to the action is set in the variable <*result*>. The `$messType` system variable should be checked to determine if a response to the action has been received. If `$messType` is 10, this indicates a response has been received to the action request.

<*EventRequest*> is a sequence of ASN.1 attributes as described in Table 14-2. The order of occurrence in the table is the order within <*EventRequest*>.

Table 14-2 Arguments in <*EventRequest*>

Argument	Data Type	Description	Required/Optional
agentHost	OCTET STRING	Name of target agent system	Required
agentProgram	INTEGER	RCP number of agent	Required
agentVersion	INTEGER	Agent's RPC version number	Required
timeout	INTEGER	Maximum time (in seconds) to wait for response from agent before request fails	Required
interval	INTEGER	Polling interval.(in seconds)	Required
group	OCTET STRING	Name of attribute group	Required
threshold	SEQUENCE of ■ attrName — OCTET STRING ■ attrType — INTEGER ■ relop — INTEGER ■ threshValue — OCTET STRING ■ priority — ENUMERATED — low (1) — medium (2) — high (3)	 •Name of the attribute used to define the threshold •Data type of the operands for relop. See Table 14-4. •Relational operator used in defining the threshold. See Table 14-3. •Threshold value to check for • Priority assigned to an SNM event generated if the threshold is crossed	Required
proxyHost	OCTET STRING	Name of a proxy system if a proxy agent is being used to access the agent system	Optional
count	INTEGER	Specifies the number of polls before terminating. If count is set to 0, this indicates that polling is to continue until request is killed.	Optional
optionalArgs	SEQUENCE of ■ name — OCTET STRING ■ value — OCTET STRING	Optional arguments are agent-specifc. For example, requests to na.ping use these to set packet size, time to wait for echo replies, etc.	Optional

Table 14-2 Arguments in <*EventRequest*>

Argument	Data Type	Description	Required/Optional
key	OCTET STRING	Row in a table. Entire table is used if no key is specified. Interpretation is agent-specific. For example, for the na.diskinfo agent this is the name of a filesystem partition.	Optional
flags	INTEGER	Request option flags. Currently defined is NETMGT_RESTART.	Optional
rendezHost	OCTET STRING	Name of host where rendezProgram (typically na.event) is running	Optional
rendezProgram	INTEGER	RPC number of the program that is to receive the events (typically the Event Dispatcher — na.event)	Optional
rendezVersion	INTEGER	rendezProgram's RPC version number	Optional

The `agentVersion` number can be found in the listing for the agent in `/etc/inetd.conf`. For example, 10 is the version number for `na.snmp` in the following inetd.conf entry:

```
na.snmp/10  tli rpc/udp  wait  root  /opt/SUNWconn/snm/agents/na.snmp  na.snmp
```

The relational operators that can be used to define thresholds are described in Table 14-3.

Table 14-3 Relational Operators in SNM Request Thresholds

Integer Value	Relational Operator
0	No operation
1	Equal To
2	Not Equal To
3	Less Than
4	Less Than or Equal To

Table 14-3 Relational Operators in SNM Request Thresholds

Integer Value	Relational Operator
5	Greater Than
6	Greater Than or Equal To
7	Value has changed
8	Value Increased By
9	Value Decreased By
10	Value Increased By More Than
11	Value Increased By Less Than
12	Value Decreased By More Than
13	Value Decreased By Less Than

The data types for operands of relational operators (attrType) are defined in Table 14-4.

Table 14-4 Data Types for Threshold Operands

Integer Value	Data Type
1	short
2	unsigned short
3	int
4	unsigned int
5	long
6	unsigned long
7	float
8	double
9	null-terminated ASCII string
10	opaque octet stream
11	Internet address
12	struct timeval
13	seconds since 1/1/70
14	enumerated type

When an SNM agent or proxy detects that a specified threshold has been crossed, an event is sent to the SNM Event Dispatcher (na.event), which is called the *rendezvous*. `rendezHost` is the name of the machine whose Event Dispatcher is to receive the SNM event. By default, `rendezHost` is the name of the MIS machine that initiated the event request. The SNM Event Forwarder (em_snmfwd) on the MIS machine receives the event from the Event Dispatcher and converts it to an `snmAlarmEvent` (a type of CMIP event notification). The Event Forwarder maps SNM event priorities to the `perceivedSeverity` values used by the Alarm Service in the manner indicated in Table 14-5. The SNM Event Forwarder sends snmAlarmEvents to the MIS.

Table 14-5 Mapping of SNM Event Severities

SNM Event Priority	perceivedSeverity Value	Default Icon Color
Low	Minor	Cyan
Medium	Major	Orange
High	Critical	Red

The following condition builds and sends an SNM event request targeted at the `ping-reach` managed object, which represents the `reach` attribute group supported by the `na.ping` proxy agent.

```
$eventRequestStr = “{agentHost \””;
$eventRequestStr = StrCat($eventRequestStr,$Hostname);
$agentStr = “\”,agentProgram 100115, agentVersion 10, timeout 10,
interval 12, group \”reach\”, threshold
{\”triptime\”,21,2,\”1\”, medium}}”;
$eventRequestStr = StrCat($eventRequestStr,$agentStr);
$request_handle = 0;
snmEventRequest($pollfdn,$eventRequestStr,$request_handle);
```

In this example, 100115 is the RPC number of the `na.ping` agent and 10 is the version number of this agent. The polling interval is set to 12 seconds. The threshold is triptime not equal to 1. If the threshold is crossed, the ping agent is to generate an SNM event with a medium priority. The variable `$hostname` holds the hostname of the target of the request; this information could be extracted from the `$pollFdnSet`. (An example that extracts the hostname

from $pollFdnSet can be found in Section 13.9.3, “WHILE Constructs.”) $pollFdnSet is a system variable that contains the set of distinguished names (FDNs) pointing to managed objects configured for the target device.

To receive SNM event notifications (snmAlarmEvents) generated by the proxy agent in response to a crossed threshold, the request that initiates the SNM event request can also use the event subscription functions, such as subscribeOi(), to subscribe for snmAlarmEvents. For more information on using RCL templates to launch SNM event requests, refer to Chapter 11, “Building Templates for SunNet Manager Event Requests.”

14.30 SnmKillRequest

Syntax:
snmKillRequest(<*oi*>,<*EventReply*>);
where <*EventReply*> is the result returned in the <*result*> variable passed to snmEventRequest().

Return Value: BOOLEAN. Returns true if a correct argument is passed.

Issues a request to kill a SunNet Manager event request that had been issued by a call to the snmEventRequest function.

14.31 StrToAsn

Syntax:
StrToAsn(<*strAsn1Type*>, <*strAsn1Value*>);
where <*strAsn1Type*> and <*strAsn1Value*> are of type OCTET STRING.

Return Value: Asn1Value

This function takes two arguments:
<*strAsn1Type*> is the canonical text representation of the Asn1Type.
<*strAsn1Value*> is the text representation of the Asn1Value. For example:

```
$asn1_int = StrToAsn(“INTEGER”,”1000”);
$asn1_bool = StrToAsn(“BOOLEAN”,”FALSE”);
```

returns the ASN1 encoding of <*strAsn1Value*>, but returns FALSE if the arguments are invalid. In the following example, strToAsn is used to build an internetAlarmInfo value for a SendTrap operation:

```
$alarminfo = "{ ";
$alarminfo = strcat($alarminfo,$cause);
$alarminfo = strcat($alarminfo,$trans_domain);
$alarminfo = strcat($alarminfo,$traddr);
$alarminfo = strcat($alarminfo,$access);
$alarminfo = strcat($alarminfo,$ainfo);
$alarminfo = strcat($alarminfo," }");
$internetAlarmInfo = strToAsn("IimcCommonDef.InternetAlarmInfo",$alarminfo);
$eventInfo = $internetAlarmInfo;
SendTrap($destIp,$itType,$eventInfo);
```

14.32 StrCat

Syntax:
strcat(<*string1*>, <*string2*>);
where <*string1*> and <*string2*> are of type OCTET STRING.

Return Value: OCTET STRING.

Returns a string built by concatenating <*string1*> and <*string2*>. For example:

```
$probCOid = "{1 3 6 1 4 1 42 2 2 2 9 1 9 1 999 1 1 1}";
$cause = "probableCause globalValue: ";
$cause = Strcat($cause,$probCOid);
```

14.33 Strstr

Syntax:
strstr(<*string1*>, <*string2*>);
where <*string1*> and <*string2*> are of type OCTET STRING.

Return Value: OCTET STRING.

Returns a string with the first occurrence of <*string2*> in <*string1*>. For example:

```
$s = “How tall is my dentist”;
$res = strstr($s, “my d”);
```

`$s` will contain “`my dentist`.”

14.34 Strstrplus

Syntax:
`strstrplus(` <*string1*>, <*string2*>`);`
where <*string1*> and <*string2*> are of type `OCTET STRING.`

Return Value: `OCTET STRING.`

Returns the remainder of a string after the first occurrence of <*string2*> in <*string1*>. For example:

```
$s = “How tall is my dentist”;
$res = strstrplus($s, “my d”);
```

`$s` will contain “`entist`.”

14.35 Subscribe

Syntax:
`subscribe(` <*event_name_str*>`);`
where <*event_name_str*> is of type `OCTET STRING.`

Return Value: INTEGER.

Used to subscribe to events of the type identified in <*event_name_str*>. The <*event_name_str*> must be one of the event names that appears as a `NOTIFICATION` in a GDMO document that the MIS knows about. The following condition subscribes to receive SNMP traps. This condition could be used to define an initial transition out of the Ground state.

```
$itindx=Subscribe("internetAlarm");
$itType=NameToOid("internetAlarm"); true;
```

The OID retrieved using `NameToOid()` can be tested against the system variable `$eventType` to determine if an event of the subscribed type has arrived, as, for example, in the following condition:

```
$eventType == itType;
```

Note – In composing a condition that tests for object creation, use `subscribeOi()`, rather than `subscribe()`, and subscribe to the creation of only a specific type of object. Do not listen for all object creations, as this can result in a deadlock (infinite loop) situation.

If a problem prevents the subscription from being implemented, a `-1` is returned. Otherwise, this function returns a handle index; that is, a unique index for a subscription in a request. This handle can be passed in a call to the `unsubscribe()` function to terminate the event subscription.

14.36 SubscribeFilter

Syntax:
`<result> = subscribeFilter(`<*cmis_filter_string*>`);`
where <*cmis_filter_string*> is a CMIS filter construct of type OCTET STRING.

