



Customizing Guide

Solstice Enterprise Manager™ 4.1

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-7967-10
October 2001, Revision A

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Solstice, Solstice Enterprise Manager, SunDocs, SunExpress, SunOS, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Solstice, Solstice Enterprise Manager, SunDocs, SunExpress, SunOS, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Please
Recycle



Adobe PostScript

Contents

Preface xxv

Part I Overview

- 1. Introducing *Solstice Enterprise Manager* 1-1**
 - 1.1 What is Solstice EM? 1-1
 - 1.1.1 What Can You Manage With Solstice EM? 1-2
 - 1.1.2 Who Uses Solstice EM? 1-3
 - 1.2 Solstice EM Features 1-3
 - 1.3 Solstice EM Components 1-4
 - 1.3.1 Solstice EM Architecture 1-4
 - 1.3.2 Solstice EM Network Management Tools 1-7
 - 1.4 Basic Solstice EM Concepts 1-9
 - 1.4.1 Network Management Software 1-9
 - 1.4.2 Agents and Stations 1-9
 - 1.4.3 Management Information Servers 1-11
 - 1.4.3.1 More About MIS Databases 1-12
 - 1.4.3.2 More About the MIS Nerve Center 1-13
 - 1.4.3.3 More About PMI and MPAs 1-13
 - 1.4.3.4 MIS Ancillary Services 1-14
 - 1.4.3.5 More About MIS Data Access 1-15
 - 1.4.3.6 More About Object Orientation 1-15

1.4.4	Network Management Protocols	1-16
1.4.4.1	More About RPC	1-17
1.4.4.2	About MIBs	1-17
1.5	Solstice EM APIs	1-17
1.5.1	API Modules	1-18
1.5.2	Application Development Support Tools	1-19
1.6	Related Reading	1-20
1.7	Solstice EM Tools—Complete Listings	1-20
2.	Network Management and the Solstice EM Architecture	2-1
2.1	The Agent/Manager Model	2-1
2.2	Client/Server Architecture	2-2
2.3	Distributed Management	2-4
2.4	Network Management Protocol Support	2-7
2.4.1	RPC Support	2-8
2.5	Simple Requests	2-11
2.5.1	SNMP Support	2-11
2.5.2	CMIP Support	2-15
2.5.2.1	Telecommunications Management Network	2-16
2.5.3	Other Network Management Protocols	2-17
2.5.4	Java Dynamic Management Kit Agents	2-17
2.6	Object Classes and Event Notification Types	2-18

Part II Customizing Solstice EM Tools

3.	Using Solstice EM for Fault Management	3-1
3.1	Fault Management Summary	3-1
3.1.1	Before Starting Fault Management	3-2
3.2	Using Fault Management	3-3
3.3	Viewing Fault Status	3-3
3.3.1	Changing the Color Associated with a Severity	3-4

3.3.2	Alarm Severity Propagation	3-5
3.3.3	Access to Tools, Features, and Database Objects	3-5
3.4	Reporting Faults as Alarms	3-5
3.5	The Event Logs Tool and Alarm Logging	3-6
3.5.1	Receiving Network Information	3-7
3.5.1.1	Polling	3-8
3.5.1.2	Monitoring Device Availability	3-8
3.5.2	Event Notifications	3-11
3.5.2.1	Example: Monitoring Event Notifications from CMIP Agents	3-11
3.5.3	Using SNMP Traps	3-12
3.5.3.1	Monitoring SNMP Traps with Nerve Center Requests	3-14
3.5.3.2	Creating a Separate Log for Enterprise-Specific Trap Notifications	3-15
3.5.3.3	Forwarding Events from SunNet Manager Consoles	3-21
4.	Using the Alarm Service	4-1
4.1	Network View Nodes	4-1
4.2	Alarm Management	4-3
4.3	The Alarm Service	4-4
4.4	Configuring the Alarm Service	4-6
4.4.1	Adding Logs to emAlarmLogList	4-7
4.4.2	Deleting Logs from the Event Logs Window	4-7
4.4.3	Turning Off the Alarm Service	4-8
4.5	Alarm Information Display in Solstice EM Tools	4-8
4.5.1	Alarm Information Display in Alarms Window	4-8
4.5.2	Alarm Information Display in Network Views	4-9
4.6	User-configurable Alarm Log Record Filter for Alarm Service	4-10
5.	Using the Event Logs Tool	5-1

5.1	Log Process Overview	5-1
5.1.1	Attributes of a Log	5-2
5.1.2	Log Records Generated by Nerve Center Request Actions	5-3
5.2	Starting the Event Logs Tool	5-4
5.3	Using the Event Logs Tool	5-5
5.3.1	Accessing Logs on a Remote MIS	5-5
5.3.2	Importing Logs from a File into the Event Logs Tool	5-11
5.3.3	Configuring Display of Log Properties	5-11
5.3.4	Adding Tools to the Event Logs Menu	5-12
5.4	Defining the CMIS Filter	5-13
5.4.1	A CMIS Filter That Accepts Notifications of a Specific Type	5-15
5.4.2	CMIS Filter with Multiple ANDs	5-17
5.4.3	A CMIS Filter That Accepts All Notifications	5-18
5.4.4	A CMIS Filter That Accepts No Notifications	5-18
5.5	Sample CMIS Filters	5-18
5.5.1	Creation of an Object Instance	5-19
5.5.2	Deletion of an Object Instance	5-19
5.5.3	Attribute Value Change of an Object Instance	5-20
5.5.4	State Changes Received From Agent	5-20
5.6	Event Logs Tool Configuration File	5-21

Part III Network Management Protocol Support

6. Managing Devices Using RPC Agents 6-1

6.1	Types of RPC Agent Management	6-1
6.2	Preparing for Device Management with RPC Agents	6-5
6.3	RPC Management Protocol Adapter	6-11
6.4	RPC MPA Configuration Parameters	6-12

7. Using Cooperative Consoles with Solstice EM 7-1

7.1	Cooperative Console Forwarding	7-1
-----	--------------------------------	-----

7.2	Filtering Criteria for Information Forwarding	7-3
7.3	Cooperative Consoles Configuration and Operation	7-4
7.4	Receiving SunNet Manager Alarms	7-7
8.	SunNet Manager Application Support	8-1
8.1	Solstice EM Compatibility with SunNet Manager	8-1
8.2	Access to Solstice EM Features from SNM Applications	8-4
8.3	Adding an SNM Application to Solstice EM	8-5
8.3.1	Forwarding Event and Topology Information from SunNet Manager to Solstice EM	8-9
8.3.2	SunNet Manager Application Support	8-10
8.4	Information for Configuring Specific SNM Applications	8-11
8.4.1	Running Solstice EM and Applications on Hosts With a New IP Address or Name	8-11
8.4.2	Configuring Remedy's Action Request System (ARS) to Work with Solstice EM	8-13
8.4.3	Configuring Konfig 2.4 to Work with Solstice EM	8-14
8.4.4	Configuring Optivity 7.0 to Work with Solstice EM	8-16
8.4.5	Configuring Landmark's Performance Works to Work with Solstice EM	8-22
8.5	Importing an SNM Database into Solstice EM	8-25
8.6	Access to SNM Agents by SNM Applications	8-25
8.7	Access to SNM Agents by Solstice EM Applications	8-28
8.7.1	Configuration	8-29
8.7.2	Agent Support	8-29
8.7.3	Support for SNM Proxy Agents	8-29
9.	SNMP Management	9-1
9.1	SNMP Managed Components	9-1
9.2	SNMP Management Protocol Adapter	9-3
9.3	SNMP MPA Configuration	9-3
9.4	Specifying the Version of SNMP Used	9-4

9.4.1	Calling the <code>set_management_protocol</code> Function of the <code>EMSnmpAgent</code> Class	9-4
9.4.2	Using the PMI to Set the <code>managementProtocol</code> Attribute	9-5
10.	SunNet Manager SNMP Proxy Agents	10-1
10.1	Proxy Agents	10-1
10.2	SNMP Proxy Agent Operation	10-4
10.3	SNMP Trap Daemon (<code>em_snmp-trap</code>) Operation	10-7
10.4	Schema Files	10-7
10.5	SNMP Version 2 Support	10-9
10.5.1	SNMPv2 Enhancements	10-10
10.5.1.1	Structure of Management Information	10-10
10.5.1.2	Protocol Operations	10-10
10.5.2	SNMPv2 Files	10-11
10.5.3	Using the <code>v2mib2schema</code> Program	10-11
11.	Mapping SNMP Traps to CMIP Event Notifications	11-1
11.1	SNMP Support	11-1
11.2	Trap Daemon Operation	11-2
11.2.1	Starting the Trap Daemon	11-4
11.2.2	Stopping the Trap Daemon	11-4
11.3	The Structure of SNMP Traps	11-5
11.3.1	SNMPv1	11-5
11.3.2	SNMPv2c	11-7
11.4	Default Trap Mapping	11-8
11.4.1	Default Method for Specifying the Source of an Alarm	11-9
11.4.1.1	SNMPv1	11-9
11.4.1.2	SNMPv2c	11-9
11.4.2	Default <code>perceivedSeverity</code> Values	11-10
11.4.3	Default <code>probableCause</code> Values	11-11
11.4.4	Default <code>additionalText</code> Information	11-12

11.4.5	Default Event Notification Type	11-12
11.4.6	Default Location of Information from Trap Variable Bindings	11-13
11.5	Trap Daemon Behavior When No Mapping is Provided	11-13
11.6	Format of <code>trap_maps</code> File	11-14
11.6.1	Enterprise Mapping Blocks	11-14
11.6.1.1	SNMPv1	11-15
11.6.1.2	SNMPv2c	11-15
11.6.2	Mapping Records	11-16
11.6.2.1	SNMPv1	11-18
11.6.2.2	SNMPv2c	11-19
11.6.3	<attr-value> Definitions	11-20
11.6.3.1	Constant	11-21
11.6.3.2	Trap Variable Binding Value	11-21
11.6.3.3	Trap Variable Binding Name	11-21
11.6.3.4	Trap Variable Binding Index	11-22
11.6.3.5	Embedding Strings in <code>varbind</code> Expressions	11-22
11.6.3.6	Defining and Using <code>varbind-to-substring</code> Tables	11-23
11.7	Customizing the Mapping of SNMP Traps	11-25
11.7.1	Overview	11-25
11.7.2	How to Customize SNMP Trap Mapping	11-25
11.7.3	Configuring CMIP notification <code>managedObjectClass</code>	11-27
11.7.3.1	The keyword <code>\$ALLVARS</code>	11-28
11.7.3.2	The Keyword <code>\$NORFC2089</code>	11-29
11.7.4	Attribute Type Mapping	11-30
11.7.5	Wild Cards for <code>trap_mapping</code>	11-32
11.7.6	Using FDN Templates to Specify the Source of a Trap	11-33
11.7.6.1	Understanding FDNs and RDNs	11-34
11.7.6.2	Building FDN Templates	11-36
11.8	Distributed Trap Handling	11-37
11.8.1	Filtering SNMP Traps for Other Managers	11-38

12.	Configuring Communication With CMIP Agents	12-1
12.1	Tasks for Setting Up Your System to Manage CMIP Agents	12-1
12.2	Preparing the System for CMIP Configuration	12-4
12.2.1	Determining the Distribution Model	12-4
12.2.2	Installing the Required SunLink Products	12-5
12.2.3	Gathering Your Configuration Information	12-6
12.3	Compile and Load CMIP Agent Object Types into MIS	12-6
12.4	Starting and Configuring SunLink OSI	12-7
12.5	Access Control	12-8
12.6	Starting and Configuring SunLink CMIP 9.0	12-8
12.7	Starting and Configuring the CMIP MPA	12-10
12.8	Runtime Parameters	12-13
12.8.1	Auxiliary Server Container	12-13
12.8.2	CMIP MPA Object	12-14
12.8.3	em_cmip Parameters	12-14
12.8.4	Sample Program to Retrieve Runtime Parameters	12-18
12.9	Configuring Multiple MPAs on One System	12-19
13.	Configuring CMIP MPA Overload	13-1
13.1	Understanding CMIP MPA Overload	13-1
13.2	Configuration Parameters	13-2
13.2.1	Overload Control Parameter	13-3
13.2.2	Overload Notification Parameter	13-3
13.2.3	Overload Threshold Parameter	13-3
13.2.4	Minimum Threshold Parameter	13-4
13.2.5	Overload Instruction Parameter	13-4
13.2.6	Poll Interval Parameter	13-4
13.3	Management Information Tree of Overload Control Objects	13-5
13.4	GDMO Classes	13-6
13.4.1	Mapping Between Attributes of the GDMO Classes and Configuration Parameters	13-6

13.4.2	emOverloadControlContainer Class	13-6
13.4.3	emOverloadController Class	13-7
13.4.4	mpaOverloadController Class	13-7
13.4.5	Overload Sample Programs	13-8
13.4.5.1	get_agent_admin_state	13-9
13.4.5.2	overload_action	13-15
13.4.5.3	overload_alarm	13-21
13.4.5.4	overload_get	13-34
13.4.5.5	overload_set	13-42
13.4.5.6	set_agent_admin_state	13-50

Part IV Nerve Center

14. Nerve Center Overview 14-1

14.1	Nerve Center Components	14-1
14.2	Nerve Center Documentation	14-2
14.3	Nerve Center Operation	14-3
14.3.1	How a Request Gets Information	14-3
14.3.1.1	Where and When a Request's Notifications Arise	14-3
14.3.1.2	When Information From Managed Objects can Arrive	14-4
14.3.2	Variables and Attributes in a Request	14-5
14.3.2.1	Attributes	14-5
14.3.2.2	System Variables	14-6
14.3.2.3	User Variables	14-6
14.3.2.4	How Notifications and Poll Responses are Delivered	14-6
14.3.3	Where and When a Condition is Evaluated	14-7
14.3.4	Action at a Transition	14-7
14.3.4.1	Supported Actions	14-7
14.3.4.2	Logging an Event	14-8
14.3.4.3	Forwarding an SNMP Trap	14-8

14.3.5	Specifying the Objects to be Polled	14-8
14.3.6	Alarm Logging and the Alarm Service	14-10
15.	Requesting Data in Solstice EM	15-1
15.1	Polling for Data in Solstice EM	15-1
15.1.1	Direct Polling	15-2
15.1.2	Indirect Polling	15-2
15.1.3	Event Request Polling	15-2
15.2	Subscribing for Events	15-3
15.2.1	Combining Polling and Event-Subscription	15-3
15.3	Using Solstice EM Tools for Polling	15-4
15.4	Working with Basic Requests	15-5
15.4.1	Viewing Basic Request Information	15-5
15.4.2	Creating, Modifying, and Initiating Basic Requests	15-10
15.5	Working with Advanced Requests	15-12
15.5.1	Creating, Modifying, and Initiating Advanced Requests	15-13
15.6	Building Blocks: States, Transitions, and Conditions	15-15
15.6.1	State Machine Diagrams	15-16
15.6.2	Sample Request Template	15-19
15.6.2.1	Setting the Target Managed Object	15-21
15.6.2.2	Polling for an SNMP Attribute	15-23
15.6.3	Controlling Fault Status Color	15-23
15.6.3.1	Using <code>alarmOi()</code> to Clear Previous Alarms	15-25
15.6.3.2	Alarm-logging Tips	15-25
15.7	Designing Request Templates	15-26
15.8	Requests Based on Polling	15-28
15.8.1	Adding States	15-30
15.8.2	Adding Conditions	15-31
15.8.3	Adding Transitions	15-32
15.9	Polling RPC Agents	15-35
15.9.1	Targeting the RPC <code>ping-reach</code> Group	15-37

15.9.2	Correlating Information from Multiple Polls	15-38
15.10	Requests Based on Event Subscription	15-41
15.10.1	Event Logging and Alarm Service Monitoring of Alarm Logs	15-41
15.10.2	Mapping of SNMP Traps to CMIP Event Notifications	15-41
15.11	Subscribing for Enterprise-Specific SNMP Traps	15-42
15.11.1	Initiating the Event Subscription	15-43
15.11.2	Listening for Incoming Events	15-44
15.12	Requests that Combine Subscription and Polling	15-47
15.12.1	Checking for a Correct Target	15-49
15.13	Building Request Definitions	15-51
16.	Debugging Request Templates	16-1
16.1	Nerve Center Debugging Agents	16-1
16.2	Activating RCL Print Statements	16-2
16.3	Turning Off Debug Agents	16-3
17.	Building Templates for SunNet Manager Event Requests	17-1
17.1	RPC Agents	17-1
17.2	Nerve Center's SNM Event Request Capability	17-3
17.3	SNM Alarms	17-4
17.4	Building SNM Event Request Templates	17-6
17.4.1	Subscribing for SNM Events	17-9
17.4.2	Sending an SNM ping Event Request	17-10
17.4.3	Waiting for a Response to the Event Request	17-11
18.	Building Advanced Requests	18-1
18.1	Components of Request Templates	18-1
18.1.1	State Machine Diagrams	18-2
18.2	Using the Design Advanced Requests Tool to Build Nerve Center Templates	18-4
18.2.1	Starting Request Designer	18-5

18.2.2	Creating a New Nerve Center Template	18-6
18.2.3	Modifying an Existing Nerve Center Template	18-6
18.2.4	Deleting Nerve Center Templates	18-7
18.2.5	Exporting Nerve Center Templates to an ASCII File	18-7
18.2.6	Importing Nerve Center Templates from an ASCII File	18-8
18.3	Conditions	18-9
18.4	States	18-11
18.4.1	Adding States to a Nerve Center Template	18-11
18.4.2	Modifying States in a Nerve Center Template	18-12
18.5	Transitions	18-12
18.5.1	Creating New State-to-State Transitions in a Template	18-13
18.5.2	Deleting Transitions from a Template	18-13
18.5.3	Reordering Transitions	18-14
18.6	Actions	18-16
18.6.1	Adding Actions at a Transition	18-17
18.6.2	Deleting Actions at a Transition	18-18
18.6.3	Reordering the Actions at a Transition	18-19
18.7	Poll Rates	18-20
18.7.1	Creating New Poll Rates	18-21
18.7.2	Modifying a Poll Rate	18-22
18.8	Modifying the Mapping of Colors to Severities	18-23
18.9	Graphical State Diagram Display	18-24
18.9.1	Creating a Template Through the State Diagram Display	18-25
18.9.2	Other Tasks in the Graphical Display	18-26
19.	Nerve Center Utilities	19-1
19.1	<code>em_ncimport</code> and <code>em_ncexport</code>	19-1
19.1.1	Options	19-1
19.1.2	Examples	19-3
20.	Request Condition Language	20-1

20.1	Conditions	20-1
20.2	Types of Operands	20-2
20.3	Constants	20-3
20.4	Variables in a Condition	20-3
20.4.1	Variable Names	20-4
20.4.2	Scope of Variables	20-4
20.5	Data Types	20-5
20.6	System Variables	20-5
20.7	Attributes	20-6
20.7.1	Syntax of Attribute Names	20-7
20.8	Operators	20-8
20.8.1	Logical Operators	20-9
20.8.2	Bitwise Operators	20-10
20.8.3	Precedence and Associativity	20-10
20.9	Control Structures	20-11
20.9.1	IF Constructs	20-11
20.9.2	IF ELSE Constructs	20-12
20.9.3	WHILE Constructs	20-13
20.9.4	FOREACH Constructs	20-13
20.9.5	Nested Constructs	20-14
20.10	Timestamp Arithmetic	20-16
20.11	Error Checking	20-16
21.	Using RCL System Variables	21-1
21.1	System Variables	21-1
21.1.1	\$eventInfo	21-2
21.1.2	\$eventOI	21-4
21.1.3	\$eventTime	21-4
21.1.4	\$eventType	21-4
21.1.5	\$messType	21-6