<*result*> `= subscribeFilter(`<*asn1_cmis_filter*>`);`
where <*asn1_cmis_filter*> is a CMIS filter construct of type `Asn1Value`.

Return Value: INTEGER. <*result*> is a a handle of type INTEGER. If there is a failure that prevents implementation of the subscription, `-1` is returned. Otherwise, a handle index is returned; this handle can be passed to the `unsubscribe()` function to terminate the subscription.

It is recommended that you check for a return value of -1 to determine if errors have been encountered. For example, if the syntax of the CMIS filter passed as <*cmis_filter_string*> is incorrect, `-1` will be returned.

This command subscribes for events that match the specified CMIS filter. If an event passes the filter, it will be forwarded to the request. The CMIS filter can be passed to the function either as a string or as an ASN.1 value. For information on the format of a CMIS filter, refer to the "CMIS Scoping and Filtering" appendix in the *Solstice Enterprise Manager Reference Manual*.

Note – If you want to use the `unsubscribe()` function to turn off a subscription created with the `subscribeFilter()` function, you cannot invoke `unsubscribe()` in a condition that is evaluated as the result of an event received on this subscription. Your request should transition to another state before calling the `unsubscribe()` function.

Considerations

- If you invoke two subscriptions using CMIS filters and one filter selects a subset of the other, your request will receive duplicate events for the overlapping subset. An example of this would be the following two subscriptions:

```
$index = subscribeFilter("CMIP-1.CMISFilter","or: {item:
equality: {objectClass,mosi}, item: equality: {eventType,
internetAlarm}}");

$index = subscribeFilter("CMIP-1.CMISFilter", "item: equality:
{eventType, internetAlarm}");
```

  If these two subscriptions are invoked, each incoming internetAlarm is forwarded twice to the request. You should tailor your event subscriptions so as to avoid this duplication of events.

- When a subscription is created using a CMIS filter, every event in the system is checked against that filter. Additional filter subscriptions thus place an increasing load on the MIS. To avoid an adverse impact on performance, it is recommended that you exercise care in the use of filter subscriptions.

Examples

- The following is an example of a subscription that passes a CMIS filter that forwards all events to a request:

```
$index = subscribeFilter("and : { }");
```

This same CMIS filter could be passed as an ASN.1 value:

```
$filter = strToAsn("CMIP-1.CMISFilter","and : { }");
$index = subscribeFilter($filter);
```

- A CMIS filter could be used to forward all events of a specified managed object class to the request. For example, all nerveCenterAlarm, generated using the RCL alarm-logging functions (`alarm()`, `alarmOi()`, `alarmStr()`)s are instances of a managed object class called `mosi`. The following filter forwards to the request all events whose managed object class is mosi.

```
$filter = strToAsn("CMIP-1.CMISFilter", "item: equality
{managedObjectClass, mosi}");
$index = subscribeFilter($filter);
```

This same CMIS filter could be passed as a string:

```
$index = subscribeFilter("item: equality {managedObjectClass,
mosi}");
```

- The following example uses a CMIS filter that selects all events whose managed object class is `system` and whose eventType is communicationsAlarm:

```
$filter = strToAsn("CMIP-1.CMISFilter", "and: {item: equality:
{managedObjectClass, system}, item: equality: {eventType,
communicationsAlarm}}");
$index = subscribeFilter($filter);
```

14.37 SubscribeOi

Syntax:
`subscribeOi(` <*event_name_str*>, <*event_oc_str* >, <*event_oi*>`);`
where <*event_name_str*> and <*event_oc_str*> are of type `OCTET STRING` and <*event_oi*> is of type `ObjectInstance`.

Return Value: INTEGER.

```
$itindx=subscribeOi("internetAlarm","",$pollfdn);
$itType=NameToOid("internetAlarm"); true;
```

Used to subscribe to events of a specific type that concern objects of a specific type, a specific object class, or a specific object instance. The <*event_name_str*> must be one of the event names that appears as a `NOTIFICATION` in a GDMO document that the MIS knows about.

The <*event_oi*> argument is an `ObjectInstance` specifying the instance of interest.

The <*event_oc_str*> argument specifies the object class of interest. The function will accept an empty string "" as the value of <*event_oc_str*>, and in that case supplies the class to which <*event_oi*> belongs. However, for performance reasons it is preferable to supply the object class explicitly.

If a failure occurs that prevents implementation of the subscription, `-1` is returned. Otherwise, this function returns a handle index; that is, a unique index for a subscription in a request. This handle can be passed in a call to the `unsubscribe()` function to terminate the event subscription.

Currently when a request is disabled or deleted, its subscription(s) are automatically deleted also.

Note – In composing a condition that tests for object creation, use `subscribeOi()`, rather than `subscribe()`, and subscribe to the creation of only a specific type of object. Do not listen for all object creations, as this can result in an infinite loop.

14.38 TrapGenericType

Syntax:
`TrapGenericType( <info>);`
where <*info*> is an event of type `InternetAlarmInfo`.

Return Value: `INTEGER`.

Returns the number of the `GenericType` of the received trap (for example, from `$eventInfo`). For example:

```
$gnum=TrapGenericType($eventInfo);
```

The possible return values are described inTable 14-6.

Table 14-6 Standard SNMP Trap Types

Value of <generic-trap>	Trap Type	Description
0	coldStart	The originating SNMP device is reinitializing itself, typically due to unexpected reboot.
1	warmStart	The originating SNMP device is reinitializing itself, typically due to normal restart.
2	linkDown	One of the agent's communication links is down. The first name/value pair in the variable bindings is the ifIndex for the interface.
3	linkUp	One of the agent's communication links has come up. The first name/value pair in the variable bindings is the ifIndex for the interface.

Table 14-6 Standard SNMP Trap Types

Value of <generic-trap>	Trap Type	Description
4	authenticationFailure	The originating system has received a protocol message that has failed authentication.
5	egpNeighborLoss	An External Gateway Protocol peer has been marked down.
6	enterpriseSpecific	Further information about the event is indicated in the <*specific-trap*> field.

14.39 TrapSpecificType

Syntax:
`TrapSpecificType(` <*info*>`);`
where <*info*> is an event of type `InternetAlarmInfo.`

Return Value: `INTEGER`.

Returns the number of the `SpecificType` of the received trap (for example, from `$eventInfo`). For example:

```
$snum=TrapSpecificType($eventInfo);
```

14.40 Undefine

Syntax:
undefine(&<*Var*>);
where <*Var*> is a variable; or

undefine(&<*Attr*>);
where <*Attr*> is an attribute.

Return Value: `BOOLEAN`. Always returns TRUE.

Sets the variable <*Var*> or the attribute <*Attr*> to have no value.

The `IsSnmpSystemUp` sample template illustrates the use of the `undefine` function. The availability of the target device is tested using a call to `define` in the `IsSystemDescr` sample condition.

```
define(&sysDescr);
```

Figure 14-3 IsSystemDescr Sample Condition

Once this condition returns true, however, the value will always remain true unless the `undefine` function is called, as in the `UndefineSystemDescr` sample condition:.

```
undefine(&sysDescr);
```

Figure 14-4 UndefineSystemDescr Sample Condition

Once the value of sysDescr has been undefined, the IsSystemDescr sample condition can once again invoke the define the function to test for system availability.

14.41 Unixcmd

Syntax:
`Unixcmd(`<*command*>`,`<*arguments*>`);`
where <*command*> and <*arguments*> are both of type OCTET STRING.

Return Value: None.

Executes the indicated UNIX command. This is the equivalent of the UNIXCMD action. The <*arguments*> parameter can contain RCL variables, either alone or embedded inside a quoted string. For example:

```
UnixCmd("echo","$pollfdn > /tmp/mydata");
```

14.42 UnSubscribe

Syntax:
`unsubscribe(`<*subscription_handle*>`);`

where <*subscription_handle*> is an INTEGER value returned from a previous call of one of the subscription functions, such as `subscribe()` or `subscribeOi()`. This function can be used to turn off a previous event subscription.

Note – If you want to use the `unsubscribe()` function to turn off a subscription created with the `subscribeFilter()` function, you cannot invoke `unsubscribe()` in a condition that is evaluated as the result of an event received on this subscription. Your request should transition to another state before calling the `unsubscribe()` function.

Adding New Event Types 15

Agents located on devices in the network typically are designed to generate reports to managers on their own initiative when certain conditions are detected; these messages are called *event notifications.*

Notifications are defined for managed objects in accordance with the ITU X.722 Guidelines for the Definition of Managed Object (GDMO) standard. The MIS acquires knowledge of event notification types when the pertinent GDMO documents are loaded into the MIS at startup. The CMIP event notifications defined in EM by default are the following:

Defined by the ITU X.722 Definition of Management Information (DMI) standard:

- objectCreation
- objectDeletion
- attributeValueChange
- relationshipChange
- stateChange
- communicationsAlarm
- environmentalAlarm
- equipmentAlarm
- integrityViolation
- operationalViolation
- physicalViolation
- processingErrorAlarm
- qualityofServiceAlarm
- securityServiceOrMechanismAlarm

- timeDomainViolation

Defined by the ISO-Internet Management Co-existence (IIMC) standard:

- internetAlarm

Solstice EM-specific event notifications:

- snmAlarmEvent
- snmAlarmTrap
- nerveCenterAlarm
- coldStartTrap
- warmStartTrap
- linkDownTrap
- linkUpTrap
- authenticationFailureTrap
- egpNeighborLossTrap
- enterpriseSpecificTrap

Custom event notifications can be added. To add new event types to the MIS, do the following:

1. **Define the GDMO and ASN.1 documents in which the event notification is defined.**

2. **You may also want to provide GDMO documents in which an equivalent event log record is defined.**
   An event notification must have an appropriate event log record type in order for it to be logged. You can either provide a custom log record class definition, or alternatively, you can use one of the existing log record classes. A single log record class can be used to log a group of event notifications. The default mapping of event notifications to log record classes is shown in Table 15-1.

Table 15-1 Default Notification to Event log Record Object Class Mapping

Event Notification	Event Log Record Object Class
attributeValueChange	attributeValueChangeRecord
objectCreation	objectCreationRecord
objectDeletion	objectDeletionRecord
relationshipChange	relationshipChangeRecord
stateChange	stateChangeRecord

Table 15-1 Default Notification to Event log Record Object Class Mapping

Event Notification	Event Log Record Object Class
communicationsAlarm	emAlarmRecord
environmentalAlarm	emAlarmRecord
equipmentAlarm	emAlarmRecord
processingErrorAlarm	emAlarmRecord
qualityofServiceAlarm	emAlarmRecord
integrityViolation	securityAlarmReportRecord
operationalViolation	securityAlarmReportRecord
physicalViolation	securityAlarmReportRecord
securityServiceOrMechanismAlarm	securityAlarmReportRecord
timeDomainViolation	securityAlarmReportRecord
internetAlarm	emInternetAlarmRecord
nerveCenterAlarm	nerveCenterAlarmRecord

3. **Compile the GDMO and ASN.1 documents.**
   Compiling GDMO and ASN.1 documents is described in Chapter 16, "Adding a Managed Object Class to the MIS."

4. **Add an event to log record entry in the `init_user` file.**
   Each mapping of an event type to a log record type requires an entry in the `init_user` file. This file is located in the following directory: `/opt/SUNWconn/em/build/acct`

   An example of the format is indicated by the entry for `enterpriseSpecificTrap` event type:

```
# enterpriseSpecificTrap

SET
{
OI = ‘subsystemId=”EM-MIS”/listname=”event2ObjectClass”’
evr2oclist += ‘{ { eventypeoid enterpriseSpecificTrap,
objectClassoid emInternetAlarmRecord } }’
}
```

   In this example we see that enterpriseSpecificTrap events are logged as log records of type `emInternetAlarmRecord`.

5. **If you do *not* want your new event type automatically logged to the alarm log, modify the alarm log discriminator.**
   By default, the AlarmLog discriminator includes all event types except for the following:

   - snmAlarmEvent
   - objectCreation
   - objectDeletion
   - attributeValueChange
   - stateChange

   If you do not want your custom event automatically logged, use the Log Manager to edit the log discriminator to add your new event type to the list of excluded event types in the alarm log discriminator. Refer to the “Log Manager” chapter in the *Solstice Enterprise Manager Reference Manual* for more information.

6. **Restart the MIS.**
   After completing the above steps, use the `em_services` command to restart the MIS.

7. **Verify that the new event types are recognized and logged.**
   Generate events of the new type and verify that they can be logged. For example, the creation of new log records can be verified in OBED, Log Viewer, or Alarm Manager. You could write a Nerve Center subscription request template to verify that the new event type is recognized by the MIS.

15

Adding a Managed Object Class to the MIS

16

This chapter provides an example to illustrate how to add a GDMO object class to the Management Information Server (MIS). The GDMO and ASN.1 definition files used in this example are called `coffee.gdmo` and `coffee.asn1`, respectively. The managed object class being added to the MIS is called `coffee`. Code Example 16-1 lists the `coffee.gdmo` definition file, and Code Example 16-2 lists the `coffee.asn1` definition file. Both of these files are located in the `$EM_HOME/src/gdmo_sample` directory.

1. **From a directory in which you have write permissions, compile the `coffee.asn1` ASN.1 definition using the ASN.1 compiler as follows:**

```
hostname% $EM_HOME/bin/em_asn1 -o . coffee.asn1
```

   The "`-o .`" parameter specifies to place the output file produced in the current directory.

2. **Copy the ASN.1 output files produced into the `/var/opt/SUNWconn/em/usr/data/ASN1` directory as follows:**

```
hostname% cp 1.2.3.4.5.1 /var/opt/SUNWconn/em/usr/data/ASN1
hostname% cp Coffee-ASN1 /var/opt/SUNWconn/em/usr/data/ASN1
```

   The files are placed in the `/var/opt/SUNWconn/em/usr/data/ASN1` directory for safe keeping. This way, when a `pkgrm` command is invoked, these files remain in existence.

3. **Specify the GDMO definition of the object class(es) in a file with a suffix of** `.gdmo`**.**
   In this example, the `coffee.gdmo` file contains the GDMO definition of the `coffee` managed object class.

4. **Compile the** `coffee.gdmo` **GDMO definition using the GDMO compiler as follows:**

```
hostname% $EM_HOME/bin/em_gdmo -file coffee.gdmo
```

   The `-file` option causes `em_gdmo` to generate an output file.

5. **Copy the GDMO output file produced into the** `/var/opt/SUNWconn/em/usr/data/MDR` **directory as follows:**

```
hostname% cp G2_Coffee_Document /var/opt/SUNWconn/em/usr/data/MDR
```

6. **Go to the** `$EM_HOME/bin` **directory and, as** `root`**, invoke the** `em_services -r` **command to restart the MIS.**

```
# cd $EM_HOME/bin
# ./em_services -r
```

7. **As** `root`**, compose the new** `coffee` **object class with the** `em_compose_all` **command as follows:**

```
# ./em_compose_all $EM_HOME/src/gdmo_sample/coffee.gdmo
```

   This command composes all of the object types within a GDMO file. It is the equivalent of running `em_compose_poc` and `em_load_name_bindings` for all objects types within the GDMO file.

This procedure can be repeated for any GDMO object class you want to add to the MIS. Alternatively, you can edit the `em_startup` file, located in `$EM_HOME/bin`, so that the GDMO object classes you want to add to the MIS are added each time the MIS is started or re-started. You could add the

necessary `em_compose_all` <*filename*> command(s) at the end of the file, immediately preceding the `exit` command. If you want to create an instance of each object type, you must use the `em_objop` command.

16.1 GDMO and ASN.1 Used in the Examples

Following is a copy of the `coffee.gdmo` GDMO definition file. It is used to describe the `coffee` Managed Object Class:

Code Example 16-1 coffee.gdmo Definition File

```
MODULE "G2 Coffee Document"
coffee MANAGED OBJECT CLASS
        DERIVED FROM "Rec. X.721 | ISO/IEC 10165-2 : 1992" : top;
        CHARACTERIZED BY
                coffeePackage;
        REGISTERED AS { 1 2 3 4 5 6 3 1 };

coffeePackage PACKAGE
        BEHAVIOUR coffeePackageDefinition BEHAVIOUR DEFINED AS
              !This managed object class represents the coffee
                pot of Peet's coffee in Jon and Kevin's office!;
        ;
        ATTRIBUTES
                coffeePotNumber GET-REPLACE,
                coffeeBlend GET-REPLACE,
                coffeeReady GET-REPLACE;
        NOTIFICATIONS
                "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
                        objectCreation,
                "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
                        objectDeletion,
                "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
                        attributeValueChange;
        REGISTERED AS { 1 2 3 4 5 6 4 1 };

-- Name Bindings

coffee-system NAME BINDING
        SUBORDINATE OBJECT CLASS coffee;
        NAMED BY
       SUPERIOR OBJECT CLASS "Rec. X.721 | ISO/IEC 10165-2 : 1992" : system;
        WITH ATTRIBUTE coffeePotNumber;
        BEHAVIOUR coffee-rootBehaviour BEHAVIOUR DEFINED AS
```

Code Example 16-1 coffee.gdmo Definition File

```
                !This name is used to define the coffee object
                name binding!;
        ;
        CREATE;
        DELETE ONLY-IF-NO-CONTAINED-OBJECTS;
        REGISTERED AS { 1 2 3 4 5 6 6 1 };

coffee-coffee NAME BINDING
        SUBORDINATE OBJECT CLASS coffee;
        NAMED BY
        SUPERIOR OBJECT CLASS coffee;
        WITH ATTRIBUTE coffeePotNumber;
        BEHAVIOUR coffee-systemcoffee BEHAVIOUR DEFINED AS
                !This name is used to define the coffee object
                name binding!;
        ;
        CREATE;
        DELETE ONLY-IF-NO-CONTAINED-OBJECTS;
        REGISTERED AS { 1 2 3 4 5 6 6 2 };

-- Attributes

coffeePotNumber ATTRIBUTE
        WITH ATTRIBUTE SYNTAX Coffee-ASN1.CoffeeInteger;
        MATCHES FOR EQUALITY;
        BEHAVIOUR coffeePotNumberBehaviour BEHAVIOUR DEFINED AS
                !This is the naming attribute for  the coffee
                object.  This will always be 1 until Jon and
                Kevin get another pot working.!;
        ;
        REGISTERED AS { 1 2 3 4 5 6 7 1 };

coffeeBlend ATTRIBUTE
        WITH ATTRIBUTE SYNTAX Coffee-ASN1.CoffeeString;
        MATCHES FOR EQUALITY;
        BEHAVIOUR coffeeBlendBehaviour BEHAVIOUR DEFINED AS
                !This is the blend of coffee that is  brewing
                in the current pot, undoubtedly Peets,
                and probably French or Italian.!;
        ;
        REGISTERED AS { 1 2 3 4 5 6 7 2 };

coffeeReady ATTRIBUTE
```

Code Example 16-1 coffee.gdmo Definition File

```
        WITH ATTRIBUTE SYNTAX Coffee-ASN1.CoffeeInteger;
        MATCHES FOR EQUALITY;
        BEHAVIOUR coffeeReadyBehaviour BEHAVIOUR DEFINED AS
                !If this attribute is true there is coffee in
                the pot, you better hurry!;
        ;
        REGISTERED AS { 1 2 3 4 5 6 7 3 };
END
```

Following is a copy of the `coffee.asn1` ASN.1 definition file. It is used to describe the syntax and encoding rules for the `coffee` managed object class:

Code Example 16-2 coffee.asn1 Definition File

```
--  Asn1 Definitions needed by the coffee class
Coffee-ASN1 { 1 2 3 4 5 1 }

DEFINITIONS ::=
BEGIN

CoffeeInteger ::= INTEGER

CoffeeString ::= GraphicString

CoffeeBoolean ::= BOOLEAN

END
```

16

Adding a MIB to the MIS 17

The Solstice EM product is shipped with the file `sun.mib`, stored in `$EM_HOME/etc/snmp_mibs`. The same file was stored in `/opt/SUNWconn/snm/agents` with SunNet Manager 2.2 and later. Using the following procedure, you can use Solstice EM utilities to convert the objects defined in the Sun MIB to GDMO objects and then load those objects into the MIS.

1. **As `root`, edit `sun.mib` to check the following line:**

```
SUN-SNMP DEFINITIONS ::= BEGIN
```

   This string occurs approximately at line 13 in the file, and should appear exactly as shown above. The critical aspect is that, in order to conform to ASN.1 standards, the first letter must be capitalized. (The ASN.1 standard requires that the first letter be capitalized, but does not require capitalization of the rest of the module name.)