- 21.1.6 \$pollfdn 21-7
- 21.1.7 \$pollFdnSet 21-8

22. RCL Functions 22-1

- 22.1 Summary of RCL Built-in Functions 22-1
 - 22.1.1 AlarmLog Functions 22-1
 - 22.1.2 String-Handling Functions 22-2
 - 22.1.3 Value Check Functions 22-2
 - 22.1.4 Name Conversion Functions 22-2
 - 22.1.5 Action Functions 22-2
 - 22.1.6 ASN.1 Conversion Functions 22-2
 - 22.1.7 SunNet Manager RPC Request Functions 22-2
 - 22.1.8 Debugging Function 22-3
 - 22.1.9 Constructed-Type Handling Functions 22-3
 - 22.1.10 Time Functions 22-3
 - 22.1.11 Event-Handling Functions 22-3
 - 22.1.12 Request Control Functions 22-3
- 22.2 The RCL Functions 22-4
 - 22.2.1 AddressStrToAddress 22-4
 - 22.2.2 Alarm 22-4
 - 22.2.2.1 Alarm Logging and Viewer Fault Status 22-5
 - 22.2.3 AlarmOi 22-6
 - 22.2.4 AlarmStr 22-8
 - 22.2.5 AnyStr 22-9
 - 22.2.6 AppendRdn 22-10
 - 22.2.7 AsnToStr 22-12
 - 22.2.8 CompareLists 22-13
 - 22.2.9 Defined 22-13
 - 22.2.10 Exit 22-14
 - 22.2.11 Exclude 22-15

22.2.12	Extract	22-15
22.2.13	FinalStr	22-16
22.2.14	FirstStr	22-16
22.2.15	GetTimeStamp	22-17
22.2.16	Include	22-17
22.2.17	InitialStr	22-18
22.2.18	IsChoice	22-18
22.2.19	IsList	22-19
22.2.20	IsMember	22-19
22.2.21	Mail	22-20
22.2.22	NameToAddress	22-20
22.2.23	NameToOid	22-20
22.2.24	NumElements	22-21
22.2.25	OiNameToOi	22-21
22.2.26	OiToOiName	22-22
22.2.27	Print	22-22
22.2.28	ReplaceMember	22-23
22.2.29	SendAction	22-23
22.2.30	SendEvent	22-24
22.2.31	SendTrap	22-25
22.2.32	Set	22-25
22.2.33	SnmEventRequest	22-27
22.2.34	SnmKillRequest	22-32
22.2.35	StrToAsn	22-32
22.2.36	StrCat	22-33
22.2.37	Strstr	22-33
22.2.38	StrStrPlus	22-34
22.2.39	Subscribe	22-34
22.2.40	SubscribeFilter	22-35
22.2.40.1	Considerations	22-36

22.2.40.2	Examples	22-36
22.2.41	SubscribeOi	22-37
22.2.42	TrapGenericType	22-38
22.2.43	TrapSpecificType	22-39
22.2.44	Undefine	22-40
22.2.45	Unixcmd	22-41
22.2.46	UnSubscribe	22-41

Figures

FIGURE 1-1	Solstice EM Architecture Overview	1-6
FIGURE 1-2	Agent Communications Overview	1-10
FIGURE 1-3	Overview of Management Information Servers	1-12
FIGURE 2-1	Agent/Manager Communication in Solstice EM Environment	2-2
FIGURE 2-2	Solstice EM Network Tools	2-3
FIGURE 2-3	A Sample Configuration Using MIS-to-MIS Communication	2-5
FIGURE 2-4	Topology Tree as Seen by Network Views Window Connected to MIS A	2-6
FIGURE 2-5	Topology Tree as Seen by Network Views Window Connected to MIS Net_B	2-6
FIGURE 2-6	Topology as Seen in Network Views Window Connected to MIS Net_D	2-7
FIGURE 2-7	MIS-to-MIS Connection From MIS A to MIS Net B	2-7
FIGURE 2-8	Polling RPC Agents	2-9
FIGURE 2-9	Using SNMP Event Requests With Solstice EM	2-10
FIGURE 2-10	MIS Communication With SNMP Agents	2-11
FIGURE 2-11	SNMP Trap Daemon Operation	2-13
FIGURE 2-12	Viewing Trap Notifications	2-14
FIGURE 2-13	SNMP Proxy Agent Operation	2-15
FIGURE 2-14	CMIP MPAs in Distributed Configuration	2-16
FIGURE 2-15	TMN Q3 Connection to Solstice EM	2-17
FIGURE 3-1	Selecting a Severity for <code>communicationsAlarm</code> Generated by Monitor	3-10

FIGURE 3-2	CMIP Management of a Cellular Network	3-12
FIGURE 3-3	Viewing Trap Notifications in the Alarms Window	3-13
FIGURE 3-4	Solstice EM Processing of SNMP Traps	3-14
FIGURE 3-5	Example of SNMP Trap Handling Using SnmpLinkUp/DownTrap Request	3-15
FIGURE 3-6	Viewing AlarmLog Properties in the Event Logs Properties Dialog	3-17
FIGURE 3-7	CMIS Filter Window	3-18
FIGURE 3-8	CMIS Filter Item Dialog Box	3-18
FIGURE 3-9	Adding an Item to the Default AlarmLog Discriminator	3-19
FIGURE 3-10	AlarmLog Discriminator Construct With enterpriseSpecificTraps Excluded	3-19
FIGURE 3-11	Creating a New Log for enterpriseSpecificTraps (will be displayed Filer pane)	3-20
FIGURE 3-12	Specifying a CMIS Filter for enterpriseSpecificTraps	3-21
FIGURE 4-1	Network View Nodes	4-2
FIGURE 4-2	Logging of Alarms to the AlarmLog	4-4
FIGURE 5-1	Customize Tools Menu	5-6
FIGURE 5-2	Creating a New Log	5-8
FIGURE 5-3	Defining a Discriminator to Log SNM Events	5-9
FIGURE 5-4	Modifying a Log's log filter	5-10
FIGURE 5-5	Viewing Log Objects in the Column Headings Window	5-12
FIGURE 5-6	Customize Tools Menu Window	5-13
FIGURE 6-1	Communication With RPC Agents in Direct Polling Requests	6-2
FIGURE 6-2	Using SNM Event Requests with Solstice EM	6-4
FIGURE 6-3	Selecting RPC Agents to be Configured During Network Discovery	6-7
FIGURE 7-1	Forwarding of Information to Central Management Station	7-2
FIGURE 7-2	Information Forwarding From SNM Console to Solstice EM MIS	7-6
FIGURE 8-1	SNM-Solstice EM Compatibility	8-3
FIGURE 8-2	SNM Application Accessing Solstice EM Features	8-4
FIGURE 8-3	Forwarding of Information to Central Management Station	8-10
FIGURE 8-4	SNM Application Accessing SNM Agents Over Solstice EM	8-26

FIGURE 8-5	Solstice EM Applications Accessing SNM Agents	8-28
FIGURE 9-1	Components of Solstice EM's SNMP Management Support	9-2
FIGURE 10-1	MIB, GDMO, and Schema Definitions	10-2
FIGURE 10-2	SNMP Proxy Agent Operation	10-4
FIGURE 11-1	em_snmp-trap Operation	11-3
FIGURE 11-2	SNMPv1 Trap PDU Structure	11-5
FIGURE 11-3	SNMPv2c Trap PDU Structure	11-8
FIGURE 11-4	SNMPv1 Trap Mapping Record Format	11-18
FIGURE 11-5	Sample FDN for internetSystem Group Object Instance	11-35
FIGURE 11-6	Sample ifTable FDN	11-35
FIGURE 11-7	Sample FDN Template	11-37
FIGURE 12-1	Configuring Solstice EM for Communication with CMIP Agents	12-3
FIGURE 12-2	Auxiliary Server Container	12-14
FIGURE 13-1	Management Information Tree of Overload Control Objects	13-5
FIGURE 15-1	Viewing Request Information in the Basic Requests Main Window	15-5
FIGURE 15-2	Viewing Requests in the Basic Requests Available Window	15-6
FIGURE 15-3	Viewing Request Groups in the Basic Requests Groups Window	15-7
FIGURE 15-4	Basic Requests Properties Conditions Window	15-8
FIGURE 15-5	Basic Requests Properties General Window	15-9
FIGURE 15-6	Basic Requests Group Properties Window	15-10
FIGURE 15-7	Viewing Available Requests in the Advanced Request Dialog	15-12
FIGURE 15-8	Advanced Request Examine Window	15-13
FIGURE 15-9	Request Example with Poll Rates	15-17
FIGURE 15-10	Request Example with Poll Rates and Severities	15-18
FIGURE 15-11	Request Example with Conditions	15-19
FIGURE 15-12	IsSnmpSystemUp Sample Request Template	15-20
FIGURE 15-13	State Diagram of IsSnmpSystemEverDown Template	15-30
FIGURE 15-14	Entering Condition Code in the Design Advanced Requests	15-32

FIGURE 15-15	IsSnmpSystemEverDown Template	15-34
FIGURE 15-16	State Diagram of SnmpPingBackoffReachable Request	15-36
FIGURE 15-17	State Diagram for IsEnterpriseSpecificTrap Template	15-45
FIGURE 15-18	SNMP Trap Subscription Template	15-47
FIGURE 15-19	SnmpLinkUpDownTrap Template State Diagram	15-48
FIGURE 17-1	Using SNM Event Requests with Solstice EM	17-2
FIGURE 17-2	State Machine Diagram for DeviceReachablePing Template	17-7
FIGURE 18-1	Request Example with Poll Rates and Severities	18-3
FIGURE 18-2	Example of Export to ASCII File	18-8
FIGURE 18-3	Viewing RCL Conditions in the Conditions Window	18-10
FIGURE 18-4	Order of Transitions in a Template	18-12
FIGURE 18-5	Reordering State Transition - Move Up	18-14
FIGURE 18-6	Reordering State Transition - Move Down	18-15
FIGURE 18-7	Use of RCL Variables in Mail Action	18-16
FIGURE 18-8	Adding a Condition as an Action at a Transition	18-18
FIGURE 18-9	Creating a New Poll Rate	18-22
FIGURE 18-10	Nerve Centre's Mapping of Colours to Severities	18-23
FIGURE 18-11	Graphical State Diagram Display	18-24

Tables

TABLE 1-1	Common Solstice EM Tools	1-7
TABLE 1-2	Solstice EM API Modules	1-18
TABLE 1-3	Solstice EM Tools – Complete List, Sorted by Binary Name	1-21
TABLE 3-1	Default Color-Coding of Severities	3-4
TABLE 3-2	Default SNMP Trap Notifications and Severities	3-13
TABLE 3-3	Mapping of SNM Console Fault Indications to <code>perceivedSeverity</code> Values	3-22
TABLE 4-1	Alarm Log Record Processing Options	4-10
TABLE 5-1	Log Object Attributes	5-2
TABLE 5-2	Command-Line Options for the <code>em_logmgr</code> Command	5-4
TABLE 5-3	Format Specifier Definitions	5-14
TABLE 5-4	Operator Definitions	5-14
TABLE 5-5	Notification Types and Numbers	5-16
TABLE 6-1	Ready-to-Use RPC Request Templates	6-8
TABLE 7-1	Mapping of SNM Console Fault Indications to <code>perceivedSeverity</code> Values	7-7
TABLE 11-1	SNMPV1 Field Descriptions	11-6
TABLE 11-2	Standard SNMP Trap Types	11-6
TABLE 11-3	SNMPV2c Field Descriptions	11-8
TABLE 11-4	Default Color-Coding of Severities	11-10
TABLE 11-5	Default IP Management Trap Event Types	11-12

TABLE 11-6	Standard Event Notifications	11-16
TABLE 11-7	Attribute Value Type Conversions	11-30
TABLE 11-8	Wild Cards for <code>trap_mapping</code>	11-32
TABLE 12-1	Object Properties/Create Object Fields	12-12
TABLE 12-2	<code>em_cmip</code> Parameters	12-15
TABLE 12-3	<code>em_oct</code> Parameters	12-17
TABLE 13-1	GDMO Mapping	13-6
TABLE 13-2	Sample Programs Description	13-8
TABLE 14-1	Action Menu Items	14-7
TABLE 15-1	Solstice EM Request Tools	15-4
TABLE 15-2	Enterprise Specific Traps Example	15-42
TABLE 17-1	Mapping of SNM Event Severities	17-5
TABLE 18-1	Action Menu Items	18-16
TABLE 18-2	Poll Rates	18-20
TABLE 20-1	RCL Syntax Restraints	20-3
TABLE 20-2	System Variables Available to a Condition	20-5
TABLE 20-3	RCL Operator Symbols	20-8
TABLE 20-4	Precedence of Operators	20-10
TABLE 21-1	System Variables Available to a Condition	21-1
TABLE 21-2	<code>perceivedSeverity</code> Values	21-3
TABLE 21-3	Values of <code>\$messType</code>	21-6
TABLE 22-1	Valid Alarm Severities	22-5
TABLE 22-2	Arguments in <code><EventRequest></code>	22-27
TABLE 22-3	Relational Operators in SNM Request Thresholds	22-29
TABLE 22-4	Data Types for Threshold Operands	22-30
TABLE 22-5	Mapping of SNM Event Severities	22-31
TABLE 22-6	Standard SNMP Trap Types	22-39

Preface

This 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, Request Designer, and Log Manager tools.

Our goal in writing this document was to anticipate what you, our customers, would want to customize 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 information on using Solstice EM tools (other than Request Designer and Log Manager) should refer to *Managing Your Network*.

Before You Read This Book

If you have just acquired the Solstice EM product, you should read Chapter 2 "Network Management and the Solstice EM Architecture" for an overview of the Solstice EM architecture and possible scenarios for deploying Solstice EM. The *Management Information Server Guide* also contains an overview of the Solstice EM product functions, features, and components. Read the *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.

How This Book Is Organized

This document is organized as follows:

Part 1 — Overview

Chapter 1 "Introducing Solstice Enterprise Manager" gives a complete overview of *Solstice Enterprise Manager*.

Chapter 2 "Network Management and the Solstice EM Architecture" describes the basic structure or architecture of *Solstice Enterprise Manager* and how that structure works to meet your network management needs.

Part 2 — Customizing Fault Management

Chapter 3 "Using Solstice EM for Fault Management" provides procedures and examples for using Solstice EM to perform fault and performance management tasks.

Chapter 4 "Using the Alarm Service" describes the Solstice EM MIS Alarm Service, which controls fault status indication in the Viewer.

Chapter 5 "Using the Event Logs Tool" provides guidance on using the Log Manager to create and modify event logs.

Part 3 — Network Management Protocol Support

Chapter 6 "Managing Devices Using RPC Agents" provides procedures and examples for using SunNet Manager Remote Procedure Call (RPC) agents with Solstice EM.

Chapter 7 "Using Cooperative Consoles with Solstice EM" describes the use of Cooperative Consoles to forward management information from Site/SunNet/Domain Manager Consoles to the Solstice EM MIS.

Chapter 8 "SunNet Manager Application Support" describes Solstice EM support for Site/SunNet/Domain Manager applications.

Chapter 9 "SNMP Management" provides an overview of the Solstice EM components that support Simple Network Management Protocol, and information on configuring the SNMP Management Protocol Adapter.

Chapter 10 "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 11 "Mapping SNMP Traps to CMIP Event Notifications" describes the SNMP trap handling capabilities of the Solstice EM SNMP trap daemon, `em_snmp-trap`, and the procedure for customizing conversion of SNMP traps to CMIP event notifications.

Chapter 12 "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.

Part 4 — Nerve Center

Chapter 14 "Nerve Center Overview" provides an overview of the operation of Solstice EM Nerve Center.

Chapter 15 "Requesting Data in Solstice EM" provides guidance and examples for using the Request Designer tool and Request Condition Language to build Nerve Center request templates.

Chapter 16 "Debugging Request Templates" describes the Solstice EM facilities for debugging Nerve Center request templates.

Chapter 17 "Building Templates for SunNet Manager Event Requests" provides information and examples for using the Nerve Center's SunNet Manager event request capability.

Chapter 18 "Building Advanced Requests" provides information on using the Request Designer tool to create and modify Nerve Center request templates.

Chapter 19 "Nerve Center Utilities" describes the command-line utilities for importing and exporting Nerve Center request templates.

Chapter 20 "Request Condition Language" provides information on the Request Condition Language (RCL) used in building conditions used in Nerve Center request templates.

Chapter 21 "Using RCL System Variables" describes the Nerve Center system variables for building Nerve Center request templates.

Chapter 22 "RCL Functions" describes the built-in functions for using construct conditions as components in Nerve Center request templates.

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 <code>.login</code> file. Use <code>ls -a</code> to list all files. <code>machine_name%</code> You have mail.
AaBbCc123	What you type, contrasted with on-screen computer output	<code>machine_name%</code> su Password:
<AaBbCc123>	Command-line placeholder: replace with a real name or value	To delete a file, type <code>rm <filename></code>
<i>AaBbCc123</i>	Book titles, new words or terms, or words to be emphasized	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> 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 tools.

Mouse/Menu Interactions

We have pursued a minimalist approach in describing a user's interactions with the graphical-based tools 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 tools 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 tools—those tools 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 tool 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 press **Esc** while your mouse pointer is in the tear-off menu window.

Accessing Sun Documentation Online

The `docs.sun.comsm` web site enables you to access Sun technical documentation on the Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject at `http://docs.sun.com`

Also, you can view the online documentation by pointing your browser to the following URL, `file:/opt/SUNWconn/em/docs/SEMDOCHP/index.html`

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions. You can send your comments by email to `docfeedback@sun.com`.

Please include the part number of your document in the subject line of your email.

PART I

Overview

Introducing *Solstice Enterprise Manager*

This chapter provides introductory information designed to help you get going with *Solstice Enterprise Manager* (Solstice EM) software and your day-to-day network management tasks.

This chapter describes the following topics:

- Section 1.1 “What is Solstice EM?” on page 1-1
- Section 1.2 “Solstice EM Features” on page 1-3
- Section 1.3 “Solstice EM Components” on page 1-4
- Section 1.4 “Basic Solstice EM Concepts” on page 1-9
- Section 1.5 “Solstice EM APIs” on page 1-17
- Section 1.6 “Related Reading” on page 1-20
- Section 1.7 “Solstice EM Tools—Complete Listings” on page 1-20

1.1 What is Solstice EM?

Solstice EM software is a distributed, standards-based, hierarchical, object-oriented suite of enterprise-grade network management tools, and a rich set of application programming interfaces (APIs) for developing custom network management applications. That means Solstice EM is a highly flexible, totally customizable network management solution that can be scaled and configured to fit your needs. You can use any or all of the Solstice EM tools “as is,” or you can combine them with your own custom applications written to the Solstice EM API.

1.1.1

What Can You Manage With Solstice EM?

Solstice EM provides tools for monitoring, evaluating, and refining your network. The management functions you can perform with Solstice EM include:

- **Discovering components on your network**

Automated tools let you keep track of components as they are added or removed from your network. Support for CMIP, SNMP, and RPC agents allow you to gather information about a wide variety of network components. Component details are stored in industry-standard relational databases on one or more management information servers.

- **Visualizing your network**

A powerful Network Views tool serves as the graphical command center of Solstice EM, from which you can organize and manage your network components as groups of icons in logical, hierarchical, and geographical *views*. You can create as many different views of your network components as you want, using whatever view is most appropriate for the task at hand. For example, you can view your network components on a cartographically accurate geographical map background—a map of the world, perhaps, or a blueprint of one of your buildings—which lets you see at a glance not only *what* component is reporting a fault but *where* that component is physically located. You can then initiate actions to correct the fault directly from the Network Views tool.

- **Configuring network components**

Solstice EM object properties editors in combination with CMIP, SNMP, and RPC agents let you configure all aspects of your network components, from general network properties to detailed component data, such as individual router interfaces. Schema- and MIB-to-GDMO compilers let you customize specific object attributes.

- **Detecting, tracking, and correcting network problem**

A sophisticated alarm manager and request template designer, in combination with data viewers, and logging, reporting, and graphing tools, let you gather detailed information about your network, configure and respond to alarms, and design dynamic request templates to automatically gather and respond to network and component conditions.

- **Managing user access to objects and applications**

The Solstice EM Security tool lets you control user access to individual Solstice EM and custom tools, as well as to specific sets of network components.

- **Customizing the Solstice EM interface**

You can tailor Solstice EM tools in a wide number of ways, from pull-down and pop-up menus, to tool palettes, to default parameters and automatic monitoring methods. You can run almost every Solstice EM component from the UNIX

command line, if you want, to create script-based maintenance routines. You can even modify MIB, ASN.1, and GDMO definitions to customize settings for the types of objects you can work with in Solstice EM.

- **Developing custom network management applications**

Solstice EM is a standards-based, object-oriented environment that supports the Portable Management Interface (PMI) API over Simple Network Management (SNMP), Common Management Information (CMIP), and SunNet Manager Remote Procedure Call (RPC) protocols, as well as full conformance with Telecommunications Management Networks (TMN) standards.

1.1.2 Who Uses Solstice EM?

Large companies, for example, the telecommunications industry use Solstice EM to manage hundreds of thousands of network nodes, with network components ranging from mainframes, subnetworks, gateways, servers, routers, and hubs, to workstations, personal computers, and agent-enabled mobile devices. Such companies use the standard suite of Solstice EM tools alongside their own custom tools developed with the Solstice EM APIs and toolkit.

Smaller companies use Solstice EM “out of the box” to manage all their network components, knowing that Solstice EM can be scaled to meet whatever needs they have as their company grows.

Any company interested in secure, high-performance management of distributed network resources, from portable handheld devices to deskbound graphical workstations, can use Solstice EM’s distributed agent-oriented technology to manage virtually any type of network component.

1.2 Solstice EM Features

Solstice EM provides the following network management features:

- Comprehensive suite of network management tools, from objects to enterprises, infrastructure to interfaces.
- Manage virtually any network-aware component, from UNIX workstations to Windows-based PCs, printers to routers, subnetworks to network interfaces.
- Graphical “point and click” CDE (Common Desktop Environment) tools and UNIX command-line interfaces allow real-time ease-of-use and script-based automation.

- Multiple concurrent users run anywhere, anytime with UNIX-based security and scalability.
- Distributed, hierarchical, ASN.1 object-oriented architecture utilizing an industry-standard SQL-compatible relational database (Informix™ by default).
- Customizable interfaces and support for custom third-party applications by means of the PMI API over SNMP, CMIP, RPC, and TMN protocols.
- Development support tools are a set of API modules, object development tools, compilers, and debuggers; for developing your own custom network management applications.
- Backward-compatibility with SunNet™ Manager and Solstice Site/Domain Manager data and applications.

1.3 Solstice EM Components

Solstice EM components can be divided into two categories:

- Architecture—the structural components underlying the Solstice EM environment.
- Network Management Tools—a way to access and manipulate the Solstice EM environment and the components on your network.

These components are described in the following sections.

1.3.1 Solstice EM Architecture

The network management in the Solstice EM environment is based on the exchange of information between managers and agents. It is client/server-based, hierarchical and object-oriented environment comprising following components:

- Motif Applications—The manager applications run on the Motif environment. The applications can either be custom applications or Solstice EM Tools.
- JMA Server—The Java Management Adapter (JMA) provides the framework for the thin client/fat server model. JMA is not exposed to end users or developers. It is a transparent component lying between services such as the JMI and the MIS. JMA provides the infrastructure for services such as JMI API, Topology API, and Alarm API to communicate seamlessly with the MIS. It is responsible for the scheduling and synchronization of all PMI calls made by each Java API. It provides an event handling mechanism, which allows clients to register their own events and servers to forward the events to the clients.

- **SEM CORBA Gateway**—SEM CORBA Gateway translates CORBA manager requests in Interface Description Language (IDL) to Solstice EM Portable Management Interface (PMI) or equivalent requests. Also translates Solstice EM PMI responses to IDL or Internet Inter-ORB Protocol (IIOP) responses, and PMI events to CORBA events. The SEM CORBA Gateway is designed to work with standard management reference models (such as SNMP/IP and CMIP/OSI). The interfaces defined and implemented by the SEM CORBA Gateway define the interaction between it and applications.
- **Managed objects**—Solstice EM is centered around the concept of *managed objects*. A managed object is a set of programmatic services and attributes that describes a type of managed resource. In practical terms, it means that each physical component on your network is represented by one or more objects in an MIS database (see below). When you perform an action in Solstice EM on a network component—for example, setting a configuration parameter or retrieving status information, you are actually performing that action on the managed object representing the network component.

Every managed object has a set of characteristics, referred to as *attributes*, that define the kind of information the object can accept or provide, and a set of programmatic behaviors, called *methods*, that define what an object can do with that information. Specific values assigned to a given set of attributes are referred to as the object's *properties*. For example, an object attribute called `IPAddress` might have a value of `129.148.20.114`. See *Managing Your Network* for more information about gathering managed object attribute data.

- **Agent software**—provides object-level communication methods between your network components and the rest of Solstice EM—that is, the means by which you can use Solstice EM to get and set properties on managed objects. Solstice EM supports SNMP, CMIP, and RPC agents, and includes a set of CMIP- and RPC-based Solstice™ Enterprise Agents (SEA) that support a wide variety of managed objects, including legacy objects created in SunNet Manager and Solstice Site/Domain Manager. See Section 1.4.2, “Agents and Stations” for more information.
- **Management Information Server (MIS)**—a machine hosting an SQL database containing a Metadata Repository (MDR) and a Management Information Tree (MIT), which are used by Solstice EM to maintain information about the properties and relationships, respectively, of all managed objects on your network. An MIS daemon and a set of ancillary MIS service daemons provide MIS services on the MIS host. For data security or organizational convenience, you can use multiple MIS hosts, with one MIS database on each host. Multiple MISs can be linked so that all appear as one MIS to a given user. See Section 1.4.3, “Management Information Servers” for more information.
- **Application programming interfaces**—support communication and protocol translation between one or more MISs and Solstice EM and/or custom network management tools. See Section 1.5, “Solstice EM APIs” for more information.

- Solstice EM and/or custom network management tools—the tools you use to access and manipulate the Solstice EM environment and managed objects. These tools are described in Section 1.3.2, “Solstice EM Network Management Tools” .

The relationships between Solstice EM architectural components are illustrated in the following figure. Explanations of basic Solstice EM network management concepts are provided in Section 1.4, “Basic Solstice EM Concepts” .”

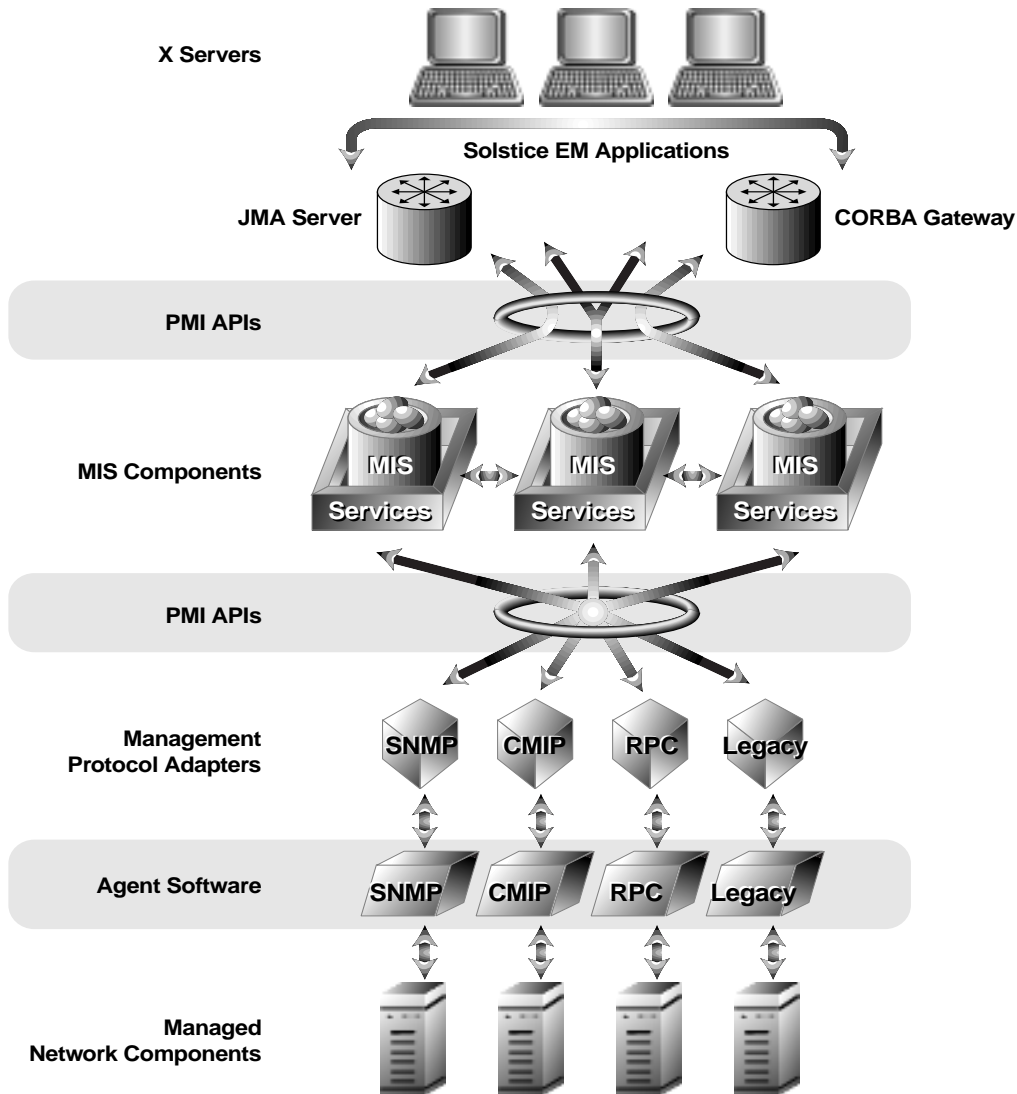


FIGURE 1-1 Solstice EM Architecture Overview

1.3.2 Solstice EM Network Management Tools

Solstice EM network management tools can be divided into three general categories:

- General management—tools for day-to-day network management tasks—these are the tools you will use most often.
- MIS—tools and services for running and managing MIS servers and databases; once MIS services have been started on one or more MIS hosts, you will use these tools for occasional MIS database maintenance chores.
- Application customization and development—tools for customizing the Solstice EM environment and creating custom management applications.

The following table briefly describes the common Solstice EM tools you are likely to use most often. The tools in the table are divided into two categories: general network management tools, those you will use on a day-to-day basis, and administration. A quick-reference table describing all of the Solstice EM tools is provided in Section 1.7, “Solstice EM Tools—Complete Listings”

TABLE 1-1 Common Solstice EM Tools

Tool	Description	More Information
<i>General Management Tools</i>		
Alarms	Configure, view, and respond to alarms posted against managed objects.	<i>Managing Your Network</i>
Data Collections	Create data requests to collect performance data about selected managed objects.	<i>Managing Your Network</i>
Grapher	Graph data from several Solstice EM tools.	<i>Managing Your Network</i>
Event Logs	View and manage logs and log objects.	<i>Managing Your Network</i>
Network Discovery	Automatically detect components on your network and populate the MIS with managed objects representing those components; a monitor function dynamically updates the MIS as components are added and removed from your network.	<i>Managing Your Network</i>
Network Tools	Customizable window providing access to Solstice EM and other tools; you can add and remove Solstice EM tools, third-party tools, or your own tools to this window; provides a handy starting point for Solstice EM.	<i>Managing Your Network</i>

TABLE 1-1 Common Solstice EM Tools *(Continued)*

Tool	Description	More Information
Network Views	Graphical tool for visualizing, organizing, and managing your network; most other Solstice EM tools can be run directly from Network Views; provides a convenient, customizable, point-and-click way to view and respond to alarms, and configure object properties. Views of your network can be superimposed over cartographically accurate map backgrounds, making it easy to determine the physical location of network components and faults.	<i>Managing Your Network</i>
Object Properties	View, create, and modify managed objects and object properties; commonly invoked from Network Views.	<i>Managing Your Network</i>
RPC/CMIP Data	Get, set, view, and modify properties for CMIP- and RPC-managed objects.	<i>Managing Your Network</i>
Administration Tools		
Administration	Customizable window providing access to Solstice EM administration tools.	<i>This Guide</i>
Advanced Requests	Create Nerve Center request templates; used by Solstice EM to poll for object properties or receive notifications from managed object agents, and then generate responses based on the data received.	<i>This Guide</i>
Automatic Management	Configure the MIS to launch and stop event requests automatically.	<i>Managing Your Network</i>
DB Backup/Restore	Back up and restore MIS databases.	<i>This Guide</i>
MIS Manager	Configure MIS communications and parameters.	<i>This Guide</i>
Object Editor	View, create, and delete managed object attributes directly in the MIS.	<i>This Guide</i>
Security	Manage user access to network management tools and managed objects.	<i>Managing Your Network</i>
SNMP MIB Browser	View attributes and modify SNMP attribute values for SNMP MIBs.	<i>Managing Your Network</i>
Topology Import/Export	Import or export an ASCII version of all or part of a MIS database; useful for recreating or transferring an MIS topology on another host.	<i>This Guide</i>

1.4 Basic Solstice EM Concepts

Before you start using Solstice EM to manage the components on your network, it is useful to review some basic network management concepts as they relate to Solstice EM.

The concepts explained in this section are:

- Network Management Software—page 1-9
- Agents and Stations—page 1-9
- Management Information Servers—page 1-11
- Network Management Protocols—page 1-16

1.4.1 Network Management Software

In the context of Solstice EM, *network management software* refers to one or more software tools, like the Solstice EM suite, that:

- Provide user interfaces through which network management tasks can be performed.
- Issue requests to network devices—to ask a given device to take some action, typically to provide some specific information to the manager.
- Receive responses to requests.
- Receive unsolicited information (*notifications*) from network components concerning component status, such as problems, abnormalities, and changes in the environment.

The Solstice EM tools function as network management software for various network management functions. For example, the Solstice EM Alarms tool lets you create and manage alarms for monitoring component status.

1.4.2 Agents and Stations

Rather than communicating directly with network components, Solstice EM communicates through software *agents* (middlemen) that are specifically configured to understand the particular kinds of information and settings a given network component can receive or provide.

In network management jargon, a *station* is a network management software component that communicates with one or more agents. In the case of Solstice EM, the station component is the MIS (see Section 1.4.3, “Management Information

Servers”). Consider the example, as illustrated in FIGURE 1-1 on page 1-6, a Solstice EM tool sends a request to an agent via the MIS and a Management Protocol Adapter (MPA). The agent, in turn, passes that request to the network component. On the return trip, the component’s response is passed first to the agent, then to the MIS by means of the MPA, and then finally back to the Solstice EM tool. Solstice EM handles the translation between various agent protocols transparently; for most management tasks, users do not need to know what type of agent is associated with a given network component.

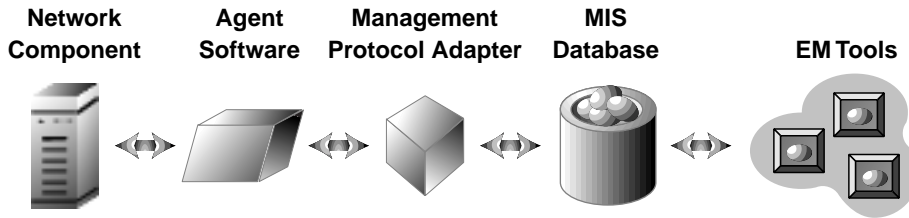


FIGURE 1-2 Agent Communications Overview

Every network component managed by Solstice EM has an associated software agent. In some cases, the agent software is embedded in the component; for example, burned into a component’s ROM. In most cases, however, the agent software resides on a host machine to which the component is connected, and provides services for a related set of components.

Solstice EM is based on the ISO-standard agent/station network management model. The strength of this model is that your network management software does not need to maintain configuration information for every network component available on the market. As new types of components are added or upgraded on your network, the agent software that travels with a given component is able to communicate the new configuration information to the management software.

Solstice EM supports the following common agent protocols:

- CMIP—Common Management Information Protocol
- SNMP—Simple Network Management Protocol
- RPC—SunNet Manager Remote Procedure Call

Solstice EM also provides full conformance with Telecommunications Management Networks (TMN) standards. See Section 1.4.4, “Network Management Protocols” for more information about network protocols supported by Solstice EM.

Note – Solstice EM installation includes by default a set of Solstice Enterprise Agents (SEA), which allow you to communicate with components configured for legacy SunNet/Site/Domain Manager environments. Other agents may be installed as well, depending on your package options. Refer to the *Installation Guide* for complete information.

1.4.3 Management Information Servers

In the Solstice EM environment, a Management Information Server (MIS) is a machine hosting an object-oriented SQL database containing information about every component on your network that is managed by Solstice EM. In this model, physical network components are represented as managed objects in an MIS database (see FIGURE 1-3 on page 1-12).

In general terms, the MIS provides the following services:

- Access control
- Requests
- Connections
- Events and alarms
- Logging
- Object management

Functionally, the resources on an MIS can be divided into four general categories:

1. An MIS database containing information about the components on your network.
2. An MIS Nerve Center that provides the logic and methods to actually do something with the information in the MIS; the Nerve Center is the source of requests and responses based on network conditions.
3. Portable Management Interface (PMI) APIs and Management Protocol Adapters (MPAs).
4. A set of ancillary MIS services that make the information in the MIS database available to network management applications and software agents.

For data security or logical convenience, you can use multiple MIS databases, with one MIS database on each host. Multiple MIS databases can be linked so that all appear as one database to a given user.