2. **Save and exit the file.**

3. **With `$EM_HOME/bin` in your PATH, run `em_cmib2gdmo` to process `sun.mib`:**

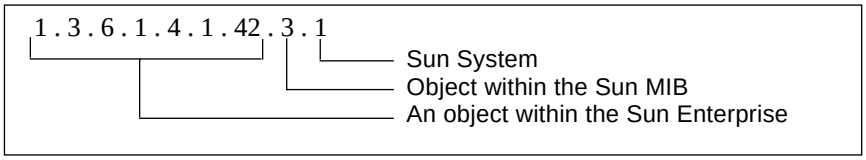
```
# em_cmib2gdmo sun.mib
```

   The preceding command produces the files `sunTYPE.asn1` and `sun.gdmo`.

Should you receive any error messages during the compilation of `sun.mib` about `em_cmib2gdmo` not being able to resolve OIDs, there are a couple of possibilities you can explore.

The first is to invoke the `em_cmib2gdmo` compiler using the `-l` parameter, meaning that you specify two schema files, but a GDMO and ASN.1 file are generated for only the second file. This is useful if one MIB imports information from another MIB, but you want to translate only the MIB with the dependency. Two of the more commonly-used MIBs are `rfc1312` and `rfc1155`, and they should be the first of the two MIBs specified with the `-l` option. For more information, see the "Compilers" chapter in the *Solstice Enterprise Manager Reference Manual*.

The second possibility is to manually edit `sun.gdmo` to replace all occurrences of a string followed by a question mark with that string followed by the numeric OID that corresponds to that string. Obtain the OID from the MIB definition in the original MIB file. An OID for a Sun machine might be:



Obtain the full OID from the MIB definition in `sun.mib`, as you submitted that file to the `em_cmib2gdmo` compiler.

4. **Load the GDMO definitions.**
   If the MIS *is not* running, invoke the commands below to move the generated GDMO and ASN.1 files to their correct destinations, build GDMO and ASN.1 objects, and start the MIS, during which definitions are loaded into the MIS.

```
# mv sun.gdmo $EM_HOME/etc/gdmo
# mv sunTYPE.asn1 $EM_HOME/etc/asn1
# $EM_HOME/bin/em_services -r
```

   If the MIS *is* running, enter the following command:

```
# em_gdmo -host <hostname> -file sun.gdmo
```

   The `-host <hostname>` parameter loads the new GDMO definition directly into the MDR, so you do not have to restart the MIS.

17

Adding an Object Class Based on an SNM Schema File to the MIS 18

This chapter provides an example on how to convert a SunNet Manager schema file to a GDMO document and, thence, add a GDMO object class for that schema to the Management Information Server (MIS).

For this example, we use the `diskinfo.schema` file that is shipped with SunNet Manager 2.x. This file resides in the `/opt/SUNWconn/snm/agents` directory by default. You must be `root` to perform this procedure.

1. **Copy `diskinfo.schema` to a directory in which `root` has write permission.**

2. **With `$EM_HOME/bin` in your PATH, enter the following command**:

```
# em_snm2gdmo diskinfo.schema <OID_branch>
```

   The <*OID_branch*> parameter is any unused OID branch number for the specified schema file. This number is used to assign a unique OID to every record type defined within the specified schema file. “0” and “1” are the only OID branch numbers that are already reserved:

   - “0” is reserved for SNM agent/proxy schemas, which can be found in `$SNM_HOME/agents`.
   - “1” is reserved specifically for the schema file `elements.schema`.

See the “Compilers” chapter in the *Solstice Enterprise Manager Reference Manual* for more information on determining which OID branch numbers are available or reserved.

The output of the preceding command is two files, `diskinfo.asn1` and `diskinfo.gdmo`, stored in the directory in which you invoked the command.

3. **If you want the `diskinfo` object class to be part of the MIS each time you start Solstice EM, copy the newly created `.asn1` and `.gdmo` files to the `$EM_HOME/etc/asn1` and `$EM_HOME/etc/gdmo` directories, respectively.**

Note – Files stored in the `$EM_HOME/etc/asn1` and `$EM_HOME/etc/gdmo` directories are compiled each time you run `em_services -r`.

At this point, after copying the `.asn1` and `.gdmo` files to their respective directories, you can run `em_services -r`, or you can proceed with the rest of the procedure. Running `em_services -r` automatically takes care of the rest of this procedure.

If you run `em_services -r` now, you can use the Topology Import/Export Tool to save your topology database, and use the `em_ncexport` utility to save your Nerve Center templates. After the platform has been reinitialized, you can import that information back into the MIS.

Keep in mind that running `em_services` means stopping and restarting the MIS, potentially disrupting applications that are using the MIS.

4. **Run `em_services -r` now, or proceed to Step 5 to continue.**
   If you run `em_services -r` now, you are finished with this procedure.

5. **Change directories to the `/var/opt/SUNWconn/em/usr/data/ASN1` directory.**
   This directory is intended for user-supplied ASN.1 output files.

6. **Enter the following command:**

```
# em_asn1 -o . <path>/diskinfo.asn1
```

The ASN.1 compiler creates the following files in the current directory:

```
1.3.6.1.4.1.42.2.2.2.12.100105
DiskinfoASN1
```

The `diskSpace` object class and the `diskSpace-rpcContainer` name binding are defined in the `diskinfo.gdmo` file. In the following steps, you load this object class and name binding into the MIS.

7. **Change directories to the `/var/opt/SUNWconn/em/usr/data/MDR` directory.**
   This directory is intended for user-supplied GDMO output files.

8. **Enter the following command:**

```
# em_gdmo -file <path>/diskinfo.gdmo
```

The GDMO compiler creates a file `RPC_Agent_-_diskinfo` in the current directory.

9. **Run the `em_services` command (no optional parameters).**
   This will add the object class to the MIS.

Note – You can use the Topology Import/Export Tool to save your topology database, and use the `em_ncexport` utility to save your Nerve Center templates. After the platform is restarted, you can import that information back into the MIS.

18

Configuring Communication with CMIP Agents

19

Overview of CMIP Configuration Tasks	*page 19-1*
Preparing the System for CMIP Configuration	*page 19-4*
Compile and Load CMIP Agent Object Types into MIS	*page 19-6*
Starting and Configuring SunLink OSI 8.1	*page 19-6*
Starting and Configuring SunLink CMIP 8.2	*page 19-8*
Starting and Configuring the CMIP MPA	*page 19-9*

This chapter provides detailed instructions for installation and configuration of the components required for managing CMIP agents.

19.1 Overview of CMIP Configuration Tasks

The following list summarizes the activities that you must complete before your system can manage a CMIP Agent.

1. **Prepare your system for CMIP configuration.**
   a. **Define the distribution model.**
   b. **Install all the required products and patches.**
   c. **Gather the configuration information that you will use later.**

2. **Load the CMIP Agent Object Classes into the MIS.**
   The MIS needs to understand the kinds of objects that your CMIP Agent supports. Many standard object classes are delivered with Solstice EM. For any others you must compile the definitions and load them into the MIS.

3. **Start up and configure SunLink OSI 8.1**
   SunLink OSI provides the lower layers of the OSI stack. This handles the data transportation and presentation aspects of communication with a CMIP Agent. This is required if you are using LLC or CONS/X.25.

4. **Start up and configure SunLink CMIP 8.2**
   Sunlink CMIP provides the upper layers of the OSI stack, and uses the services provided by SunLink OSI to communicate with a CMIP Agent.

5. **Start up and configure the EM CMIP MPA.**
   The Solstice EM CMIP MPA translates CMIP requests and responses to and from Solstice Enterprise Manager. It uses the services provided by SunLink CMIP. Before Solstice EM can access the objects in the a CMIP Agent, the CMIP agent must be configured in the MIS. You can use the EM Object Configuration Tool (OCT) to configure CMIP objects. Refer to the “Object Configuration Tool” chapter in the *Solstice Enterprise Manager Reference Manual* for detailed instructions on OCT.

Note – Perform all procedures in this chapter as `root`. All commands assume a `PATH` environment variable that includes `/opt/SUNWconn/bin`. See the *Solstice Enterprise Manager Installation Guide* for instructions on setting your `PATH` environment variable.

The following diagram summarizes the configuration procedure.