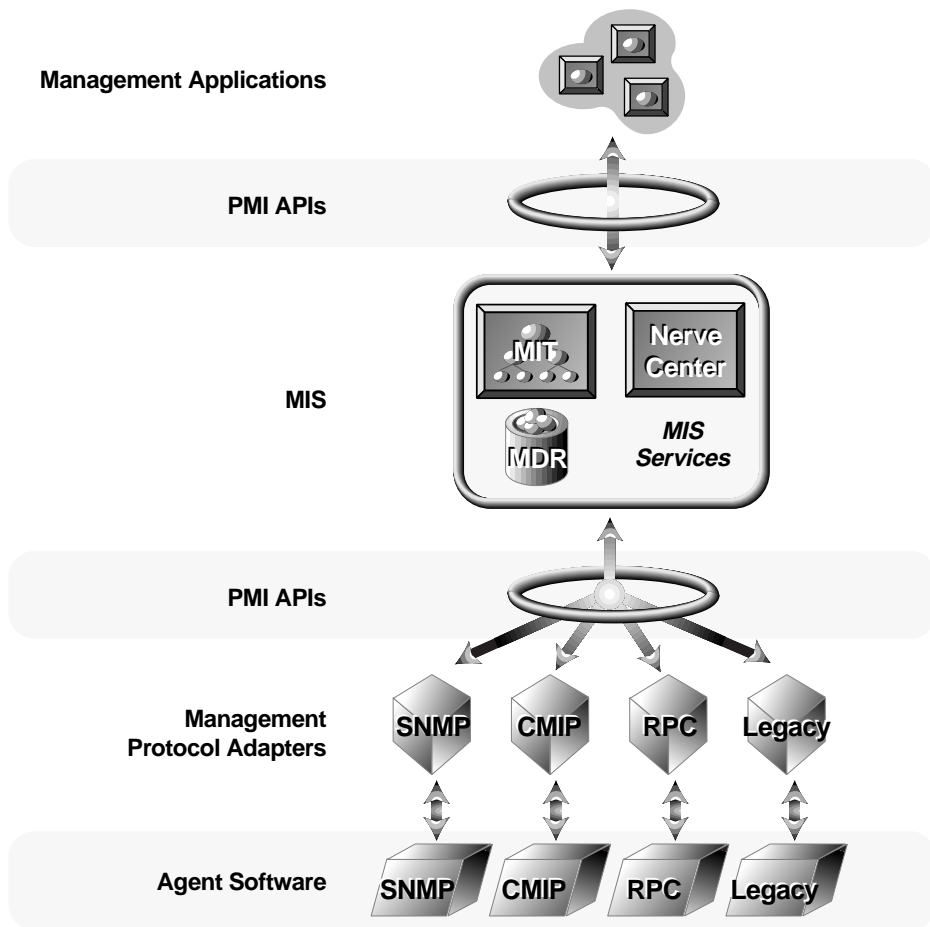


FIGURE 1-3 Overview of Management Information Servers

1.4.3.1 More About MIS Databases

A Solstice EM MIS database comprises two primary components:

1. **Metadata Repository (MDR)**—maintains information about managed object attributes and properties; the specific configuration settings used by network components. This data encompasses everything from the syntax required to refer to an attribute to the composition of an object package. The language used to describe network components is the Guidelines for the Definition of Managed Objects (GDMO), outlined in the ITU X.722 ISO/IEC 10165-4 standard. The MIS allows for dynamic updates to the MDR.

2. Management Information Tree (MIT)—a tree-like, hierarchical representation of the relationships between network components. For example, a network might have subnetwork branches, from which routers, switches, and hubs are parents to gateways, workstations, and printers. The MIS supports dynamic creation, maintenance, and deletion of objects in the MIT.

1.4.3.2 More About the MIS Nerve Center

The MIS Nerve Center provides the logic and methods to interrogate agent software for information about network components, receive notifications from agents, and initiate actions based on network conditions.

Nerve Center activities are based upon user-defined *request templates*, which are rules-based configuration files that tell Solstice EM how to interrogate for network component information, and how to respond to the information received. In this context, the term *request* should not be confused with the common object-oriented use of the term as it relates to `GET` and `SET` methods; a Nerve Center request is a set of programmatic rules, whereas an object-oriented request refers to a specific set of methods for manipulating object properties.

The Solstice EM Advanced Requests tool, and the Simple Requests function in Network Views, provide advanced and simple tools, respectively, for viewing, creating, and managing Nerve Center Request templates. The Advanced Requests tool is described in this Guide; Simple Requests are described in Chapter 3 of *Managing Your Network*.

1.4.3.3 More About PMI and MPAs

Although there are potentially many interfaces to Solstice EM, only one is required by the Solstice EM architecture. This primary interfaces is the high-level Portable Management Interface (PMI). The PMI defines the management protocol, services, and transport mechanisms for all Solstice EM components. Specifically, the PMI provides the following features:

- Protocol-independent management applications—management applications can communicate with managed objects via the PMI regardless of the protocol used by the managed object.
- Distributed, multi-user access—multiple users can access and modify objects and object definitions in the MIS.
- Proxy agent support—proxy agents can use the Management Protocol Adapter (MPA) library to access managed objects over protocols other than CMIP or SNMP.

MPAs translate information between managed objects and the MIS. For example, if you have a network component that uses SNMP, then the SNMP MPA receives data from an SNMP agent, translates the data into the PMI, and sends the data to the MIS.

Solstice EM includes the following three MPAs:

- CMIP—Common Management Information Protocol
- SNMP—Simple Network Management Protocol
- RPC—SunNet/Site/Domain Manager Remote Procedure Call

Solstice EM also supports other MPAs; see Chapter 11 in *Developing C++ Applications* for information about adding MPAs to your Solstice EM environment.

1.4.3.4 MIS Ancillary Services

In addition to the MIS database and Nerve Center, an MIS provides ancillary services that make information about managed objects available and modifiable by network management applications and agent software. These services are invoked at MIS startup with the `em_services` command.

Note – Solstice EM MIS services are usually run on one or more of the network's workstations. It is not necessary to dedicate a machine to running MIS services, although Solstice EM requires at least one MIS to be running somewhere on the network. As mentioned previously, you can run multiple MISs on a network

Specifically, the Solstice EM MIS provides the following ancillary services:

- Protocol and location transparency—a network management tool provides a managed object name, and the Solstice EM services determine what protocol to use to access the object, and what address to use within that protocol. This means that management clients do not need to know where an object is physically located or what protocol to use to communicate with it.
- Coordinate requests from multiple tools—Solstice EM supports concurrent access to MIS data from multiple network management clients.
- Persistent storage—data in the MIS database is not lost when an MIS host is shut down.
- SNMP traps and events—event handling for SNMP traps and events, and a registry mechanism that allows applications and objects to register interest in an event.
- Log management services—keep track of what is happening where on the network. Solstice EM provides detailed logging services, which are maintainable through Solstice EM's Event Logs tool.

1.4.3.5 More About MIS Data Access

One of the primary functions of an MIS is to make managed object data available to network management tools and agent software. A related function is to make this data transparently available across multiple MIS machines on multiple subnetworks.

In Solstice EM, all managed objects are accessed through the MIT. There is one global tree, with a single naming scheme for all data. The MIT is constructed according to the rules provided by OMNIPoint™, and has a single `root` branch. The shape of the tree descended from this root branch is arbitrary, and may vary widely from one MIS to another. The tree's structure is hierarchical and based on parent/child containment relationships; below any given object are one or more related subordinate objects.

A Solstice EM MIS makes MIT data accessible both through the naming conventions that come from the managed resources it describes, and from a resource-independent naming convention, in which identifiers are specified using Fully Distinguished Names (FDNs). FDNs enable Solstice EM MIS to provide transparent access to managed objects.

Managed objects in the Solstice EM environment may be stored *locally*, in an MIS database. However, Solstice EM also supports access to *remote* objects that are stored outside a given MIS. When representing a remote object, a Solstice EM MIS stores information about that object's physical location, along with a description of the address and protocol required to access that object. When getting or setting properties for a given object, Solstice EM handles the address and protocol resolution transparently, without user intervention, whether the object is local or remote to the MIS.

1.4.3.6 More About Object Orientation

As mentioned previously, Solstice EM MIS services are based on an ISO-standard ASN.1 (Abstract Syntax Notation One) object-oriented architecture. Specifically, Solstice EM:

- Describes managed objects in terms of OMNIPoint and ISO terminology
- Uses C++ objects for internal storage and manipulation of network data

The following list describes some object-oriented terms as they relate to Solstice EM.

- **Class**—C++ classes, as described in the *Annotated C++ Reference Manual* (Margaret Ellis and Bjarne Stroustrup; Addison-Wesley Publishing Co., Reading, MA; copyright 1990 by AT&T Bell Laboratories).
- **Instance**—memory that is allocated for a C++ class according to its definition. A variable name is usually associated with a particular instance of a class.

- Managed resource—an actual physical device or entity that exists in a network or system. This is consistent with the OSI/Network Management Forum (NMF) definition of the term.
- Managed Object (MO)—is a set of services and attributes that describes a type of managed resource. This is also consistent with the OSI/NMF definition of the term.
- Managed Object Class (MOC)—within the Solstice EM development environment, this term refers to the internal representation of a managed object, as described by the OSI/NMF. An MOC is based on the GDMO description, which could itself be a translation of an SNMP MIB or SunNet Manager schema description—that is, the MOC represents the attributes and behaviors for particular types of manageable objects. The MOC defines the type of data stored and the behaviors that can be taken, but does not represent the actual data for any managed object. The MOC is the internal representation used by the Solstice EM MIS.
- Managed Object Instance (MOI)—relates to a managed object class in the same way as an instance relates to a class, where the MOC determines the type of attributes and behaviors available to operate on an object of this type, and MOI refers to actual data that represents an object being managed by the MIS. An MOI is also an internal representation used by the Solstice EM MIS.

1.4.4 Network Management Protocols

In order to be able to pass management data from component to agent to MIS to management tool and back again, all components in the environment must agree on and understand how and what data is being exchanged. The rules defining such exchanges are referred to as *protocols*.

As mentioned previously, Solstice EM is based on the ISO-standard agent/station network management model, and supports the following common protocols:

- CMIP—Common Management Information Protocol
- SNMP—Simple Network Management Protocol
- RPC—SunNet/Site/Domain Manager Remote Procedure Call

Solstice EM also provides full conformance with Telecommunications Management Networks (TMN) standards.

SNMP and RPC are network management protocols used to manage resources in a TCP/IP network environment. CMIP is the protocol used in ISO networks. Both protocols specify ASN.1 as the language used to encode and decode object request and response messages.

Note – When you install Solstice EM, you are asked if you want support for IP management, CMIP management, or both. The choice you make depends on the types of devices on your network, and the network management protocols they support.

1.4.4.1 More About RPC

Solstice EM includes a suite of agents developed for the SunNet/Site/Domain Manager (SNM) platform. These agents communicate with Solstice EM using RPC protocols, and are included primarily to provide a migration path from SNM installations to Solstice EM.

In addition to migrating to Solstice EM, however, SNM agents communicating over RPC can be a useful part of your network management strategy. Specifically, SNM agents can act as *proxies* for communicating via older protocols.

1.4.4.2 About MIBs

Part of the SNMP protocol includes attribute definition files called MIBs (Management Information Base). MIBs define:

- Attributes or types of data that can be supplied by an agent to a manager
- Actions performed by an agent that can be requested by a manager
- Behavior exhibited by the agent
- Notifications—unsolicited information the agent can send to a manager

The ISO standards organization defines a MIB in ISO/IEC 7948-4 as follows: “The conceptual repository of management information within an open system.” A network management package normally contains management information describing each type of agent the manager is capable of managing. This information typically includes Internet MIB definitions and ISO GDMP definitions for managed objects and agents. An agent typically presents or contains management information for one type of device, although this information can include descriptions and data for several types of devices.

1.5 Solstice EM APIs

In addition to the network management tools shipped with Solstice EM, you can define your own tools and applications to work with Solstice EM through the Application Programming Interfaces (APIs) included with Solstice EM.

If an application developer also provides access to a managed object type not previously known to the MIS, the MIS must be informed of the Managed Object Class through the use of GDMO ASN.1 documents.

The Solstice EM APIs provide a rich set of functions for the application programmer. The libraries, written in C++, contain the objects and methods necessary to communicate with the MIS and obtain information about the managed resources it controls. The APIs provide the following services:

- Initialization, including establishment of a distributed message passing interface to the MIS
- Event subscription
- Remote caching and cache control
- Local object cache management for applications
- Encoding and decoding of parameters into ASN.1
- Encoding and encapsulation of data into a format (message class) passed to the MIS

1.5.1 API Modules

The Solstice EM APIs consist of several different groupings or modules, as summarized in the following table. Refer to the *Developing C++ Applications* and *Developing Java Applications* for detailed information about developing applications that interact with Solstice EM.

TABLE 1-2 Solstice EM API Modules

API Module	Description
High-level PMI (Programming Management Interface)	For most applications, all interaction with the MIS can be handled through the high-level protocol-independent functions of the PMI. These functions hide the encoding/decoding of ASN.1 values, and provide CMIS-like messages used in communication with the MIS and managed objects (through the MIS). It also provides for initialization and for event subscription and propagation.
Low-level PMI	Used to exchange messages between the Solstice EM MIS and client services using CMIS-like messages (M-GET, M-SET, M-CREATE, M-DELETE, M-EVENT-REPORT, M-CANCEL-GET, and M-ACTION).
Application-to-Application API	Allows applications to send messages to other Solstice EM applications through an <code>emApplicationInstance</code> object.
Grapher API	Allows developers to create graphical representations of data.

TABLE 1-2 Solstice EM API Modules *(Continued)*

API Module	Description
Nerve Center Interface (NCI) Library	A programmatic interface for controlling Nerve Center operation. The NCI library enables applications to create, edit, and launch Nerve Center requests.
Object Services API	Allows developers to access services provided by the Solstice EM MIS to implement intra-object behaviors or specialized behaviors.
Topology API	Allows developers to create applications for the Solstice EM environment without learning the details of the MIT topology naming tree.
Viewer API	Allows applications to communicate with the Solstice EM Network Views tool to control specific Network Views features. This allows developers to leverage Network Views functionality and integrate their applications with Network Views.

1.5.2 Application Development Support Tools

In addition to the API classes and methods described in the preceding table, Solstice EM includes the following application development support tools:

- **Object Development Tools (ODT)**—provide a simple and automated framework developers can use to add and write behaviors for managed objects that reside in the MIS. Refer Chapter 10 in *Developing C++ Applications* for detailed information.
- **Compilers**—Solstice EM includes several compilers that developers and, in some cases, system administrators need to use. The compilers provide a means by which you can add new managed object definitions to the MIS. Solstice EM includes the following compilers:
 - **ASN.1 (em_asn1)**—Compile descriptions of managed objects into the MDR. These descriptions are provided as ASN.1 documents.
 - **GDMO (em_gdmo)**—Compile new GDMO object descriptions and then add them to the MDR.
 - **Concise MIB (em_cmib2gdmo)**—Convert object descriptions written in Concise MIB format to GDMO for use in Solstice EM.
 - **Schema (em_snm2gdmo)**—Convert object descriptions written in SunNet Manager schema format to GDMO for use in Solstice EM.
 - **em_debug**—Solstice EM includes a dynamic debugging function that helps you track information going in to and out of the MIS.

1.6 Related Reading

Depending on what you want to do in Solstice EM, one or more of the following Solstice EM manuals may interest you:

- *Installation Guide*—complete instructions on installing all Solstice EM components and utilities, including agent software.
- *Managing Your Network*—information about using the Solstice EM tools to perform day-to-day management tasks.
- *Customizing Guide*—detailed information about advanced Solstice EM configuration and customization.
- *Management Information Server (MIS) Guide*—advanced information about MIS servers.
- *Developing C++ Applications*—information for developers who want to write custom applications that can be integrated with Solstice EM.
- *Troubleshooting Guide*—solutions to problems you encounter with Solstice EM.
- *CORBA Gateway Administration Guide*—provides information on installing, configuring, and using SEM CORBA ToolKit to build and package the SEM CORBA Gateways.
- *Glossary*—a glossary of network management terms as they relate to Solstice EM.

1.7 Solstice EM Tools—Complete Listings

The remainder of this chapter provides a quick-reference table that briefly describe all of the Solstice EM network management tools, along with pointers to the books in which you can find more detailed information.

Refer to the preceding sections in this chapter for information about general Solstice EM network management concepts, and explanations of how the major Solstice EM tools work together.

The following table lists the Solstice EM tools, sorted by binary name. All tools listed here are located by default in `/opt/SUNWconn/em/bin` (`$EMHOME/bin`).