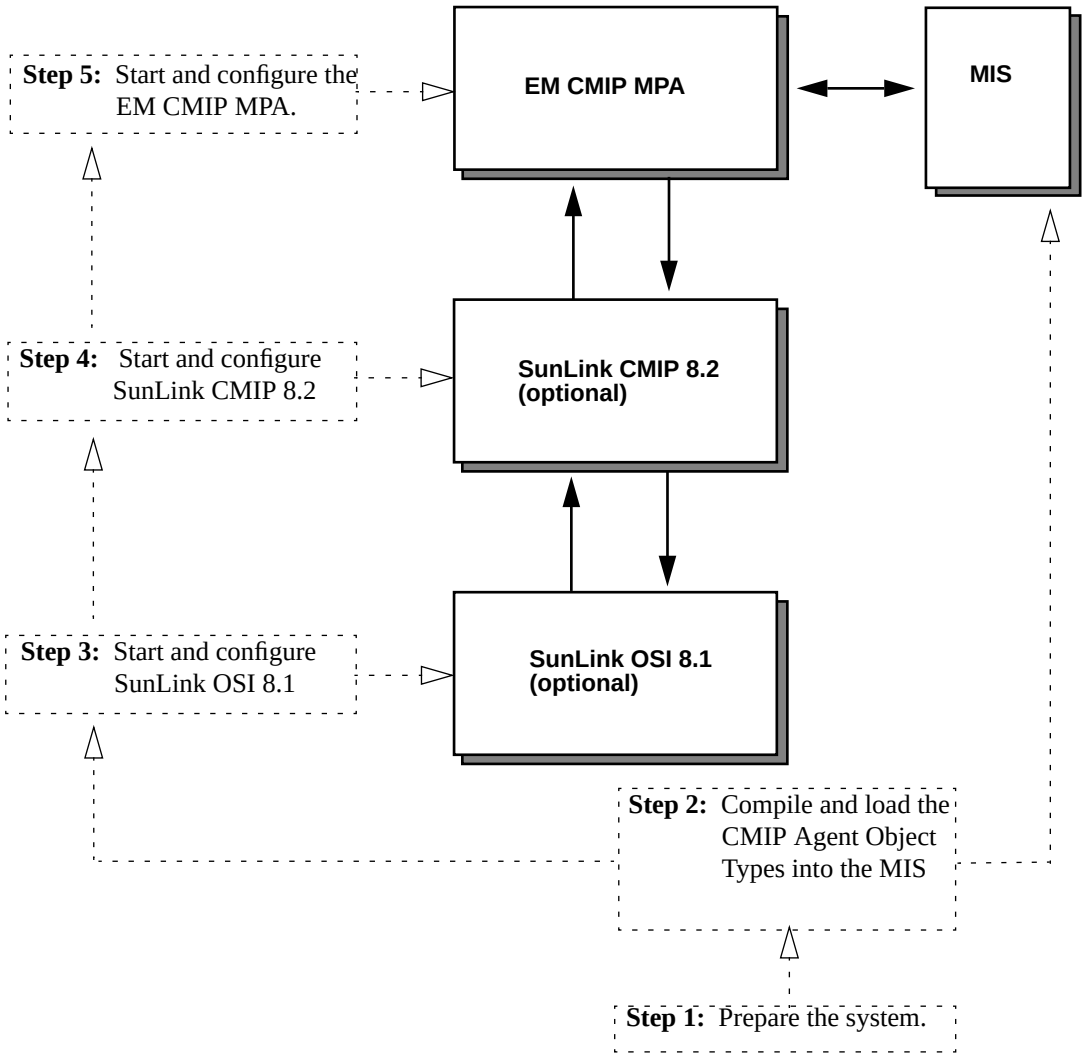


Figure 19-1 Configuring EM for Communication with CMIP Agents

The following sections provide detailed instructions for performing these tasks.

19.2 Preparing the System for CMIP Configuration

Before you begin, there are several steps you must undertake before you can configure the system for communication with CMIP agents. These are:

1. **Determine the distribution model.**
   Before you begin, you must determine the distribution model you will use for CMIP communications.
2. **Install the required SunLink products.**
   Install the following products, as required for your environment:
   a. **SunLink CMIP 8.2 RT or SunLink CMIP 8.2 SDE**
   b. **SunLink OSI 8.1 (optional)**
   c. **SunLink X25 9.0 or above (optional)**
3. **Gather your configuration information.**
   Before proceeding, you should gather all the information that you will need to complete the configuration process.

19.2.1 Determining the Distribution Model

Before you begin, you must determine the overall distribution model you are to use for CMIP communications. For this you must be aware of which MPAs and MISs will be communicating, how many there are, and how they are configured. You will need the following information:

1. **Determine with which MISs the MPA(s) will be communicating.**
2. **Determine how many agents there are, and how they are configured.**
3. **Determine the address(es) of the MPA(s).**
4. **Determine the address(es) of the agents.**

19.2.2 Installing the Required SunLink Products

You can use one of the following configurations of SunLink products for CMIP communications, depending upon your target environment:

- SunLink CMIP 8.2 (if using RFC1006)

- SunLink CMIP 8.2 & SunLink OSI 8.1 (if using CLNP/LLC)
- SunLink OSI 8.1 & SunLink CMIP 8.1 SDE & SunLink X25 9.x (if using CONS/X.25)

Some of these products may require a patch. The appropriate patches for your installation should be on hand before proceeding. Patches can be obtained from your normal source or Sun point of contact. Refer to your installation documentation for these products for patch information.

You should now install the lowest layer of the protocol stack first, as follows:

1. **SunLink OSI 8.1. (Optional)**
   First you should install SunLink OSI 8.1. See the provided documentation for details on installation. A typical installation for OSI 8.1 will include the following product packages:

   - SUNWcorpc
   - SUNWcosia
   - SUNWcosib
   - SUNWcosic
   - SUNWcosid
   - SUNWlicsw
   - SUNWlit

2. **SunLink CMIP 8.2 (Required)**
   Install either SunLink CMIP 8.2 RT or SunLink CMIP 8.1 SDE. See the provided documentation for details on installation. A typical installation for SunLink CMIP 8.2 RT will include the following product packages:

   - SUNWomgta
   - SUNWomgtb
   - SUNWomgtc
   - SUNWrk6 (if RFC1006 is used)
   - SUNWlicsw
   - SUNWlit

   The License Installation Tool package, SUNWlit, must be installed even though you will have already installed a version for SunLink OSI 8.1.

3. **SunLink X.25 9.0. (Optional)**
   You will require to install SunLink X.25 if you are going to communicate with a CMIP Agent over X.25. See the provided documentation for details on installation. Once you have installed SunLink X.25 you should also install any required SunLink X.25 9.*x* patches.

19.2.3 Gathering Your Configuration Information

The following checklist summarizes the information you will need for configuring Solstice EM for communication with CMIP agents:

- Presentation Selector for CMIP Agent
- Session Selector for CMIP Agent
- Transport Selector for CMIP Agent
- Whether underlying communication with CMIP Agents is via TCP/IP(RFC1006), CLNP/LLC or X.25 (CONS).
- The Network Service Access Point Address: IP Address of CMIP Agent (RFC1006) or OSI Network Address (CLNP/CONS)
- Name of object directly contained by Root in the Management Information Tree (MIT) of the CMIP Agent.
- ASN.1 and GDMO descriptions of the objects supported by the CMIP Agent.

19.3 Compile and Load CMIP Agent Object Types into MIS

The MIS must understand the kinds of objects that your CMIP Agent supports before it can access the objects maintained in the CMIP Agent. Many standard object types are delivered with Solstice EM. For those of which the MIS is unaware, you must compile and load the appropriate CMIP Agent ASN.1 and GDMO definitions into the Solstice EM MIS. You can do this by using the compilers `em_asn1` and `em_gdmo`, supplied with Solstice EM. Refer to Chapter 16, “Adding a Managed Object Class to the MIS,” for instructions on using these compilers.

19.4 Starting and Configuring SunLink OSI 8.1

To configure SunLink OSI to communicate with CMIP agents, do the following:

Note – If you will be using RFC1006, you need not set up the Network Layer Address.

1. **Halt the CMIP MPA.**
   Halt the Solstice EM CMIP MPA by entering the following command:

```
host# /etc/rc2.d/S98cmipmpa stop
```

2. **Halt the CMIP stack.**
   Halt the CMIP stack by entering the following command:

```
host# osistop osimcs
```

3. **Start the SunLink OSI stack.**
   If SunLink OSI is not running, start the SunLink OSI stack by entering the following command:

```
host# /etc/rc2.d/S90osinet start
```

4. **Run** `ositool`**.**
   Your distribution of SunLink OSI 8.1 provides a tool for configuring SunLink OSI, called `ositool`. Use this tool to configure the Network Layer Address, OSI routing, and Application Selectors to successfully communicate with the Agent.

   You can run `ositool` by entering the following command:

```
host# ositool &
```

   For detailed instructions on using this tool, consult the *SunLink OSI 8.1 Communication Platform Administrator's Guide.*

5. **Restart** `osinetd`**.**
   After you have entered all of your configuration information, use `ositool` to restart `osinetd`.

19.5 Starting and Configuring SunLink CMIP 8.2

To configure SunLink CMIP 8.2 to communicate with CMIP agents, do the following:

1. **Ensure the OSI stack and CMIP stack are running.**
   If the OSI stack and CMIP are not running, start them.

2. **Run** `cmiptool`**.**
   Issue the `cmiptool` command at the operating system prompt:

```
host# cmiptool &
```

   The `cmiptool` main screen then appears.

3. **Enter the type of Subnetwork.**
   Go to the section entitled “Default XMP Address.” Select the subnetwork that you are using. Depending on the protocol used to communicate with the CMIP Agents, click on one of the buttons as follows:

   - CONS(X.25), if you are using X.25.
   - TCP-IP(RFC1006), if you are using TCP/IP.
   - CLNP(LLC1) If you are using Ethernet.

4. **Enter the value for the Request Timer (optional).**
   The Request Timer in the CMIP/MCS Parameters section specifies the maximum time allowed for requests to extract information from agents. By default, SunLink CMIP has a timeout parameter value of 5 which indicates a 50 second timeout. The parameter value is then multiplied times 10 to calculate the actual length of the timeout in seconds.

   If you intend to issue requests to CMIP Agents which require a longer timeout, you should increase the value of this parameter. It is recommended that you choose a much larger value. The maximum allowable Timeout value is 127, which equals 1270 seconds.

5. **Select the Apply button.**

6. **Exit** `cmiptool`**.**

19.6 Starting and Configuring the CMIP MPA

A CMIP MPA is a Solstice EM component that provides access to CMIP Agents and Managers. The CMIP MPA receives management directives from the MIS and translates the directives into proper CMIP messages. In other words, the CMIP MPA is the CMIP Proxy Agent for the MIS. The CMIP MPA can also act as a CMIP Agent allowing the objects in the MIS to be managed by a CMIP Manager.

The EM CMIP MPA performs association management. When a previously established association goes down, a `communicationsAlarm` is generated by the EM CMIP MPA and sent to the MIS. When the connection is re-established, the alarm is automatically cleared by the MPA.

A CMIP MPA must be configured prior to attempting to access managed objects over CMIP. A CMIP MPA can be located on the same machine where the MIS is located (the default) or on a remote machine.

To configure a CMIP MPA for communication over CMIP, do the following:

1. **Stop all running CMIP MPAs.**
   As only one CMIP MPA can be run on a single machine, enter the following command to stop all running CMIP MPAs:

```
host# /etc/rc2.d/S98cmipmpa stop
```

2. **Set the $EM_MIS_DEFAULT environment variable.**
   If you are using csh as your shell, set the EM_MIS_DEFAULT environment variable as follows:

```
host# setenv EM_MIS_DEFAULT -host <hostname>
```

   Where:

   <*hostname*> is the name of the host on which the default MIS is running.

   If you are using sh or ksh as your shell, set the environment variable as follows:

```
host# EM_MIS_DEFAULT=<hostname>
host# export EM_MIS_DEFAULT
```

3. **Start the CMIP MPA.**
   To start the CMIP MPA, enter the following command at the operating system prompt

```
host# /etc/rc2.d/S98cmipmpa start
```

4. **Bring up the Object Configuration Tool (`em_oct`).**
   The Object Configuration Tool (OCT) enables you to configure objects managed under EM. To use OCT to configure a CMIP agent, start OCT from the command line as follows:

```
hostname% em_oct -cmip [options] &
```

The optional parameters are shown in Table 19-1:

Table 19-1 `em_oct` Parameters

Option	Description
`-help`	Print list of options (with descriptions) for the `em_oct` command.
`-host` <*hostname*>	Specify the <*hostname*> of a remote MIS.
`-cmip`	Configure a CMIP object. Replaces `em_cmipconfig`.
`-id` <*id*>`...`	Specify topology IDs. Multiple IDs are delimited by a space.
`-link` <*id1*> <*id2*>	Create a link between <*id1*> and <*id2*>.
`-mis`	Configure an MIS object.
`-name` <*name*>`...`	Specify the name of an object. Multiple names are delimited by a space.
`-parent` <*parent_id*>	Specify the parent of the object you want to create.
`-rpc`	Configure an RPC object.
`-snmp`	Configure an SNMP object.
`-type` <*type*>	Specify the topology type of the object you want to create.

When you invoke the `em_oct` command with the `-cmip` option, the OCT main window appears as illustrated in Figure 19-2:

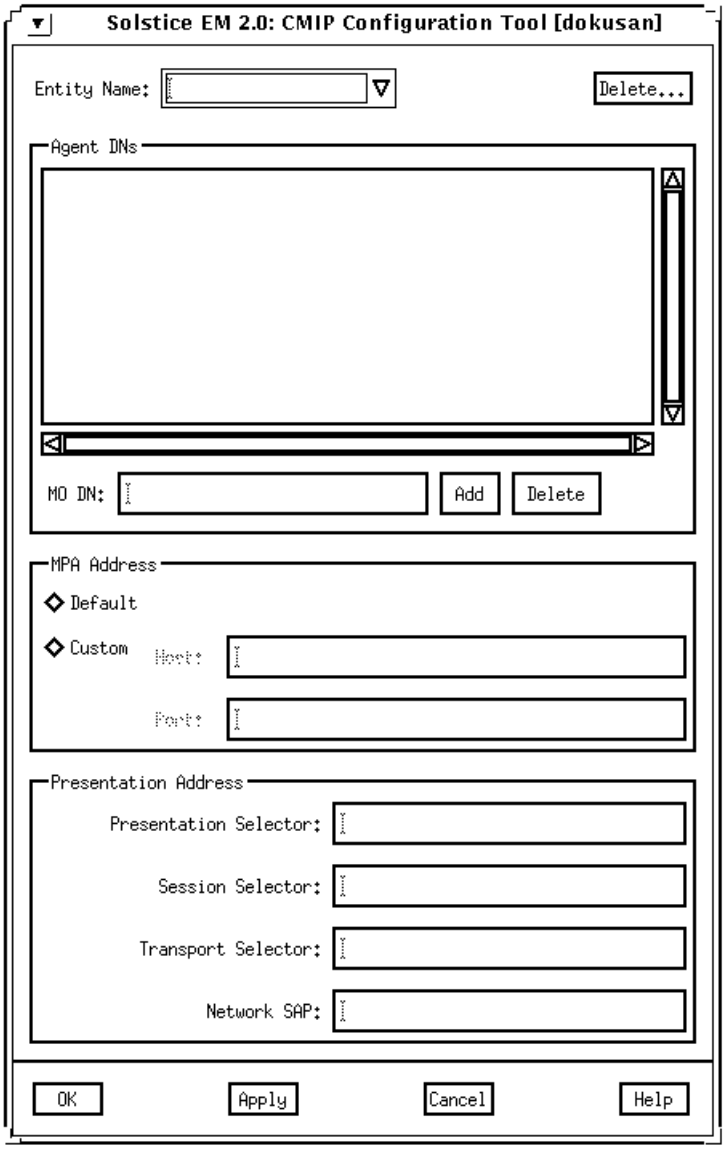


Figure 19-2 OCT CMIP Configuration Tool Window

For detailed instructions on using OCT, refer to the “Object Configuration Tool” chapter in the *Solstice Enterprise Manager Reference Manual.*

Enter the appropriate information into the following fields in the OCT CMIP Configuration window:

Entity Name

Specify the name of the remote MIS with which you want to communicate. The value can be a string, object identifier (OID), or distinguished name (DN). Click on the down arrow to see a list of known MISs. To delete an MIS, click on the Delete button, select an MIS from the resulting list, then click on the OK button. After specifying an MIS, all the other fields in this window for which information exists in the MIS are filled.

Agents DNs

This field displays the list of distinguished names (DN) of objects that the agent manages. When configuring a CMIP agent for a particular topology node, you must select an Agent DN (from the list) by which that topology node is managed.

MO DN

Specify the DN (top-most node) of the MIT to be managed by the remote MIS. For example, if the remote MIS is located on a machine called `poignant`, enter `/systemId=name:"poignant"` in the MO DN field and click on the Add button. `/systemId=name:"poignant"` will appear in the Agents DNs field. To delete a DN, select the one you want to delete and click on the Delete button.

MPA Addresses

Select the Default toggle button to apply the default values for the MPA Host and MPA Port. The default host is <*localhost*>, and the default port number is 5557. To customize these values, select the Custom toggle button and enter the MPA host and port number.

Presentation Address

For MIS-MIS awareness, you must enter the appropriate Presentation Selector, Session Selector, Transport Selector, and Network SAP for the agent you are configuring.

Click on the Apply button to create the object, or click on the OK button to create the object and dismiss the CMIP Configuration window.

Following are sample Presentation Address values when using CLNS(LLC1)/CONS(X.25):

```
Presentation Selector: 4444
Session Selector: 3333
Transport Selector: 3007
Network SAP: 47000400060001080020l1e7f001
```

Following are sample Presentation Address values when using TCP-IP (RFC1006):

```
Presentation Selector: dflt
Session Selector: Prs
Transport Selector:CMIP
Network SAP: 81924b94
```

Note that the Network SAP in this case is the value of the IP address of the CMIP Agent represented in hexadecimal.

19.6.1 Configuring Multiple MPAs on One System

The following is an example of configuring multiple MPAs on a single system:

1. **Set the PSEL, SSEL, and TSEL environment variables.**
   Use the `setenv` command to set the PSEL, SSEL, and TSEL environment variables:

```
host# setenv PSEL rfc0
host# setenv SSEL Prs
host# setenv TSEL CMIP
host# setenv NSAP 0xIPlocalhostinHex
```

   If you want to specify the AE-TITLE for the MPA, set the APNAME environment variable as well:

```
host# setenv APNAME "<AE-title-OID>:<AE-qualifier>:\
     <AP-invoke-id>:<AE-invoke-id>"
```

2. **Start MPA #1 for communication with agent #1.**
   Next, start the MPA #1 for communication with Agent #1. Enter the start command at the command line, as shown in this example:

```
host# /etc/rc2.d/S98cmipmpa start
```

   This mpa is bound to `PAddr(rfc0,Prs,CMIP,0xIPlocalhostinHex)` and uses the default MPA port 5557.

3. **Set up MPA #2 for communication with Agent #2:**

```
host# setenv PSEL rfc1
host# setenv SSEL Prs
host# setenv TSEL CMIP
host# setenv NSAP 0xIPinHex
host# setenv EM_CMIP_DEFAULT_PORT 5558
```

4. **Start MPA #2:**

```
host# /etc/rc2.d/S98cmipmpa start
```

In this example, MPA #2 is bound to:

```
PAddr(rfc1,Prs,CMIP,0xIPlocalhostinHex)
```

and uses the non-default MPA port of 5558.

5. **Configure Agent(1) and (2) using e**m_oct -cmip**.**
Whereas Agent(1) uses the default MPA to talk to Agent #1, Agent #2 will use the custom MPA port to talk to Agent #2.

The configuration for Agent #1 would therefore be as follows:

```
Psel=dflt
Ssel = Prs
Tsel= CMIP
NSAP=0xIPAgent1addinHex
    'MDN=/systemId=name:"<system_name>"'
```

The configuration for Agent #2 would be as follows:

```
 Psel=gom
Ssel = ses
Tsel=
NSAP=0xIPAgent2addinHex
Custom MPA hostname="<hostname>"
MPA Port=5558,
    'MDN=/networkId=pString:"network"'
```

As seen in above examples, both MPA's are running on same system and MIS forwards requests to Agent1 and Agent2 using MPA1 and MPA2 respectively.

Index

Symbols

$eventInfo, 13-9
$eventOI
  definition of, 13-9
$eventType variable, 13-10
$eventTypes
  that MIS knows by default, 13-11
$messType
  checking after MSet, 14-24
  checking after SendAction, 14-22
  possible values of, 13-13
$messType system variable, 13-12
$pollfdn
  resetting the value of, 13-8
  use of Append_rdn to set, 14-9
$pollfdn system variable in RCL, 9-14
$pollfdn variable, 13-7
$pollFdnSet, 13-6
  use in Nerve Center requests, 9-13

A

access to managed resources via CMIP, 19-9
accessing SNM agents, 6-13
action
  definition of, 9-3
  deleting, 12-15
  three types in request templates, 12-2
actions
  adding to a transition, 12-13
actions available for request templates, 12-13
Adding a GDMO object class automatically
  editing em_startup, 16-2
adding objects to the MIS
  how to, 2-1
additionalText
  specifying in alarmStr function, 14-7
additionalText information, 8-9
AdminOperStatusUp sample template, 3-25
agent address, 8-5
Agent Mapping
  Selecting SNM RPC agents, 2-6
agentVersion number
  how to find, 14-27
alarm function, 14-3
Alarm Service
  and alarm-logging by requests, 9-15
  what it does, 3-3
Alarm Service perceived severity, SNMP traps, 8-7

Alarm Service, perceived severity, 5-8
alarms
logging via RCL SendEvent function, 14-22
alarms, SNM, 5-7
ampersand
use of in RCL, 13-14
appendRdn
use of, 13-8
appendRdn function
syntax of, 14-9
ASN.1 compiler
use in adding SNM-based object classes, 18-3
use of, 16-1
ASN.1 values
converting to strings in RCL, 14-13
Asn1Type
use in RCL, 13-5
associativity
of operators in RCL, 13-18
asynchronous event reports (traps), 7-8
attribute
definition of, 9-3
attributes
referenced in RCL conditions, 13-13

B

bitwise operators
use of in RCL, 13-17
bridge, creating in Viewer, 2-10
bus, creating in viewer, 2-10