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name

Binary Name	Tool Name	Description	More Information
<code>build_oid</code>	Build Object IDs	SunNet Manager (SNM) SNMP utility; creates an Object ID Database (<code>/var/usr/SUNWconn/snm/oid.dbase</code>).	<i>This Guide</i>
<code>build_tt</code>	Build Textual Convention Types	Builds the textual convention types database; used by the <code>mib2schema</code> utility to look for new IMPORT definitions local to a MIB.	<i>Developing C++ Applications</i>
<code>create_admin</code>	***	***	***
<code>db_services</code>	Initialize Database Services	Initializes MIS database services; usually run as part of the <code>em_services</code> command.	<i>Management Information Server (MIS) Guide</i>
<code>em</code>	Network Tools	Primary Solstice EM tools window; Solstice EM and custom network tools can be started, added, and removed from this window.	<i>Managing Your Network</i>
<code>em_accesscmd</code>	Access Control – command line interface	Command-line Access Control tool; lets you manage user access to Solstice EM tools and objects.	<i>Managing Your Network</i>
<code>em_accessmgr</code>	Access Control – graphical interface	Graphical Access Control tool.	<i>Managing Your Network</i>
<code>em_add_db_server</code>	Add Database Server	Lets you specify an MIS database server to use instead of or in addition to the default.	<i>Management Information Server (MIS) Guide</i>
<code>em_admintool</code>	Administration	Access advanced administration functions; for example, MIS connections and parameters, database backup and restore, request template designer, etc.	<i>This Guide</i>
<code>em_alarmmgr</code>	Alarms	View and manage object alarms.	<i>Managing Your Network</i>
<code>em_asn1</code>	ASN.1 Compiler	ASN.1 object compiler.	<i>Management Information Server (MIS) Guide</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
em_autod	Automatic Management Daemon	Automatic management daemon; monitors creation and deletion of MIS objects; starts and stops requests when objects are added and deleted.	<i>Management Information Server (MIS) Guide</i>
em_autoexd	Database Table Extender	MIS daemon that automatically extends database tables when they get full.	<i>Management Information Server (MIS) Guide</i>
em_automgr	Request Controllers	Automatic management configuration tool; used in conjunction with em_autod.	<i>Management Information Server (MIS) Guide</i>
em_auxdb	***	***	<i>This Guide</i>
em_clear_alarms	Clear Alarms – command line interface	Command-line utility for clearing alarms against a specified range of toponodes.	<i>Managing Your Network</i>
em_cmib2gdm	Concise MIB Compiler	Converts MIB files in Concise MIB format to GDMO and ASN.1 formats.	<i>Management Information Server (MIS) Guide</i>
em_cmip	CMIP Management Protocol Adapter	Implements CMIP MPA functions; starts during product installation.	<i>Management Information Server (MIS) Guide</i>
em_cmipautoreg	CMIP Autoregistration	Automatically registers CMIP agents in the MIS.	<i>Management Information Server (MIS) Guide</i>
em_compose_all	Compose/Load GDMO Bindings	Compose and load all name bindings from a GDMO file.	<i>Management Information Server (MIS) Guide</i>
em_compose_oc	Instantiate Volatile Class	Instantiates a new MIS object class with volatile data storage.	<i>Management Information Server (MIS) Guide</i>
em_compose_poc	Instantiate Persistent Class	Instantiate a new MIS object class with persistent data storage.	<i>Management Information Server (MIS) Guide</i>
em_datacollector	Data Collections	Create and manage data collection entry objects, and display data gathered from requests.	<i>Managing Your Network</i>
em_datad	Data Collection Daemon	Daemon used in conjunction with em_datacollector.	<i>Managing Your Network</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
em_dataviewer	RPC/CMIP Data	View and manage RPC, CMIP, and SNM managed objects.	<i>Management Information Server (MIS) Guide</i>
em_db_abort	Abort Database	Script to abort an Informix database on the specified MIS.	<i>Management Information Server (MIS) Guide</i>
em_db_create	Create Database	Script to create a new Informix database on the specified MIS	<i>Management Information Server (MIS) Guide</i>
em_db_drop	Drop Database Tables	Script to drop log or non-log tables from an Informix database on the specified MIS.	<i>Management Information Server (MIS) Guide</i>
em_db_start	Start Database Server	Script to start a database server on the specified MIS.	<i>Management Information Server (MIS) Guide</i>
em_db_stop	Stop Database Server	Script to stop the database server on the specified MIS.	<i>Management Information Server (MIS) Guide</i>
em_dbarchive	Database Backup/ Restore	Graphical administration tool for backing up and restoring MIS databases.	<i>Management Information Server (MIS) Guide</i>
em_dbbackup	Database Backup – command-line interface	Command-line interface for MIS database backup.	<i>Management Information Server (MIS) Guide</i>
em_dbrestore	Database Restore – command-line interface	Command-line interface for MIS database restore.	<i>Management Information Server (MIS) Guide</i>
em_debug	Solstice EM Debugging Tool	Command-line debugging tool that supports the Solstice EM remote dynamic debugging feature.	<i>Cooperative Consoles Administration Guide</i>
em_discover	Network Discovery	Discover components on your network and update the MIS database.	<i>This Guide</i>
em_eds	Event Distribution System	Distributes events from event sources to event listeners.	<i>Management Information Server (MIS) Guide</i>
em_gdmo	GDMO Compiler	GDMO compiler; lets you extend Solstice EM capabilities with new GDMO object class descriptions.	<i>This Guide</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
em_grapher	Grapher	Create graphs from alarm and agent data.	<i>Managing Your Network</i>
em_help	Help	Solstice EM help system.	<i>Online Help</i>
em_imex	Log File Import/ Export	Import and export log objects from and to ASCII files.	<i>Managing Your Network</i>
em_java2gdm	Java GDMO Compiler	Compile Java classes in GDMO format.	<i>This Guide and Developing Java Applications</i>
em_jdmk_config	Configure Java Agents	Configure Solstice EM to work with Java-based agents.	<i>This Guide and Developing Java Applications</i>
em_jdmk fwd	Forward Java Events	Forwards Java events to the Solstice EM MIS for processing.	<i>Management Information Server (MIS) Guide</i>
em_layout	Network Views Layout	Define how object views are laid out in Network Views; commonly invoked from Network Views.	<i>Managing Your Network</i>
em_load_name_bindings	Load Name Bindings	Loads object name bindings; allows the MIS to resolve MIT object names.	<i>Management Information Server (MIS) Guide</i>
em_load_nc_templates	Load Nerve Center Templates	Script that invokes the <code>em_ncimport -file</code> command; loads Solstice EM Nerve Center templates.	<i>Management Information Server (MIS) Guide</i>
em_loaddefs	Load Object Definitions	Load MIB, GDMO, and SunNet Manager schema files into the MIS.	<i>Management Information Server (MIS) Guide</i>
em_log	Initialize Log Server	Initializes log server MPA functions.	<i>This Guide</i>
em_log2hist	Save Log Entries	Saves logs in history files.	<i>Managing Your Network</i>
em_log2rdb.ifmx	Transfer Log Histories Daemon – Informix	Daemon that reads and transfers log histories to an Informix database.	<i>Managing Your Network</i>
em_log2rdb.orcl	Transfer Log Histories Daemon – Oracle	Daemon that reads and transfers log histories to an Oracle database.	<i>Managing Your Network</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
em_log2rdb.sybs	Transfer Log Histories Daemon – Sybase	Daemon that reads and transfers log histories to an Sybase database.	<i>Managing Your Network</i>
em_login	Login Daemon	Listens for Solstice EM connection requests for password authentication.	<i>Management Information Server (MIS) Guide</i>
em_logmgr	Event Logs	Create, modify, and delete log objects.	<i>Managing Your Network</i>
em_logview	View Logs	View logs and log objects.	<i>Managing Your Network</i>
em_mis	MIS Services Core	Primary Solstice EM services; commonly invoked from the em_services command.	<i>Management Information Server (MIS) Guide</i>
em_mismgr	MIS Manager	Manage MIS parameters and connections.	<i>Management Information Server (MIS) Guide</i>
em_mpa_jdmk	Initialize JDMK MPA functions.	Initializes JDMK MPA functions on the MIS.	<i>Management Information Server (MIS) Guide</i>
em_mpa_rpc	Initialize RPC MPA functions.	Initializes RPC MPA functions on the MIS.	<i>Management Information Server (MIS) Guide</i>
em_mpa_snmp	Initialize SNMP MPA functions.	Initializes SNMP MPA functions on the MIS.	<i>Management Information Server (MIS) Guide</i>
em_ncam	Nerve Center Daemon	Daemon that handles nerve center actions, such as sending email or executing UNIX commands.	<i>This Guide</i>
em_ncexport	Nerve Center Export	Export Nerve Center templates to an ASCII file.	<i>This Guide</i>
em_ncimport	Nerve Center Import	Import Nerve Center templates that have been previously exported with em_ncexport.	<i>This Guide</i>
em_nnadd	Global Nickname Service Addition	Enable a global nickname server on the MIS; generally started with em_services.	<i>Developing C++ Applications</i>
em_nnconfig	Global Nickname Configuration	Populate the global nickname translation server.	<i>Developing C++ Applications</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
em_nnmpa	Global Nickname Service Daemon	Starts the global nickname translation server.	<i>Developing C++ Applications</i>
em_ns_server	Debug Nickname Services	Debug global nickname translation services.	<i>Developing C++ Applications</i>
em_obcodegen	Object Code Generator	Used to generate object behavior code from GDMO and ASN.1 definitions.	<i>Developing C++ Applications</i>
em_obed	Object Editor	View and edit objects in the MIT; see also em_oct.	<i>This Guide</i>
em_objop	Object Operations Utility	Command-line utility for sending CREATE, SET, DERIVE, and DELETE requests; primarily used by init_platform to create and modify MIS objects at MIS startup.	<i>Management Information Server (MIS) Guide</i>
em_oct	Network Views Object Configuration Tool	View, modify, and create managed objects; commonly run from a Network Views window; see also em_obed.	<i>Managing Your Network</i>
em_panel	Network Tools	Starting point for Solstice EM general network management applications; usually invoked with the em command and em -host	<i>Managing Your Network</i>
em_purged	Alarm Deletion Daemon	Periodically deletes alarms based on severity, stat, time, etc.	<i>Managing Your Network</i>
em_purgemgr	Alarm Deletion Controller	Sets up conditions for em_purged to delete alarms.	<i>Managing Your Network</i>
em_reqedit	Request Template Designer	View, modify, create, and delete Nerve Center request templates.	<i>This Guide</i>
em_restart	Restart MIS Services	Restart MIS services.	<i>Management Information Server (MIS) Guide</i>
em_services	Initiate MIS Services	Initiate MIS services; starts the MIS server and several related daemons.	<i>Management Information Server (MIS) Guide</i>
em_simplerequests	Network Views Simple Requests	View, modify, and create simple MIS request templates; generally invoked from Network Views.	<i>Managing Your Network</i>
em_snm2gdm	SNM Schema to GDMO Compiler	Convert SNM schema files to GDMO descriptions.	<i>This Guide</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
em_snm_type_import	Import SNM Object Types	Import SNM object types into the Solstice EM environment.	<i>This Guide</i>
em_snmdb_import	Import SNM Topology	Imports SNM topology databases into an Solstice EM MIS.	<i>Management Information Server (MIS) Guide</i>
em_snmfwd	Forward SNM Events	Forwards SNM events to the Solstice EM MIS for processing.	<i>Management Information Server (MIS) Guide</i>
em_snmp-trap	Listen for SNMP Traps Daemon	Daemon that listens on port 162 for SNMP traps.	<i>Management Information Server (MIS) Guide</i>
em_snmpbrowser	SNMP Data	Get, set, view, and modify SNMP agent attributes.	<i>Managing Your Network</i>
em_sql	Log In to SQL Database	Login process for SQL databases.	<i>This Guide</i>
em_srm	***	***	***
em_startup	Start MIS Server	Start the MIS server; usually invoked as part the <code>em_services</code> command.	<i>Management Information Server (MIS) Guide</i>
em_topo_args	Modify Toponodes	Command-line interface for modifying topoNode objects.	<i>This Guide</i>
em_topoimex	Topology Import/Export	Import or export MIT topology information.	<i>Management Information Server (MIS) Guide</i>
em_topoimex_BC	Convert Legacy Topology Information	Converts topology information from prior versions of Solstice EM.	<i>This Guide</i>
em_trapd	Initialize Topology Services	Initializes topology services on the MIS; commonly invoked as part of <code>em_services</code> .	<i>This Guide</i>
em_viewer	Network Views	View, organize, modify, and create managed objects.	<i>Managing Your Network</i>
emenv.csh	Environment Configuration – C Shell	Source this file to configure Solstice EM environment variables, such as <code>\$EMHOME</code> , license server, etc.; C Shell environment only.	<i>Managing Your Network</i>
emenv.sh	Environment Configuration – Korn/Bourne Shell	Source this file to configure Solstice EM environment variables, such as <code>\$EMHOME</code> , license server, etc.; Korn or Bourne shell.	<i>Managing Your Network</i>

TABLE 1-3 Solstice EM Tools – Complete List, Sorted by Binary Name *(Continued)*

Binary Name	Tool Name	Description	More Information
get_local_host	Find Local MIS Host	Display the name of the current local MIS host.	<i>This Guide</i>
hyperhelp	Help Viewer	Bristol HyperHelp™ Viewer; Solstice EM online help is in HyperHelp format.	<i>Managing Your Network</i>
jme_jre	JME Services helper script	Helper script for jme_services; do not invoke directly.	<i>Management Information Server (MIS) Guide</i>
jme_services	JME Services	Start and stop Solstice EM Java daemons	<i>Management Information Server (MIS) Guide</i>
mib2schema	SNMP MIB to SNM Schema compiler	Convert SNMP MIB files to SNM scheme format.	<i>This Guide</i>
snm_br	Results Browser	List RPC, CMIP, or SNMP data collected with the Data Collections tool.	<i>This Guide</i>
snm_cmd	Manage SNM Agents – command-line interface	Command- line manager for Site/ SunNet/Domain Manager agents.	<i>This Guide</i>
snm_cmdtool	Run Command from SNM Session	Run a UNIX command from an SNM session.	<i>This Guide</i>
snm_exec	Execute command via EVAL from SNM	Uses the Bourne shell's eval to execute a UNIX command from an SNM session.	<i>This Guide</i>
snm_gr	Grapher	Display and graph SNM data.	<i>This Guide</i>
snm_kill	Stop SNM Agent Requests	Stop one or more SNM agent requests.	<i>This Guide</i>
snm_version	SNM Version Information	Displays information about the current version of SunNet Site/ Domain Manager software.	
v2mib2schema	SNMP2 MIB to Schema Compiler	Convert SNMP2 MIBs to SNM schema format.	<i>This Guide</i>
var-install	Install Solstice EM Packages on Remote	Install Solstice EM packages on another machine.	<i>Installation Guide</i>
var-obj-install	Install Solstice EM object Definitions	Install Solstice EM object definitions.	<i>This Guide</i>

Network Management and the Solstice EM Architecture

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 (Solstice EM) is a distributed, multi-user management platform, with a set of user tools, that allows you to accomplish the network management goals. This chapter describes the key aspects of the Enterprise Manager architecture.

This chapter describes the following topics:

- Section 2.1 “The Agent/Manager Model” on page 2-1
- Section 2.2 “Client/Server Architecture” on page 2-2
- Section 2.3 “Distributed Management” on page 2-4
- Section 2.4 “Network Management Protocol Support” on page 2-7
- Section 2.5 “Simple Requests” on page 2-11
- Section 2.6 “Object Classes and Event Notification Types” on page 2-18

2.1 The Agent/Manager Model

Solstice EM is based on the agent/manager 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” which could be an

NFS server such as a hub, a cellular base station, or a WAN link; or components such as a circuit, or a router interface, or software entities such as tool 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 by their initiative when they detect predefined thresholds or events on a managed resource. Agent/manager communication is illustrated in the following figure.

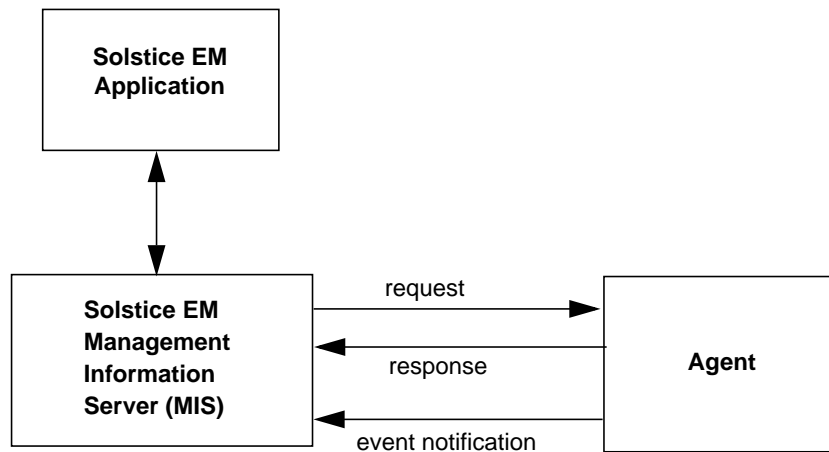


FIGURE 2-1 Agent/Manager Communication in Solstice EM Environment

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

2.2 Client/Server Architecture

The management functionality of Solstice EM is based on the client/server architecture. Solstice EM is shipped with a set of tools to carry out network management tasks. For example, the Solstice EM Network Views window provides a graphical, dynamically updated display of your network topology. Colored icons indicate the fault status of devices displayed in the Network Views window. The Network Views Request window allows users to launch Nerve Center requests one-at-a-time to monitor devices for the occurrence of critical events.

In addition, Solstice EM 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 numerous devices.

Another key tool, Alarms window, allows you to view and sort incoming alarms and acknowledge or clear them. (The Alarms window, Auto Manager, and other Solstice EM core tools are documented in *Managing Your Network*.)

These and other user tools may be installed on machines remote from the machine that runs the MIS. Multiple users, running Solstice EM tools on one or more workstations, may be connected to the same MIS. You have access to most of the Solstice EM tools from the Solstice EM Network Tools, shown in the following figure. The launcher can be configured to include other tools in addition to those shipped with Solstice EM.

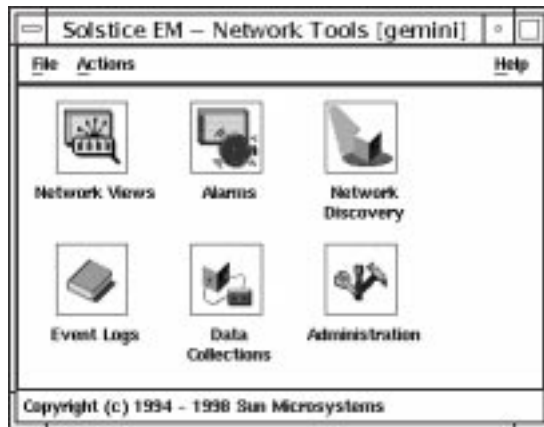


FIGURE 2-2 Solstice EM Network Tools

Solstice EM tools that are installed on the same machine as the MIS can be displayed remotely by means of an X windows session. Same machine installation differs from the installation of Solstice EM tools on a remote machine. In the latter case, Solstice EM tools connect to the MIS using a PMI connection. In general, tools running on a remote machine consume far less network bandwidth than tools that are run on the MIS machine and displayed remotely.

The multi-user capabilities of Solstice EM are based on Solstice EM's ability to provide consistent management information to components of the network management solution—operators, tools, and other management stations. Solstice EM enables management tasks to be divided across geography and organization with

confidence that all users will see the same view of management data. This universal view is particularly useful in fault management scenarios where cooperation among staff members leads to prompt resolution of problems.

2.3 Distributed Management

The powerful Solstice EM platforms the ability to distribute the management information base to multiple Management Information Servers while allowing transparent access management data to users irrespective of the MIS on which the data is located. The data may reside in the local MIS to which the tools are connected or in a remote MIS in another geographical locale.

The Solstice EM MIS Connection Tool 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.

Access to all objects is achieved through the Management Information Tree (MIT). The MIT is the globally defined object naming or containment tree as defined in the ITU-T X.700 series standards. Every object has a name that distinguishes it from every other object in any MIS. The globally unique name of any object is its full path name from global root (the top of the naming hierarchy) to its position in the tree—analogue to the absolute path to a file in a UNIX file system.

When a connection is initiated from MIS A to MIS B, the local MIT of MIS B is “mounted” into MIS A—and becomes visible in the Navigator of a Network Views window connected to MIS A.

The user running the Network Views window connected to MIS A then has access to the views and devices represented in MIS B. These devices become manageable from the local MIS A. 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 B. Whether the request is running on the local MIS or a remote MIS is transparent to the MIS A user.

Many of the tools shipped with Solstice EM have this ability to access managed resources via MIS-to-MIS connections. However, there are some tools (such as Design Advanced Request window, Security window, and Network Discovery window) that only access data in the local MIS.

The example in the following figure 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 arrow direction from MIS A to MIS Net_B indicates that the MIS-to-MIS connection was initiated from MIS A.

The Network Views window connected to MIS A will see a topology like that shown in FIGURE 2-4. The topology tree for the Network Views window connected to MIS on Net_B is shown in FIGURE 2-5. The Network Views window connected to the MIS on Net_D, however, sees only the local MIS, as illustrated in the FIGURE 2-5. If the user running the Network Views window on MIS A were to select the Bldg_1_Subnet view, under the MIS Net_B local root, the Network Views window accesses the data on MIS Net_B and the user sees the same view of this subnet as a user running the Network Views window connected to MIS Net_B.

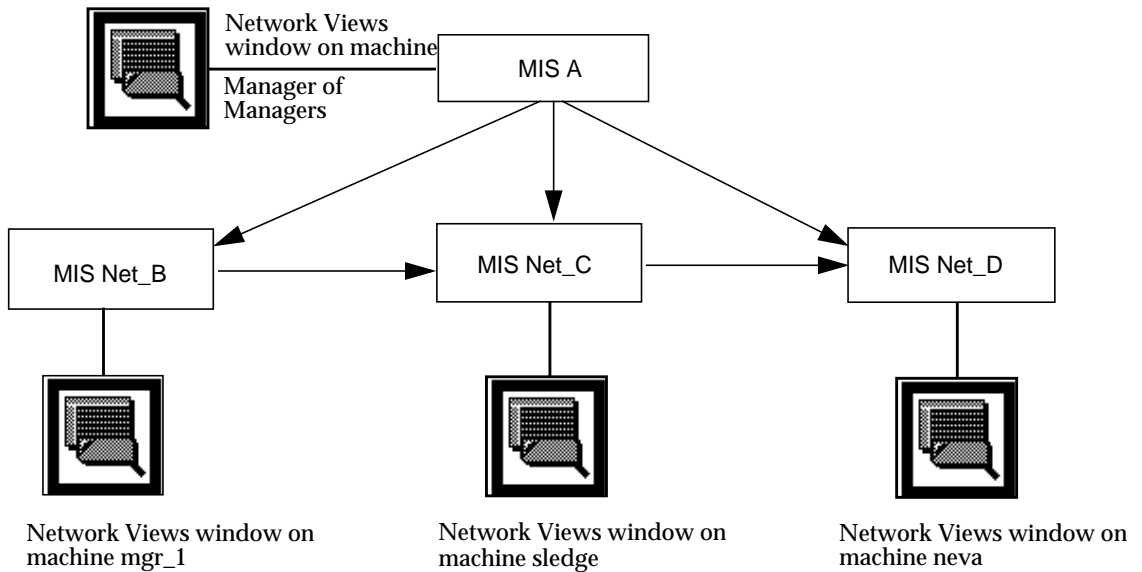


FIGURE 2-3 A Sample Configuration Using MIS-to-MIS Communication

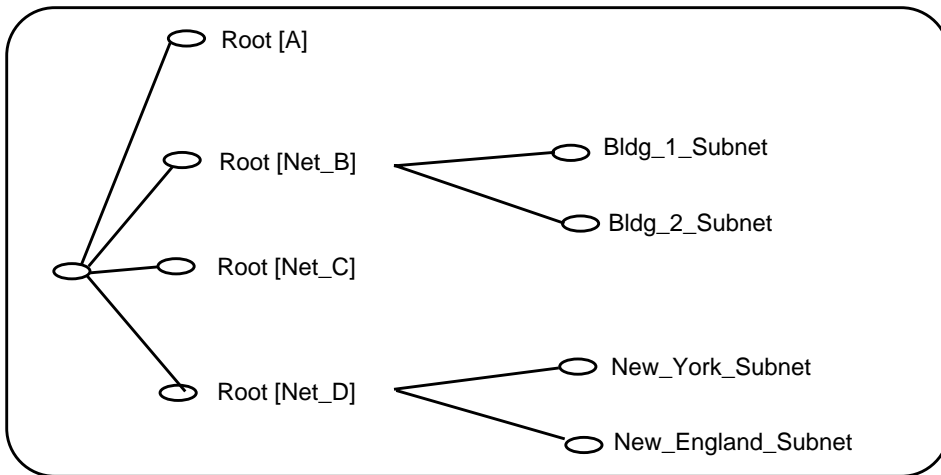


FIGURE 2-4 Topology Tree as Seen by Network Views Window Connected to MIS A

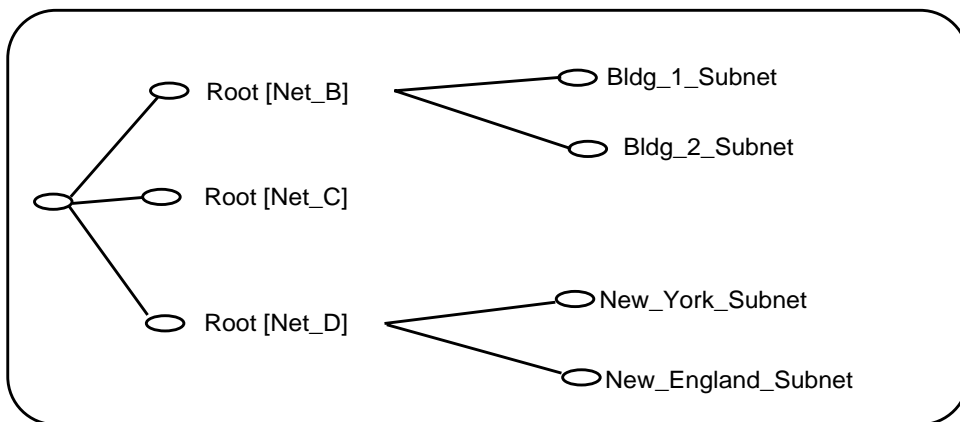


FIGURE 2-5 Topology Tree as Seen by Network Views Window Connected to MIS Net_B

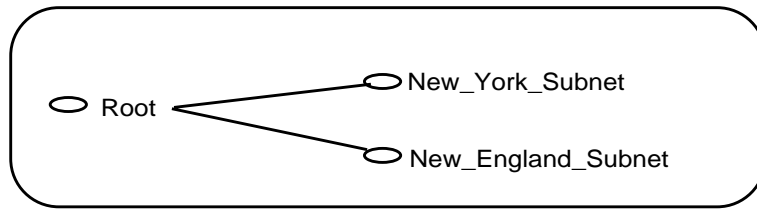


FIGURE 2-6 Topology as Seen in Network Views Window Connected to MIS Net_D

When MIS A initiates a request for data from MIS Net B, MIS A takes on the “manager role” in a MIS-to-MIS communication. As illustrated in the following figure, MIS Net B plays the agent role, responding to requests initiated by MIS A. For information about setting up MIS-to-MIS connections, refer to the *Management Information Server Guide*.

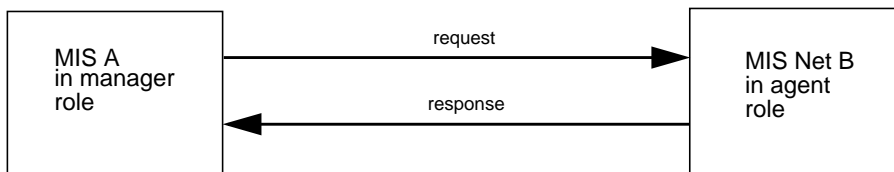


FIGURE 2-7 MIS-to-MIS Connection From MIS A to MIS Net B

2.4 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. The Solstice EM shipped to you offers support for four network management protocols:

- Simple Network Management Protocol (SNMP)
- Common Management Information Protocol (CMIP)
- Remote Procedure Call (RPC) protocol (as used by Site/SunNet/Domain Manager)
- Access to JDMK agents via Java PMI

SNMP and RPC are network management protocols used to manage resources in the context of an Internet (IP) network environment. When you install Solstice 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.

2.4.1 RPC Support

Solstice EM 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 the following figure, 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 for gathering information from other agents. The RPC Management Protocol Adapter (MPA) translates requests from management tools, such as Nerve Center requests, into appropriate SNM RPC messages, which it forwards to the RPC proxy agent. RPC responses from the agent are in turn translated from SNM RPC format into the PMI format used for messages internal to the MIS. The RPC MPA may be installed on the MIS machine or it may be distributed elsewhere in your network.

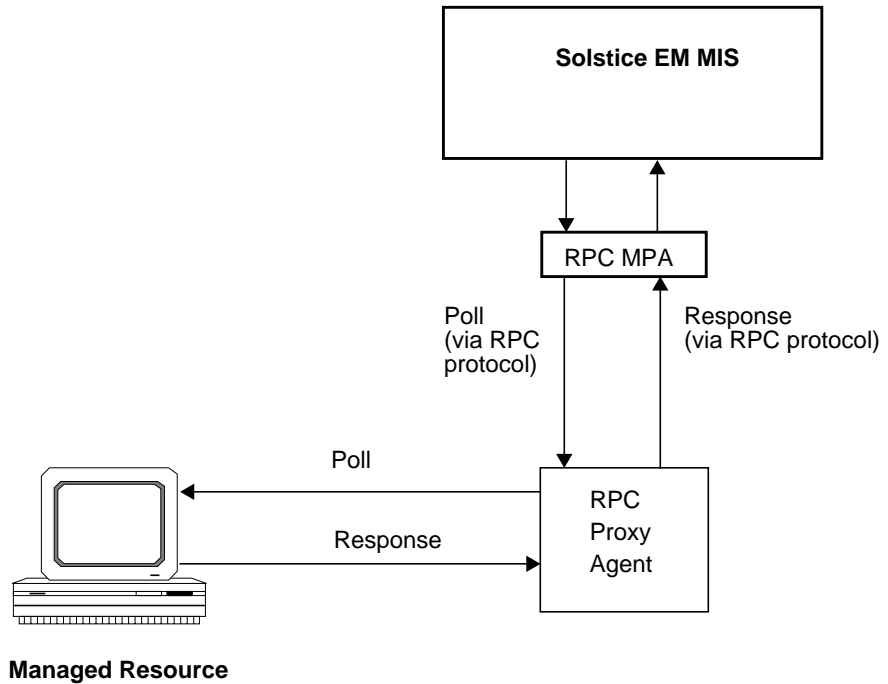


FIGURE 2-8 Polling RPC Agents

Step-by-step guidance in using RPC agents as part of your network management solution is provided in Chapter 6.

An important aspect of Solstice EM's RPC support is the ability of the Solstice 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 Solstice EM's SNM event request capability is illustrated in the following figure. It 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.

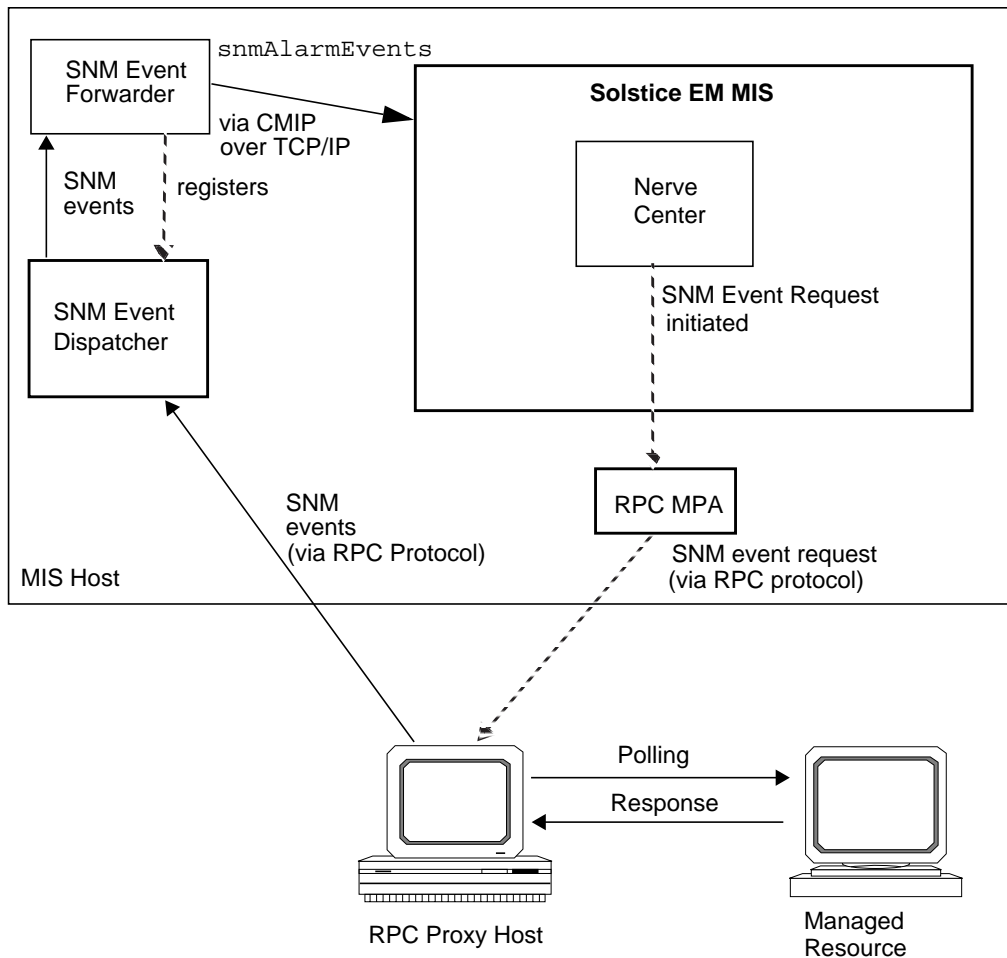


FIGURE 2-9 Using SNM Event Requests With Solstice EM

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 17.

2.5 Simple Requests

Simple Requests lets you create event requests. You can pick specific attributes and set threshold requests. When the threshold is exceeded, an event is generated. The Simple Request tool accepts one or more topology node IDs from the command line.

Simple request templates are used for monitoring:

- disk usage
- cpu usage
- router interface status

A Real Time Graphing capability is used to view data collected via the Data Collection system. The Data Collector can be invoked from the SNMP Browser and the Data Viewer.

2.5.1 SNMP Support

A key component of Solstice EM's Simple Network Management Protocol (SNMP) support is the SNMP Management Protocol Adapter (MPA). The SNMP MPA 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 the following figure.

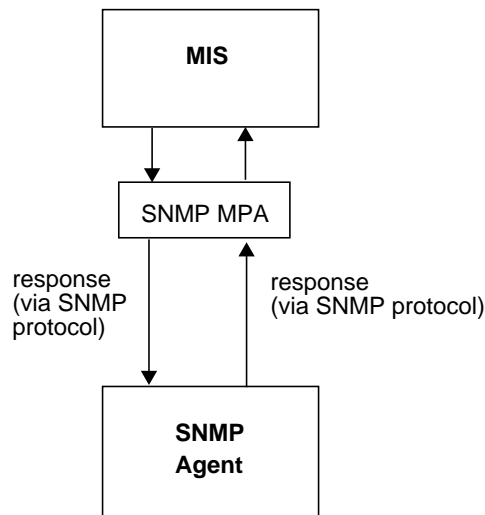


FIGURE 2-10 MIS Communication With SNMP Agents

For example, if you select a device in the Network Views window that is manageable via SNMP, and invoke Solstice EM's SNMP Data window, you can retrieve the current values of SNMP attributes or poll for selected attributes. The SNMP Data window, which connects to the MIS, sends requests for data which are translated by the MIS into SNMP requests via the SNMP MPA. The MPA may be installed on the MIS machine or distributed elsewhere in your network.

A second important aspect of Solstice EM's SNMP support is the Solstice EM SNMP trap daemon, which can be distributed to various sites in your network. SNMP agents have the ability to generate event notifications on their own initiative when certain conditions are detected; these notifications are called *traps*. The Solstice EM trap daemon listens for incoming SNMP traps and converts them to CMIP event notifications for forwarding to one or more MIS. Like other Solstice EM tools, the trap daemon uses a PMI 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 the following figure.

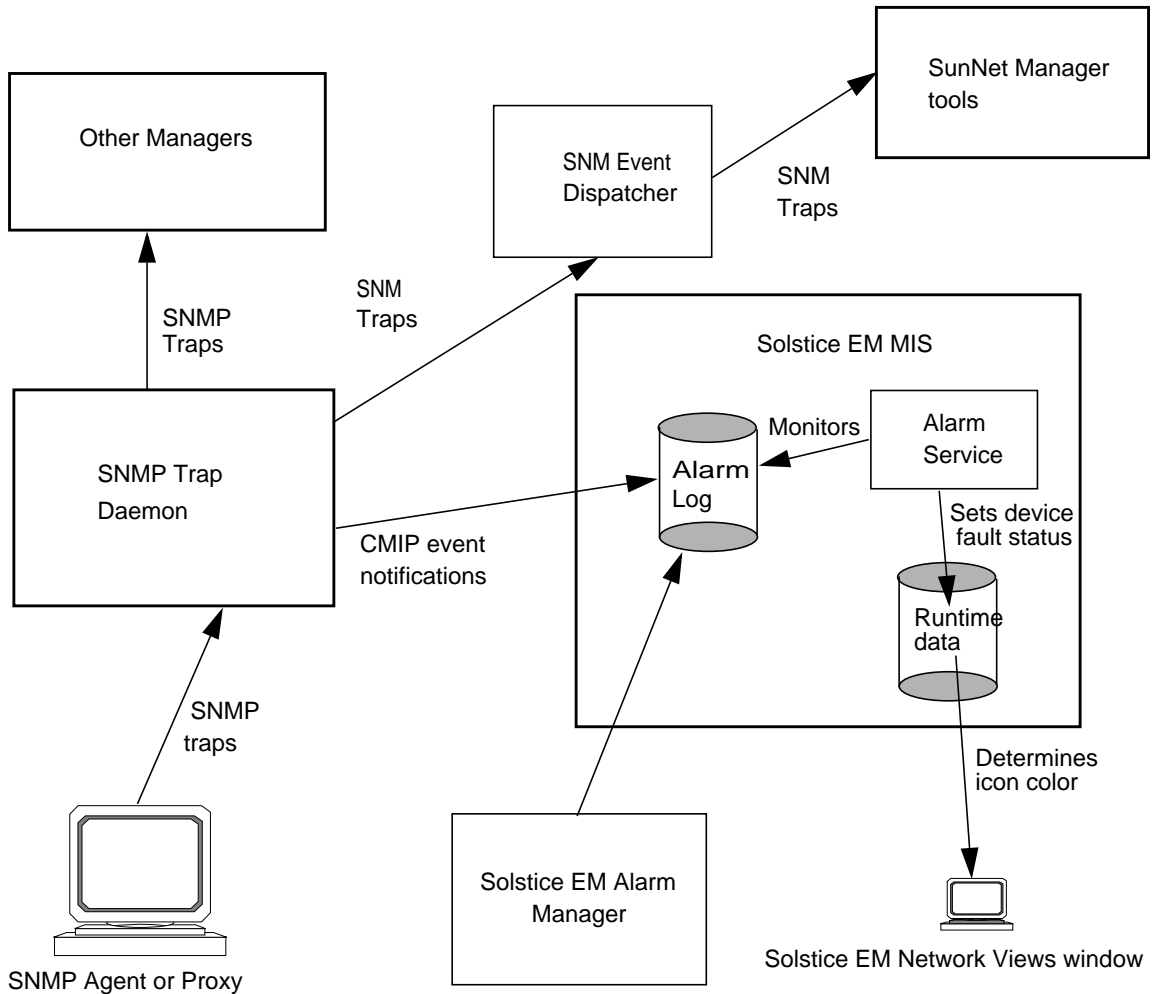


FIGURE 2-11 SNMP Trap Daemon Operation

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 11.

A default mapping is provided when you install the trap daemon. With this default mapping, a user who invokes the Alarms window 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 the following figure.






Solstice EM – Alarms [saturn]					
File Edit View Actions Tools					Help
	Severity	Most Recent	Total	Open	Ack'd
	critical	04/24/2001 12:23:37	5	5	0
	major	04/24/2001 12:23:38	4	4	0
	minor	04/24/2001 12:23:38	4	4	0
	warning	04/24/2001 12:23:38	4	4	0
	indeterminat.	04/24/2001 12:23:37	1	1	0
					5 Alarms, 0 Selected

FIGURE 2-12 Viewing Trap Notifications

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 Solstice EM Nerve Center's SNM event request capability. Using the RPC MPA, the MIS communicates with the SNMP proxy agent via RPC protocol (over UDP/IP), and the proxy agent talks to SNMP devices. The following figure illustrates the use of the SNMP proxy agent for offloaded polling of SNMP devices.

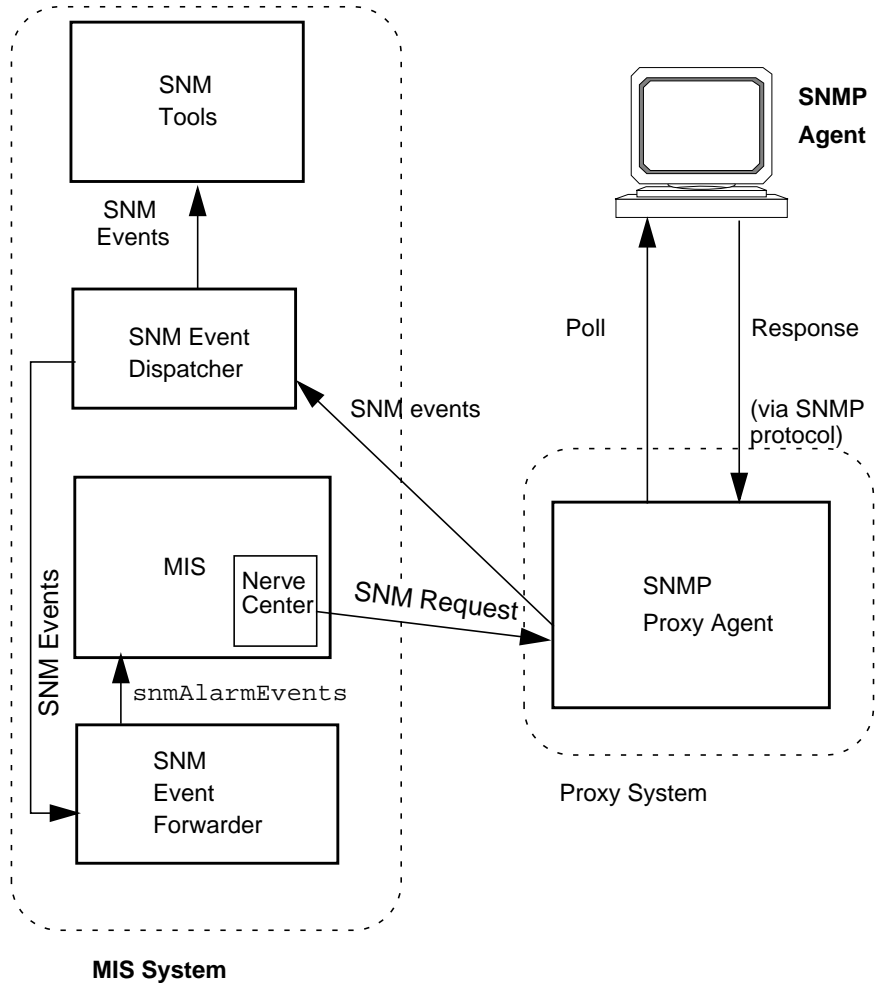


FIGURE 2-13 SNMP Proxy Agent Operation

2.5.2 CMIP Support

The Solstice EM CMIP Management Protocol Adapter (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 the following figure. 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 8.2.1 patch 6. The MPA can be used with SunLink CMIP 9.0 using RFC 1006 (over TCP/IP) or SunLink CMIP 9.0 over SunLink OSI 9.0. This enables communication with conformant CMIP management entities. SunLink CMIP and SunLink OSI are not shipped with Solstice EM.