C

CC
filtering capabilities, 5-3
forwarding information, 5-1
glyph traps, 5-2
installing packages, 5-5
periphery-to-center configuration, 5-1
Receiver application, 5-4
receiving station, definition, 5-4
Sender daemon, 5-5
Sender daemon, remote, 5-7
sending station, 5-5
SNM database traps, 5-2
SNM events, 5-2
SNM glyph traps, 5-2
SNMP Traps, 5-2
CC and SNM, 5-1
CC event types, 5-4
Circle
creating in Viewer, 2-10
clear alarms
generated by glyph reset on SNM Console, 5-2
clearing alarms
by Nerve Center requests, 9-16
CMIP
changing timeout values, 19-8
configuring, 19-8
CMIP agents
use in device management, 3-8
CMIP messages, 19-9
CMIP MPA, 19-9
configuration, 19-9
configurations, 1-20
CMIP MPA, versions of, 19-9
CMIP Proxy Agent, 19-9
CMIP support
in Solstice EM, 1-19
CMIS filters
use in RCL event subscriptions, 14-33
color to severity mapping, 12-18
how to change, 12-18
colors
mapped to severities, 12-18
Concise MIB compiler
loading Sun-defined objects into MIS, 17-1
condition
definition of, 9-3, 12-2, 13-1
role as an action, 12-2
role in defining transition, 12-2

set supplied with Solstice EM, 12-17
Condition Language, 13-1
assignment operators, 13-16
Constants, 13-3
operator
assignment, 13-16
operator symbols, 13-15
Operators, 13-15
System variables, 13-6
value check functions
defined, 14-13
Variables, 13-3
Condition language
See Request Condition Language (RCL), 13-1
conditions
evaluating in request, 9-11
two functions in requests, 9-3
two roles of, 13-2
Configuration Tool, 5-5
configuring
SunLink CMIP, 19-8
container object, creating in Viewer, 2-10
containment relationship, 14-9
converting SNM schema files, 6-14
Cooperative Consoles
operation, 5-7
setting up for EM, 5-5
using with EM, 5-1
correlation
of information in requests, 3-24
creating
container object in Viewer, 2-10
element object in Viewer, 2-10
multimonitor object in Viewer, 2-10
objects in Viewer, 2-9
Creating a View
an example, 2-7
custom mapping of SNMP traps, 8-11
customizing trap mapping, 8-15

D

data types
in SNM event requests, 14-28
Data Viewer
use of, 3-26
database, SNM
loading into EM, 6-9
debugging
via RCL Print() function, 14-21
debugging request templates
RCL Print function, 14-21
deleting a transition, 12-15
deleting an action, 12-15
device, creating in Viewer, 2-10
DeviceReachablePing sample template, 3-26
Discover
configuration options, 2-4
creating views with, 2-4
how to configure, 2-6
how to invoke, 2-2
discover
default properties, 2-4
Discovering routers
an example, 2-6
discriminator constructs
define event logging, 3-4
DMI standard for event notifications, 15-1
Domain Manager
See SunNet Manager, 5-1

E

element.schemas, 6-7
em_cmib2gdmo
error messages, 17-2
em_cmipconfig
replaced by -cmip option of em_oct, 19-11
em_debug
use with RCL print() function, 14-21
em_debug utility, 14-21

em_ncexport utility
options, 12-22
em_ncimport utility
options, 12-22
em_oct command
optional parameters, 19-11
em_snm_type_import utility
syntax of, 6-7
em_snmdb_import utility, 6-9
em_snmp-trap, 8-7
em_snmp-trap daemon, 8-1
em_trapd, 8-4
enterprise field, SNMP trap, 8-5
enterprise mapping blocks, 8-12
enterprise-object-identifier, 8-13
enterpriseSpecific trap, 8-9
environment variable
EM_SERVER, xxvii, 6-9, 6-12
LD_LIBRARY_PATH, 6-2, 6-8, 6-11
Event Dispatcher (na.event)
role in CC, 5-7
event information
two types, 3-5
event log mapping, table, 15-2
event logging, 9-12
event notification
as handled by requests, 9-7
event notification, default type, 8-10
event notifications
adding custom definitions, 15-2
definition, 15-1
DMI standard, 15-1
use in fault management, 3-8
Event requests, SNM, 3-26
event subscription
RCL statements for, 9-8
via RCL functions, 14-32
what it is, 3-12
event types
known to MIS by default, 13-11
event types for trap mapping, 8-15
event types,CC, 5-4
events
extracting attributes of in RCL, 14-14
logging via RCL SendEvent function, 14-22
events generation, 15-5

F

fault management
definition of, 3-2
fault status
color-coding of, 3-2
monitoring, 3-2
procedure for monitoring, 3-17
fault status indication
in SunNet Manager, 5-8
FDNs
and Nerve Center requests, 9-13
filtering, CC, 5-3
Find window
use in creating a view, 2-7
forwarding an SNMP trap in a request template, 9-12
forwarding SNMP traps, 8-2
fully distinguished name (FDN), 14-9

G

GDMO and ASN.1, compiling, 15-3
GDMO compiler
use of, 16-2
GDMO description
determines names of attributes, 13-13
GDMO document, 14-36
GDMO documents
event logs, 15-2
generating events, 15-5
generic trap values, 8-8
generic trap, SNMP, 8-5
graphical display
for Request Designer, 12-20
maximum number of states displayed for Request Designer, 12-21

maximum number of states in Request Designer .., 12-9
Guidelines for the Definition of Managed Objects (GDMO), 1-22

H

Hexagon
creating in Viewer, 2-10
Hexagon 120
creating in Viewer, 2-10
host, creating in Viewer, 2-10
hub, creating in Viewer, 2-10

I

icons
converting SNM glyphs to EM format, 6-6
identifier field, traps, 8-10
IF construct
example of in condition, 14-25
IF constructs
use in RCL conditions, 14-22
use of in RCL, 13-18
IF ELSE constructs
nesting of in RCL, 13-22
use of in RCL, 13-19
IIMC standard event notifications, 8-14
IIMC standard for event notification, 8-11
incoming SNMP traps, 8-11
installing CC, 5-5
interface
creating in Viewer, 2-10
InternetActionInfo, 13-5
ISO X.722 standard event notifications, 8-14

L

LD_LIBRARY_PATH
setting for SNM API access, 6-6
libnetmgmt library, 6-11
libnetmgt.so library, 6-2
libnetmgt_db.so compatibility library, 6-2
libnetmgt_db.so library, 6-2
link
creating in Viewer, 2-10
Log Manager
verify event types, 8-16
Log Manager, editing, 15-4
log record entries, adding, 15-4
logging alarms
from Nerve Center requests, 9-15
logging events, editing, 15-4
logical view
creation of, 2-7
logs, creating
example of, 3-13

M

M-ACTION
sent by RCL SendAction function, 14-21
managed object class
adding to MIS, 16-1
managed objects, attributes of
accessing in RCL, 14-14
Management Protocol Adaptors (MPAs), 1-21
manager-to-manager capability, 7-11
managing devices
request templates for, 3-20
using RPC agents, 4-4
mapping blocks, 8-14
mapping records, 8-13
mapping SNM for perceived severity, 5-8
MetaData Repository (MDR), 13-14
MIBs
adding to MIS, 1-22
MIS
adding a managed object class to, 16-1
MIT, 14-19
Monitor
ability to generate alarms, 3-6

use in tracking device availability, 3-6
Monitor function
what it does, 2-11
Monitor Properties window, 2-12
monitoring fault status, 3-19
multimonitor, creating in Viewer, 2-10

N

na.event
role in SNM event requests, 4-3
na.snmp.schemas, 6-11
na.snmp-trap, 8-2
Nerve Center
definition of, 9-1, 9-4
Message types, 13-12
related terminology, 9-3
Request Condition Language, 13-1
nerveCenterAlarm
structure of, 13-9
nerveCenterAlarms
logged by RCL functions, 14-7
network
creating in Viewer, 2-10
network management
role of requests in, 3-17
steps in performing, 3-17
notification
definition of, 9-4

O

Object Configuration Tool
role of, 2-15
object creation events
subscribing for in RCL, 14-36
object instance
creating, 2-11
how to create, 2-9
Object Palette, 2-11
using to create object instance, 2-9
objects
methods for adding to MIS, 2-1
OCT window
information that should be specified, 2-10
OmniSector
creating in Viewer, 2-10
OR
use of in RCL conditions, 13-17
OSI stack
use with CMIP MPA, 1-20

P

PC, creating in Viewer, 2-10
perceived severity values, SNMP trap, 8-7
perceivedSeverity
in snmAlarmTraps, 5-8
permissible values of, 13-10
periphery-to-center configuration
using Cooperative Consoles, 5-1
poll rate
creating a new, 12-17
definition of (for requests), 9-4
poll responses to a request, 9-7
polling, 3-6
definition of (for requests), 9-4
offloading to RPC agents, 4-3
polling for system availability, 3-21
populate the MIS
how to, 2-1
precedence
of operators in RCL, 13-18
print
as RCL function, 14-21
Printer, creating in Viewer, 2-10
priority
in SNM, 5-8
probable cause values, 8-9
probableCause
and clearing of alarms, 9-16
protocol operations, 7-11
protocols, network management
support for, 1-11
proxy agents, 6-14

Proxy host
entering a proxy host, 2-6

Q

Q3 connection, 1-21

R

reachability
using SNM event requests to check for, 3-26
Receiver application, 5-4
receiving SNM alarms, 5-7
receiving station
definition, 5-4
relational operators
in SNM event requests, 14-27
Remote Procedure Call (RPC)
support in Solstice EM, 1-11
Remote Procedure Call (RPC) protocol
See RPC, 4-2
request
definition of (for Nerve Center), 9-4, 10-1, 12-1
evaluating conditions in, 9-11
poll responses to, 9-8
polling and event subscription, 9-6
scope of variables in, 9-10
use of variables and attributes in, 9-9
Request Condition Language
attributes, 13-2
capabilities of, 9-1
constants, 13-2
definition of, 9-5
operators
precedence of, 13-18
valid severities, 14-4
variables, 13-2
Request Condition Language (RCL)
AddressStrToAddress function, 14-3
alarm function, 14-3
alarm logging functions
Send_Event, 14-22
alarmOi function, 14-6
alarmStr function, 14-7
anystr function, 14-8
appendRdn function, 14-9
arithmetic operators, 13-16
ASN.1 conversion functions
sasnToStr, 14-12
StrToAsn, 14-30
asnToStr function, 14-13
attributes in, 13-16
bitwise operators, 13-17
built-in functions, 14-1
components of, 13-1
debugging function
print, 14-21
defined function, 14-13
event handling functions
SendAction, 14-21
SendTrap, 14-23
Subscribe, 14-32
SubscribeFilter, 14-33
SubscribeOi, 14-36
TrapGenericType, 14-37
TrapSpecificType, 14-38
Unsubscribe, 14-40
Extract function, 14-14
FinalStr function, 14-15
FOREACH constructs, 13-21
GetTimeStamp function, 14-16
IF constructs, 13-18
IF ELSE in, 13-19
include function, 14-16
InitialStr function, 14-17
IsChoice function, 14-17
IsList function, 14-18
logical operators, 13-16
examples, 13-17
Mail function, 14-18
minus sign, use of, 13-16
MSet function, 14-24
name conversion functions
Append_rdn, 14-9
NameToAddress, 14-19
NameToOid, 14-19
NameToAddress function, 14-19
NameToOid function, 14-19
nesting of constructs, 13-22