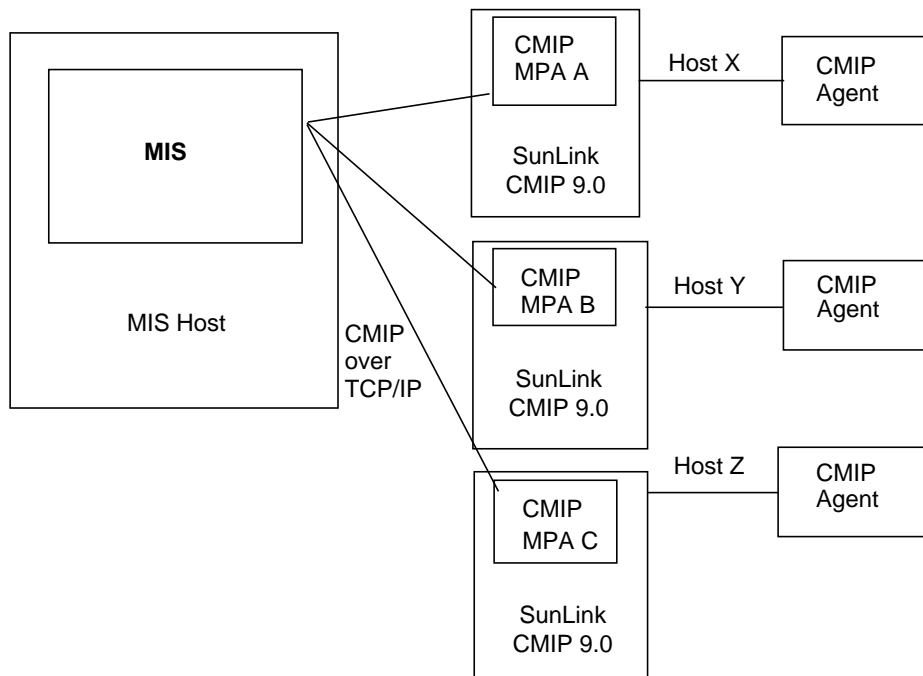


FIGURE 2-14 CMIP MPAs in Distributed Configuration

For information about configuring CMIP support, see Chapter 12.

2.5.2.1 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 the following figure, Solstice EM's CMIP Management Protocol Adapter (MPA), installed on the MIS machine, can support a TMN Q3 connection to a CMIP agent, which provides access to the managed resources.

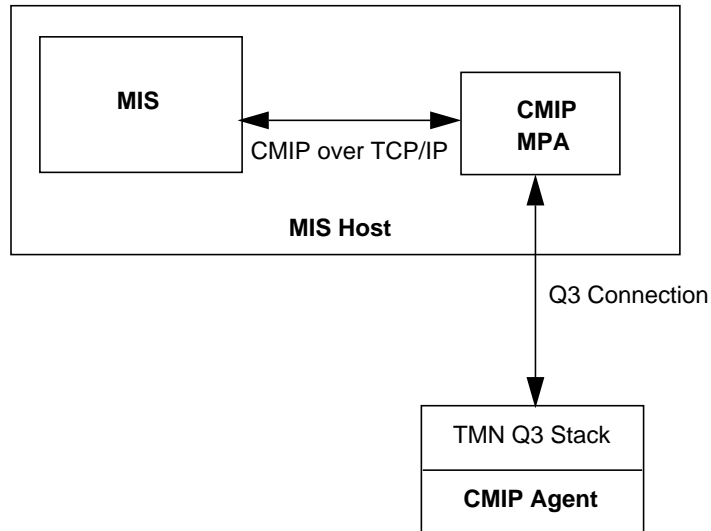


FIGURE 2-15 TMN Q3 Connection to Solstice EM

2.5.3 Other Network Management Protocols

Legacy or proprietary network management protocols can be supported by Solstice EM through the development of a custom Management Protocol Adapter (MPA). Third-party developers interested in creating such custom MPAs should refer to Chapter 11 in *Developing C++ Applications*.

2.5.4 Java Dynamic Management Kit Agents

Java Dynamic Management Kit (JDMK) is a set of Java classes, Java interfaces, and tools that simplifies the development of management services.

Writing object or agent behavior using JDMK allows you to add object behavior without being an expert in the fore mentioned areas. It will also allow dynamic updating of these behaviors, remote access, and a separation from the MIS process on either NT or Solaris boxes. The components necessary to allow this within Solstice EM are:

- **JavaBean to Guidelines for the Definition of Managed Objects tool**
A tool that takes as input a package or set of JavaBean classes and outputs a corresponding Guidelines for the Definition of Managed Objects (GDMO) document which is then loaded into Solstice EM.
- **JDMK MPA**
An MPA that maps GDMO CMIS requests into JavaBean JDMK calls. This will allow any PMI application to access JavaBeans through JDMK as GDMO objects. This will not allow Java objects to make CMIS requests back to the MIS. This may be achieved via the JMI or the PMI. A JDMK agent will be able to send M-Events to the MIS through the JDMK to CMIS Event listener.
- **JDMK to CMIS Event Listener**
A daemon process that converts JDMK JavaBean events to Solstice EM CMIS notifications. JDMK agents will be able to send events through the JDMK event mechanism which will be translated into Solstice EM CMIS notifications. The type of notification generated will be transparently handled as part of the JavaBean to GDMO tool. Any JavaBean which implements the `addXXXListener` and `removeXXXListener` will cause the JavaBean to GDMO tool to generate an `XXX` notification type.

There are three types of users for JDMK agent behavior in Solstice EM.

- End users managing Java agents developed using JDMK.
- Solstice EM application developers using JDMK JavaBean objects to implement agent behavior.
- Solstice EM application developers using remote JDMK JavaBean objects to implement remote object behavior within Solstice EM.

2.6 Object Classes and Event Notification Types

The definition language used to represent management information internally in the MIS is the GDMO, outlined in the ITU ISO/IEC 10165-4 standard. This provides the Solstice 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 JDMK 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 are 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, adding new GDMO object class definitions, converting an SNMP Concise MIB to a GDMO document, or converting an SNM schema to a GDMO document, refer to Chapter 8 in *Management Information Server (MIS) Guide*.

PART II Customizing Solstice EM Tools

Using Solstice EM for 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.

This chapter provides you with some ideas on how to use *Solstice Enterprise Manager* (Solstice EM) to meet your network management goals. These methods and scenarios are not the only ways to meet your goals. The approach best suited for a given situation will depend on the particular network configuration, available network management tools, and network management priorities.

This chapter describes the following topics:

- Section 3.1 “Fault Management Summary” on page 3-1
- Section 3.2 “Using Fault Management” on page 3-3
- Section 3.3 “Viewing Fault Status” on page 3-3
- Section 3.4 “Reporting Faults as Alarms” on page 3-5
- Section 3.5 “The Event Logs Tool and Alarm Logging” on page 3-6

3.1 Fault Management Summary

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

- 1. Decide on the information you need to manage your network.**

- 2. Create request templates, if needed. To create Basic request templates, use the Design Simple Request. To create Advanced request templates, use the Design Advanced Request.**

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. For information on using the Design Advanced Requests tool, see Chapter 18. For information on designing Nerve Center request templates, see Chapter 15 and Chapter 20. For information on using Simple Requests, see Chapter 9.

- 3. Use the Event Logs tool to create logs, to store those events for which you want to have a historical record and to define which events are logged to which logs.**

For information on creating and modifying alarm logs, see Chapter 5.

- 4. Choose 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 Network Views window. For information on configuring the Alarm Service, see Chapter 4.

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

For information on customizing the SNMP trap daemon mapping of SNMP traps to event notification, see Chapter 11.

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

For more information, see Chapter 7.

3.1.1 Before Starting Fault Management

Before customizing Solstice EM to perform fault management tasks, you should complete the following:

- Populate your MIS to add multiple managed objects.

Managed objects can be added automatically or one-by-one using Network Discovery. Refer to Chapter 4 in the *Managing Your Network* for how you can add managed objects to your MIS.

- Configure objects representing your network according to the network management protocol and agents they support (CMIP, SNMP, or SunNet Manager (SNM) RPC).

This is done either via the Network Discovery process, CMIP agent registration, or one-at-a-time using the Network Views-Object Properties.

3.2 Using Fault Management

Three key tools provided by Solstice EM for tracking fault status are as follows:

- Network Views
- Alarms
- Event Logs

Three important Solstice EM components that can provide you with information about critical network events are as follows:

- Nerve Center requests, which are launched from the Network Views window.
- SNMP trap daemon, which listens for traps generated by SNMP agents.
- Simple Requests, which are launched from the Viewer.

The various options available to you to set up Solstice EM for tracking fault status of devices on your network are described in this section.

The steps involved in monitoring the fault status of devices are described in *Managing Your Network*.

3.3 Viewing Fault Status

The Network Views window 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 memory usage of a router network being rated as less severe than an event indicating that the router not working at all.

Solstice EM provides six severities; by default, these are color-coded as indicated in the following table.

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 Alarms window—a Solstice EM tool that enables you to selectively view, acknowledge, and clear alarms.

3.3.1 Changing the Color Associated with a Severity

▼ To Change the Color Associated with a Severity

1. In the Network Tools window click Network Views.
2. Click File → Customize → Display Settings → Colors to open the Severities window.
3. Select Alarm Severity.
4. Select the color you want to use for that severity in the field.
5. Click Modify.

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.

3.3.2 Alarm Severity Propagation

For container objects the propagated severity is the most severe outstanding alarm posted against the container and all its children—for example, if a container contains devices such as a host with a major alarm and also a router with critical alarm, the severity of the container will be critical, as critical is more severe than a major alarm. This severity is evaluated recursively for all containers within containers.

3.3.3 Access to Tools, Features, and Database Objects

Network Views window displays the Solstice EM tools to which you have access. Solstice EM has four levels of security:

- Tool Access
- Tool Feature Access
- Object Access
- Database Access

Tool access enables administrators to set up the environment so that certain users or groups can use or run a Solstice EM tool that has been registered with the Management Information Server (MIS).

Tool feature access enables administrators to provide or restrict access to certain features within a tool to specific users or groups. Tool and feature level access is enforced by the tool; the MIS is used only to store the list of features for each tool.

Object class or instance access enables administrators to permit or restrict access to specific managed object classes or managed object instances.

Database access enables administrators to provide or restrict access to information found in the MIS. Prompting database access allows the respective users to view summaries of log information, while users or groups with complete access can view detailed log information.

For more information about security and access levels, refer to Chapter 6 in *Managing Your Network*.

3.4 Reporting Faults as Alarms

The fault status of objects displayed in the Network Views window and Alarms window is controlled by the Alarm Service, which is in the topology server. The Alarm Service monitors incoming alarms posted to the `AlarmLog` and updates the fault status of objects to match the highest severity amongst the outstanding

(uncleared) alarms posted against that object. If the Solstice EM 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 reflecting the uncleared minor alarms. If all the alarms are cleared or purged, the icon has no status coloring—indicating that the state of the device is “normal.”

The Alarm Services monitors a log called `AlarmLog`. When alarms are logged to this log, they automatically affect the icon color in the Network Views window.

For more information about using the Alarms window, refer to Chapter 5 in *Managing Your Network*. Configuration of the Alarm Service is described in Chapter 4.

3.5 The Event Logs Tool and Alarm Logging

The Event Logs tool is the tool used 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` and to all logs is determined by a Common Management Information Service (CMIS) filter, called a *discriminator construct*. You use the Event Logs tool 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 “Creating a Separate Log for Enterprise-Specific Trap Notifications” on page 15.)

Actions of the Alarms window also affect fault indication in the Network Views window. If a network administrator uses the Alarms window to clear all the outstanding alarms against router sledge, the Alarm Service changes sledge's fault status to `cleared`, and the Network Views window icon changes color accordingly. Thus, the Alarm Service ensures that the Network Views window and Alarms window 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 Network Views window icons automatically) depends upon the types of network events you want to track and the management protocols you are using.

For example, SunNet Manager RPC agents (shipped with Solstice 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 17.) 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-generating functions that can be inserted in request templates. The SNM event request templates shipped with Solstice EM, such as `AdminOperStatusUp`, `CheckCPU`, and `DeviceReachablePing`, use this method of handling `snmAlarmEvents`.
- Alternatively, the `AlarmLog` could be configured to automatically log incoming `snmAlarmEvents`. If you want to implement this, you can use the Event Logs tool 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 Network Views window to examine the contents of event logs.

For more information ...

- *Managing Your Network* describes the use of the Network Views window, Alarms window, and Log Network Views window in accomplishing network management tasks.
- The Alarm Service is described in this guide in Chapter 4.
- The Event Logs tool is described in Chapter 5.

3.5.1 Receiving Network 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*.

3.5.1.1 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. For managing large numbers of devices, fault management strategies that rely on event notifications and indirect polling by proxy agents are more efficient than direct polling because such strategies minimize network traffic and MIS processing load. (Offloading of polling to RPC agents is described in Chapter 17.)

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 in the following sections.

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 Chapter 7 in *Managing Your Network*.

3.5.1.2 Monitoring Device Availability

Solstice EM's Network Discovery tool provides a form of polling for device status that does not require the use of Nerve Center requests.

The main purpose of Network Discovery's Monitor function is to update the representation of your network view 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 Network Discovery was last run. Monitor compares the existing network view 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.

If the 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 Network Views window changes accordingly.

By default, Monitor's "No Response" event generation capability is turned off.

▼ To Activate the Event Generation Capability

1. **Invoke Network Discovery from the Network Tools window, if it is not currently running.**

Select the Actions menu → Monitor Network option to invoke the Monitor Network window.

2. **Select On for the Generate Event if Object is Down option and select a severity from the pulldown menu (shown in FIGURE 3-1 on page 3-10).**

Monitor Network

Monitor Ping/RPC/SNMP Schedule Logging

Objects to Monitor

Monitor These Containers Only:

Ignore These Objects:

Object Down Timeout: 30 minutes

Time Between Cycles: 15 minutes

Holding Container: Root

Generate Event if Object is Down: ☒ On ☐ Off

Type of Event: Critical ☐

Save... Load... Start Cancel Help

FIGURE 3-1 Selecting a Severity for `communicationsAlarm` Generated by Monitor

3. Click **Schedule** tab, select the time of day and days of the week when you want the Monitor to be active. Click **Start** for your choices to be reflected.

Using Network Discovery is described in more detail in Chapter 3 of *Managing Your Network*.

3.5.2 Event Notifications

There are two ways 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, Alarm Service monitoring of alarm logs ensure that icon color in the Network Views window is dynamically changed to reflect the severity of the alarms.

3.5.2.1 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 the following figure.

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

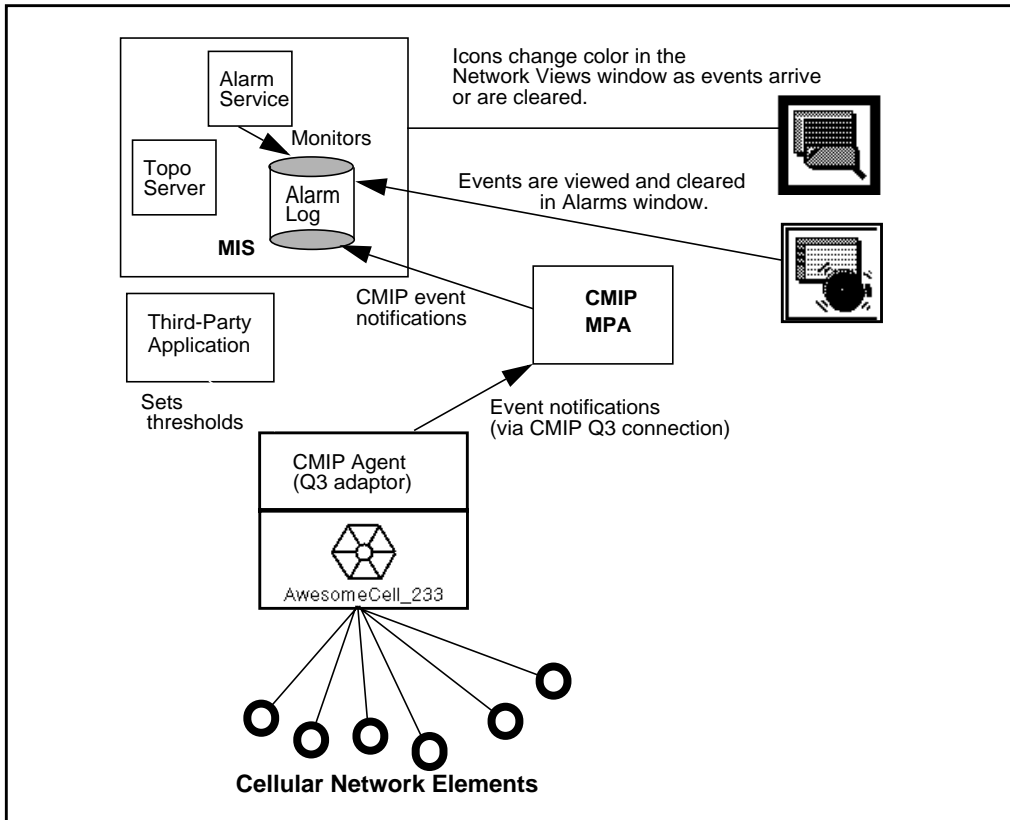


FIGURE 3-2 CMIP Management of a Cellular Network

3.5.3 Using SNMP Traps

Simple Network Management Protocol (SNMP) agents also have the ability to initiate the generation of event notifications; these messages are called *traps*. The 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 SNMPv1 and SNMPv2c 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 the following table.

TABLE 3-2 Default SNMP Trap Notifications and Severities

SNMP Trap	Notification Name	Default Severity
coldStart	coldStartTrap	warning
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 Alarms window, you can tell at a glance the types of traps that have been logged against devices in your network, as shown in the following figure.

The screenshot shows a window titled "Solstice EM – Alarms [saturn]". Inside, there is a menu bar with "File", "Edit", "View", "Actions", "Tools", and "Help". Below the menu bar is a table with the following data:

Severity	Most Recent	Total	Open	Ack'd
critical	04/24/2001 12:23:37	5	5	0
major	04/24/2001 12:23:38	4	4	0
minor	04/24/2001 12:23:38	4	4	0
warning	04/24/2001 12:23:38	4	4	0
indeterminat	04/24/2001 12:23:37	1	1	0

At the bottom right of the window, it says "5 Alarms, 0 Selected".

FIGURE 3-3 Viewing Trap Notifications in the Alarms Window

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 11.)