OiNameToOi function, 14-20
OiToOiName function, 14-20
operators
and timestamp arithmetic, 13-23
arithmetic, 13-16
equality, 13-16
logical, 13-16
relational, 13-16
operators in, 13-15
precedence of operators, 13-18
Print function, 14-21
relational operators, 13-16
SendAction function, 14-21
SendEvent function, 14-22
SendTrap function, 14-23
SnmEventRequest function, 14-25
SnmKillRequest function, 14-30
statement blocks, 13-19
StrCat function, 14-31
string handling functions
AnyStr, 14-8
FinalStr, 14-15
InitialStr, 14-17
StrCat, 14-31
StrToAsn function, 14-30
Subscribe function, 14-32
SubscribeFilter function, 14-33
SubscribeOi function, 14-36
summary of functions, 14-1
syntax checking, 13-24
syntax of attribute names in, 13-14
system variables, 13-6
timestamp arithmetic
operators, 13-23
TrapGenericType function, 14-37
TrapSpecificType function, 14-38
type checking, 13-24
types of operands in, 13-2
Undefine function, 14-38
UNIXCMD function, 14-38, 14-39
Unsubscribe function, 14-40
use of ampersand in, 13-16
value check functions
Undefine, 14-38
variables, dynamic typing of, 13-5
WHILE constructs in, 13-20
Request Condition Languate (RCL)
NumElements function, 14-19
Request Designer, 12-1
attribute names, 9-10
creating a new template, 12-5
graphical and text modes, 12-9
graphical display, 12-18
performing tasks using, 12-21
how to start, 12-3
log object, 9-12
logging an event, 9-12
notification, 9-8
notification arrival, 9-11
poll
conditions, 9-8
rate, 9-9
read-only text display area, 12-4
request
attribute name, 9-9
notification arrival, 9-11
poll response arrival, 9-11
Request Condition Language, 13-1
request notification
response to a poll, 9-7
request template, 9-9
starting, 12-3
Transitions window, 12-10
variable name, 9-10
request template
definition of, 9-5
forwarding a trap in, 9-12
high-level procedure for creation of, 10-16
logging an event, 9-12
procedure for creating in graphical display, 12-20
sample explained, 10-8
system and user variables, 9-9
request templates
changing order of transitions, 12-11
exporting to ASCII file, 12-6, 12-22
for RPC agents, 4-7
importing from ASCII file, 12-22
importing previously exported templates, 12-6

opening, deleting, and saving, 12-5
shipped with EM, 3-20
request templates, incomplete
saving via Export Current, 12-7
request terminology, 9-3
request-related applications, 9-2
requests
how target device is set, 9-13
how to launch, 3-21
Retix agents
configuring SunLink CMIP 8.1 for, 19-8
retry-interval for SNMP proxy, 7-7
router interfaces
checking via SNMP Browser, 3-28
router, creating in Viewer, 2-10
routers
finding .. on network, 2-6
polling for status of interfaces, 3-25
RPC agents
building request templates for, 4-7
direct polling of, 4-2
snapshot of in Data Viewer, 3-26
type of machine supported on, 4-5
using, 4-1
using Discover to configure support for, 4-7
RPC Protocol Driver Module, 6-2

S

sash
in Request Designer's Conditions window, 12-16
schema files
relationship among .. shipped with SNM, 7-4
schemas
See SNM schemas, 4-5
SNMP, 7-3
security, 7-11
Sender daemon, 5-5
filters for information forwarding, 5-4
sending station
definition, 5-5
server
creating in Viewer, 2-10
severities
color-coding of, 3-3, 8-8
in alarm logging functions, 14-4
role of in Request Designer, 12-10
severity
definition of, 9-5
See perceivedSeverity, 5-8
two meanings in Request Designer, 9-5
what it is, 3-2
severity assignments, changing, 8-8
severity in Request Designer
definition of, 12-18
short-circuiting
not implemented in RCL, 13-16
Simple Network Management Protocol (SNMP), 2-4
SNM
See also SunNet Manager, 14-26
SNM agent and schema files, 6-11
SNM and SNMP traps, 8-2
SNM API
use by Cooperative Consoles, 5-4
SNM application
definition of, 6-1
requirement for EM compatibility, 6-5
setting LD_LIBRARY_PATH for, 6-8
SNM applications
adding to EM, 6-6
compatibility with EM, 6-4
SNM configuration files, 6-11
default locations, 6-11
SNM console fault indications, table, 5-8
SNM database
importing into EM, 6-9
SNM database traps, 5-2
SNM element types

adding third-party SNM types to EM, 6-7
SNM elements
converting schemas for third-party elements, 6-7
SNM event requests, 4-3
example of, 3-26
SNM events, 5-2
SNM fault status
mapping to perceivedSeverity, 3-17, 5-8
SNM glyph reset
generates clear alarm via CC, 5-2
generating clear alarms on EM, 5-9
SNM glyph traps, 5-2
SNM glyphs
converting third-party glyphs to EM format, 6-6
SNM schema file
coverting to GDMO document, 18-1
SNM schemas
adding new object classes based on, 4-5
snm.conf file, 7-4, 7-6, 7-7
snm2gdmo compiler, 6-14
snmAlarmTraps
how perceivedSeverity is determined, 5-8
SNMP, 2-4
request templates
shipped with EM, 3-20
SendTrap condition, 9-12
trap daemon translation, 8-2
SNMP attributes
retrieving values of in SNMP Browser, 3-28
SNMP Browser
invoking, 3-28
use of, 3-28
SNMP daemon
polling for availability, 3-21
SNMP proxy
receiving responses, 7-7
retry-interval, 7-7
SNMP schemas, 7-3
SNMP support
in Solstice EM, 1-15
SNMP trap
additional text information, 8-9
agent address, 8-5
changing severity, 8-8
custom mapping, 8-11
customizing trap mapping, 8-15
default event notification, 8-10
default trap-mapping, 8-6
enterprise field, 8-5
enterprise mapping blocks, 8-12
generic trap, 8-5
generic values, 8-8
identifier field, 8-10
incoming, 8-11
mapping blocks, 8-14
mapping file location, 8-16
mapping records, 8-13
mapping records format, 8-17
probable cause values, 8-9
specific trap, 8-5
specifying source of alarm, 8-7
time stamp, 8-5
user-configurable capability, 8-11
variable bindings, 8-5
SNMP trap daemon
starting, 8-3
SNMP trap message type, 8-5
SNMP trap severity values, 8-7
SNMP trap types, table, 8-6
SNMP trap_maps file, 8-4
SNMP traps
and Cooperative Consoles, 5-2
extracting generic type in RCL, 14-37
extracting specific type in RCL, 14-38
generic types of, 14-37
monitoring with Nerve Center requests, 3-12
operation, 8-1
structure, 8-4
use in device management, 3-9

SNMP traps, forwarding, 8-2
snmp.schema, 7-3
snmp-mibII.schema, 7-3
SnmpPingBackoffReachable sample template, 3-25
SNMPv2
  files, 7-13
  SMI, 7-11
  translation program, 7-13
Solaris x86
  RPC agents for, 4-5
Solstice EM-specific event notifications, 8-14
specific trap, SNMP, 8-5
specifying source of alarm, 8-7
starting/stopping the trap daemon, 8-4
state
  definition of (for requests), 9-5
statement blocks
  in Request Condition Language, 13-19
States window
  in Request Designer, 12-9
subnetwork
  creating in Viewer, 2-10
subscription
  use in requests, 3-12
SUN Workstation, creating in Viewer, 2-10
SunLink
  configuring CMIP, 19-8
SunLink CMIP 8.2, 1-20
SunNet Manager
  forwarding information to EM, 5-1
  glyph reset traps, 5-2
  topology traps, 5-2
SunNet Manager alarms
  receiving via Cooperative Consoles, 5-7
SunNet Manager event priorities
  mapping to perceivedSeverity, 14-29
SunNet Manager event requests
  data types for thresholds, 14-28
  initiating via RCL, 14-25
  relations for thresholds, 14-27
  specifying thresholds in, 14-26
  structure of, 14-26
SunNet Manager schemas
  adding object classes based on, 18-1
system variable
  $eventTime, 9-10
  $eventType, 9-12
  $messageType, 9-10
system variables
  $eventInfo, 9-12
sysUpTime
  polling for the current value of, 3-24
  use in device management, 3-24

T

TCP/IP network
  use of RPC agents within, 4-1
Telecommunications Management Network (TMN), 1-21
templates, request
  shipped with EM, 3-20
time stamp, SNMP trap, 8-5
time, on MIS host
  retrieving in RCL, 14-16
timeout values
  changing, 19-8
transition
  definition of (for requests), 9-6
  deleting, 12-15
trap mapping formats, 8-17
trap mapping, event types, 8-15
trap mapping, file location, 8-16
trap variable bindings, 8-5
trap_maps file, 8-4, 8-10, 8-11
trap-mapping, default, 8-6
traps, SNMP
  see SNMP traps, 3-9
TrapSpecificType, 13-5
type of SNMP message, 8-5

U

universe
creating in Viewer, 2-10
user variables, 9-10
user-configurable trap-mapping, 8-11

V

variable
definition of (for requests), 9-6
variables
declaring in RCL, 9-10
Viewer
creating object instances in, 2-9
how color to severity mapping is set, 12-18
Object menu, 2-9
Viewer Edit menu, to create an object, 2-9
views
creating a Routers view, 2-7

W

WHILE loops in RCL
example of, 13-21