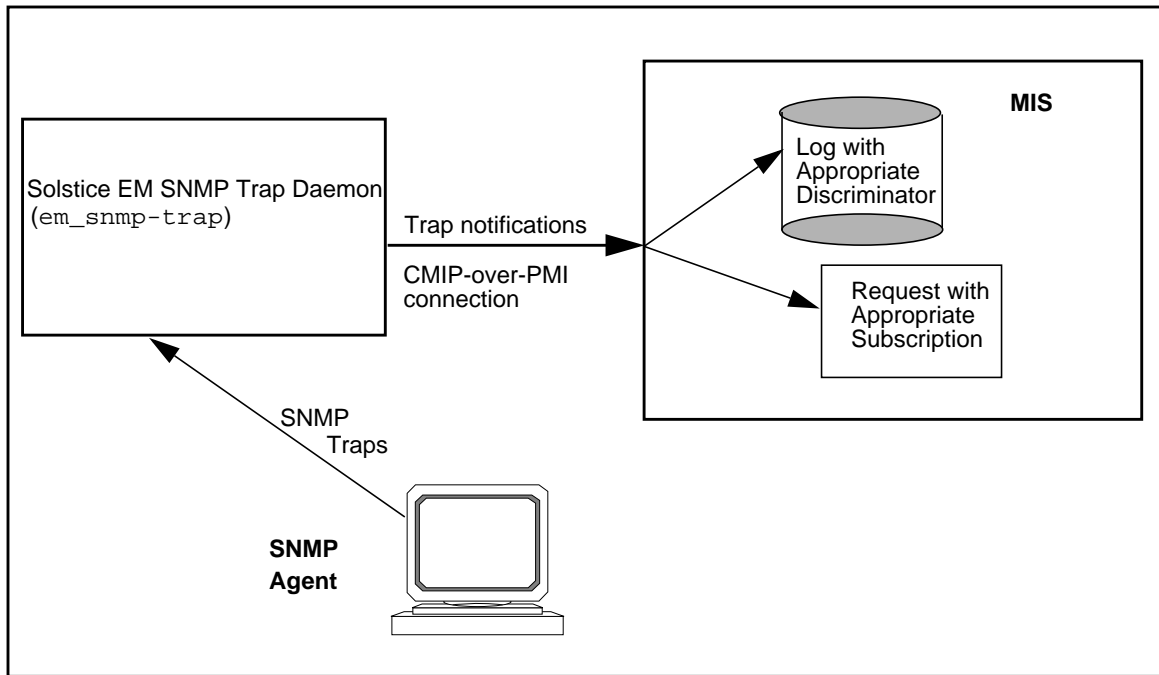


FIGURE 3-4 Solstice EM Processing of SNMP Traps

3.5.3.1 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 Network Views window, 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. Since the critical alarm is of 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.

The following figure shows the flow of information from traps to logs using the `SnmpLinkUpDownTrap` request.

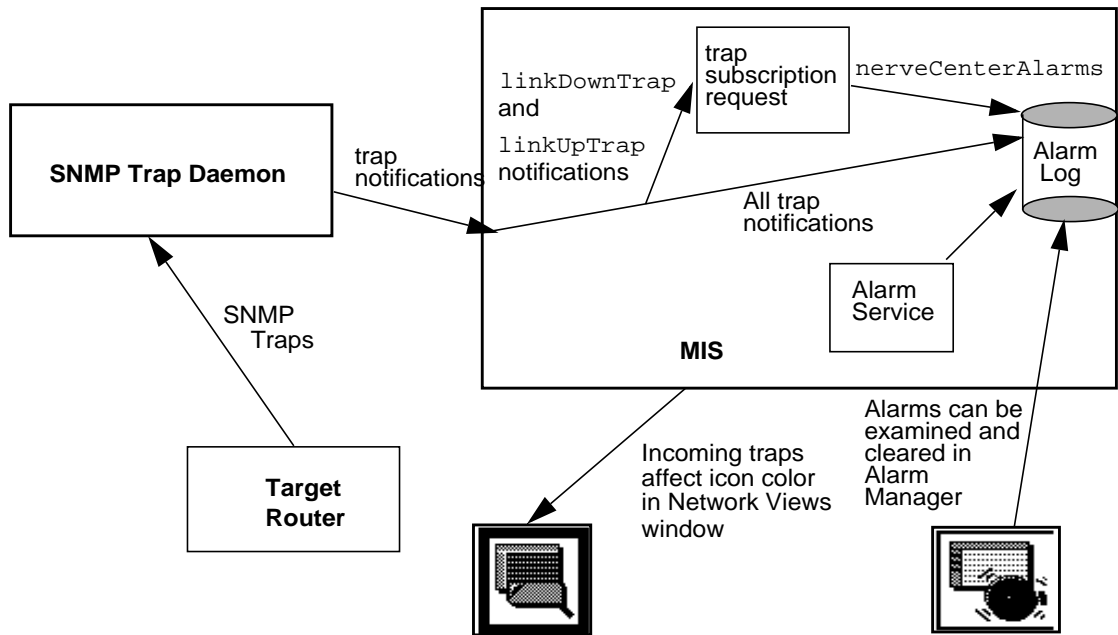


FIGURE 3-5 Example of SNMP Trap Handling Using `SnmpLinkUp/DownTrap` Request

3.5.3.2 Creating a Separate Log for Enterprise-Specific Trap Notifications

As 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 15.)

If you do not want `enterpriseSpecificTrap` notifications to automatically affect the icon color in the Network Views window, 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 historical record.

Creating a separate Enterprise-specific trap event log includes two separate tasks:

1. **Modifying the Alarm Log to exclude Enterprise-specific traps.**
2. **Creating a separate log and setting the discriminator to log Enterprise-specific traps.**

▼ To Modify the AlarmLog

1. **Invoke the Event Logs tool from Network Tools.**
2. **Select the `AlarmLog`.**
3. **Select Actions → Properties.**

This invokes the Event Logs properties dialog box, as shown in the following figure.

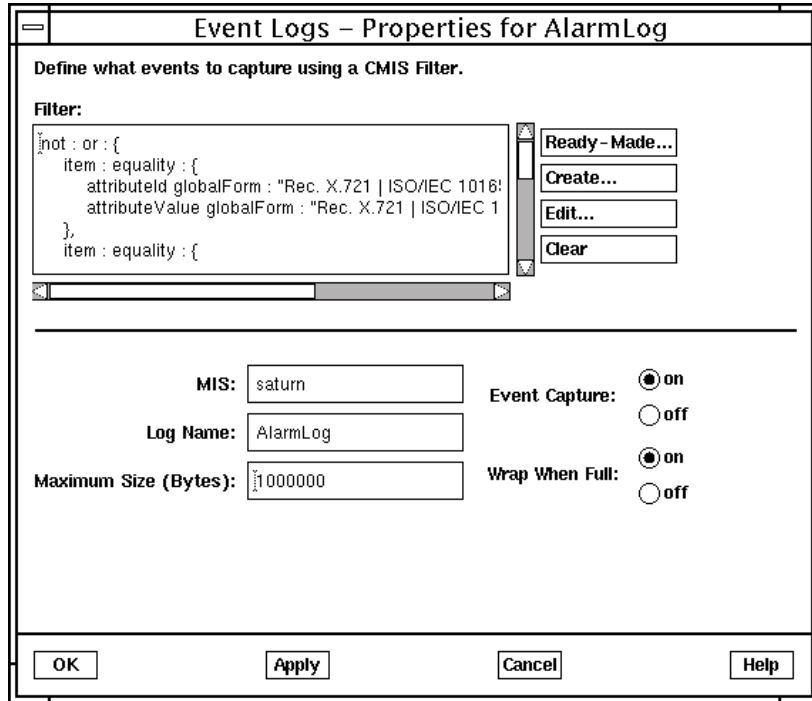


FIGURE 3-6 Viewing AlarmLog Properties in the Event Logs Properties Dialog

4. To add a new CMIS filter entry for `enterpriseSpecificTraps`:
 - a. Click **Edit** to add a new item entry for `enterpriseSpecificTraps`.
Solstice EM displays the CMIS Filter dialog box as shown in the following figure.

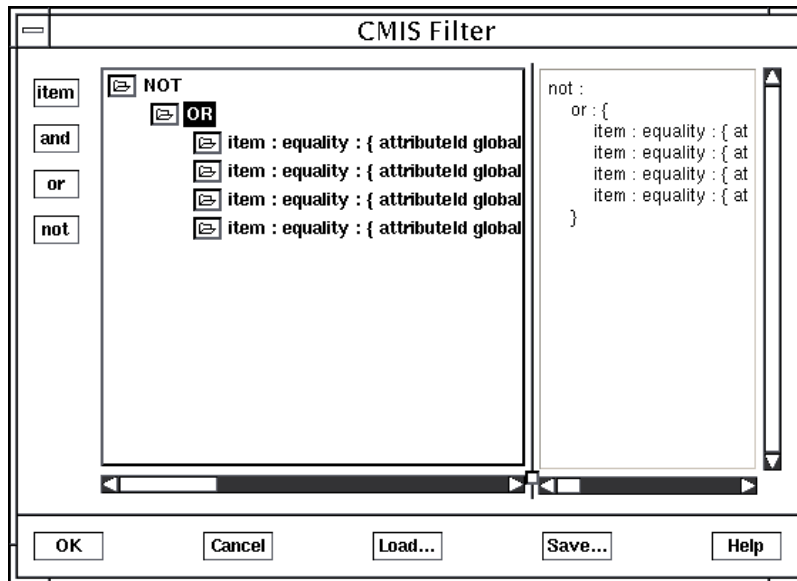


FIGURE 3-7 CMIS Filter Window

- b. Select OR to highlight the editing buttons on the left (item, and, or, and not).
- c. Click the item button to display a new item window for the CMIS Filter, as shown in the following figure.

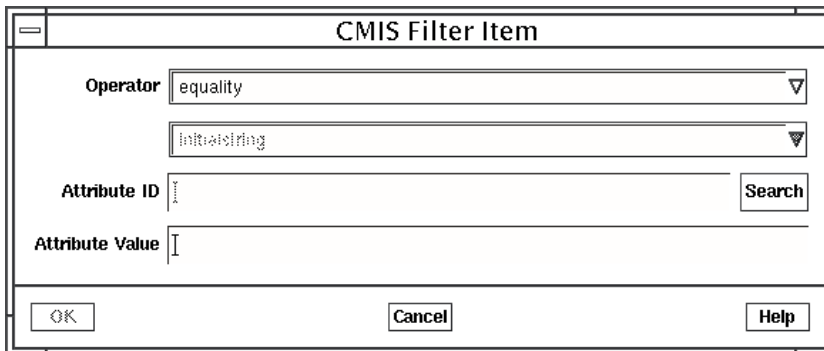
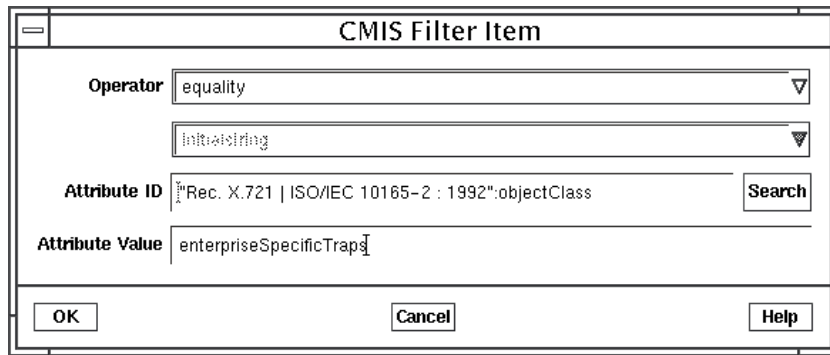


FIGURE 3-8 CMIS Filter Item Dialog Box

- d. Enter the Attribute ID and Attribute Value.
- e. Click OK in the CMIS Filter Item box to add the new entry to the CMIS filter.

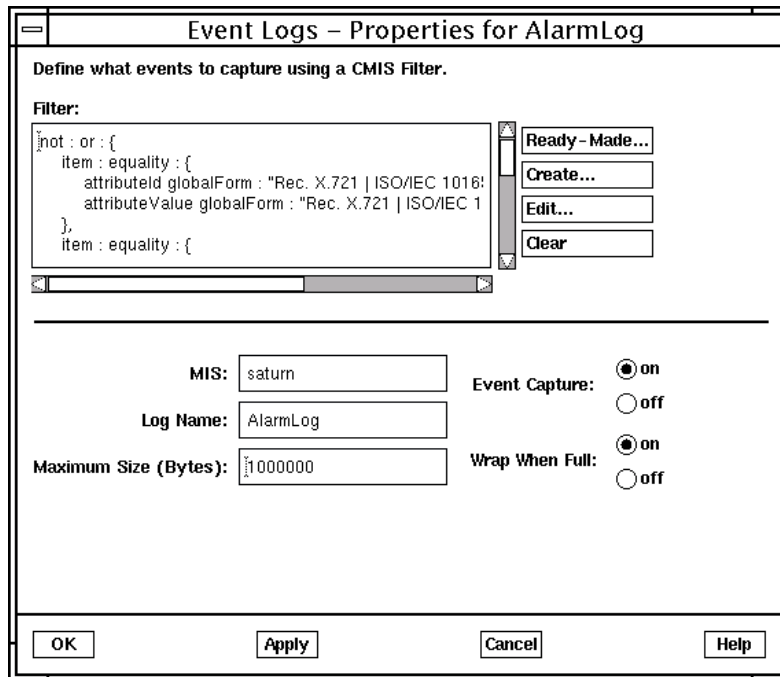


The CMIS Filter Item dialog box contains the following fields and controls:

- Operator:** A dropdown menu with "equality" selected.
- Initial string:** A text input field with "initialstring" entered.
- Attribute ID:** A text input field with "Rec. X.721 | ISO/IEC 10165-2 : 1992".objectClass. To its right is a "Search" button.
- Attribute Value:** A text input field with "enterpriseSpecificTraps" entered.
- Buttons:** "OK", "Cancel", and "Help" buttons are located at the bottom of the dialog.

FIGURE 3-9 Adding an Item to the Default AlarmLog Discriminator

- f. Click OK in the CMIS Filter window to modify the log discriminator.
5. Click OK in the Event Logs properties dialog box to have the changes you made to the AlarmLog to be reflected.



The Event Logs – Properties for AlarmLog dialog box contains the following sections and controls:

- Define what events to capture using a CMIS Filter.**
 - Filter:** A text area containing a CMIS filter expression:

```

not : or : {
  item : equality : {
    attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992".objectClass
    attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992".enterpriseSpecificTraps
  }
}

```
 - Buttons:** "Ready - Made...", "Create...", "Edit...", and "Clear" buttons are located to the right of the filter text area.
- MIS:** A text input field with "saturn" entered.
- Log Name:** A text input field with "AlarmLog" entered.
- Maximum Size (Bytes):** A text input field with "1000000" entered.
- Event Capture:** Radio buttons for "on" (selected) and "off".
- Wrap When Full:** Radio buttons for "on" (selected) and "off".
- Buttons:** "OK", "Apply", "Cancel", and "Help" buttons are located at the bottom of the dialog.

FIGURE 3-10 AlarmLog Discriminator Construct With enterpriseSpecificTraps Excluded

▼ To Create a Separate Log

1. **To create a new log for enterpriseSpecificTraps, select Action → Create Log.**

This invokes the Create Log window, shown in the following figure.

Event Logs – Properties for AlarmLog

Define what events to capture using a CMIS Filter.

Filter:

C 10165-2 : 1992".objectClass, enterpriseSpecificTraps }

Ready-Made...
Create...
Edit...
Clear

MIS: saturn

Log Name: AlarmLog

Maximum Size (Bytes): 1000000

Event Capture: ☒ on ☐ off

Wrap When Full: ☒ on ☐ off

OK Apply Cancel Help

FIGURE 3-11 Creating a New Log for enterpriseSpecificTraps (will be displayed Filer pane)

2. **Enter the name of the new log in the Log Name field.**

If you leave the Maximum Size field as 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.

3. **Select Create to build the discriminator construct for the new log.**

This invokes the CMIS Filter window.

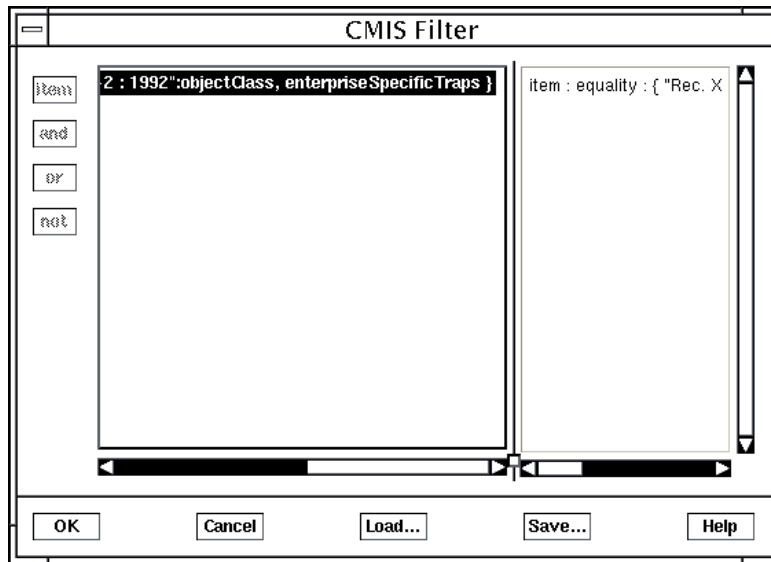


FIGURE 3-12 Specifying a CMIS Filter for enterpriseSpecificTraps

4. **Select Item to create a discriminator that selects enterpriseSpecificTraps (as shown in FIGURE 3-9).**
5. **Enter the Attribute ID and Attribute Value.**
6. **Click OK in the CMIS Filter Item box to add the new entry to the CMIS filter.**
7. **Click OK in the CMIS Filter Item window to add the item to the CMIS filter.**
8. **Click OK.**

The new discriminator construct appears in the Create Log window (as shown in FIGURE 3-11 on page 3-20).

9. **Click OK in the Create Log window to create the new log.**

3.5.3.3 Forwarding 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 Solstice 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 Solstice EM MIS. When SNM event notifications are forwarded to Solstice 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 Solstice EM alarm severities in the manner indicated in the following table. 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 Solstice 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 Network Views window, 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. (For information on changing the icon colors for the perceivedSeverity of alarms, see Section 3.3.1, “Changing the Color Associated with a Severity .) Incoming snmAlarmTraps will thus affect fault status color of icons in the Network Views window. (For more information on forwarding of information from SNM Consoles to Solstice EM, see Chapter 7.)

Using the Alarm Service

The Alarm Service is the module in the Log Server responsible for updating and storing the state of managed object instances (MOI) in the MIS. The state of an MOI is reflected in the color of its corresponding Network View node in the Network Views.

This chapter describes the following topics:

- Section 4.1 “Network View Nodes” on page 4-1
- Section 4.2 “Alarm Management” on page 4-3
- Section 4.3 “The Alarm Service” on page 4-4
- Section 4.4 “Configuring the Alarm Service” on page 4-6
- Section 4.5 “Alarm Information Display in Solstice EM Tools” on page 4-8
- Section 4.6 “User-configurable Alarm Log Record Filter for Alarm Service” on page 4-10

4.1 Network View Nodes

Network View nodes are created by users to logically model managed objects in their network environment. This is the object seen by the Network Views. Each Network View node has an attribute `topoNodeMOSet` which points to the managed object instances (MOI) the Network View node represents. Thus, the Network View node represents the state of the managed objects or MOI. When an alarm comes into the platform, it is against the MOI, not the Network View node.

When an alarm is received by MIS, the log server logs the alarm and the `discriminatorConstruct` determines whether to log the alarm or not.

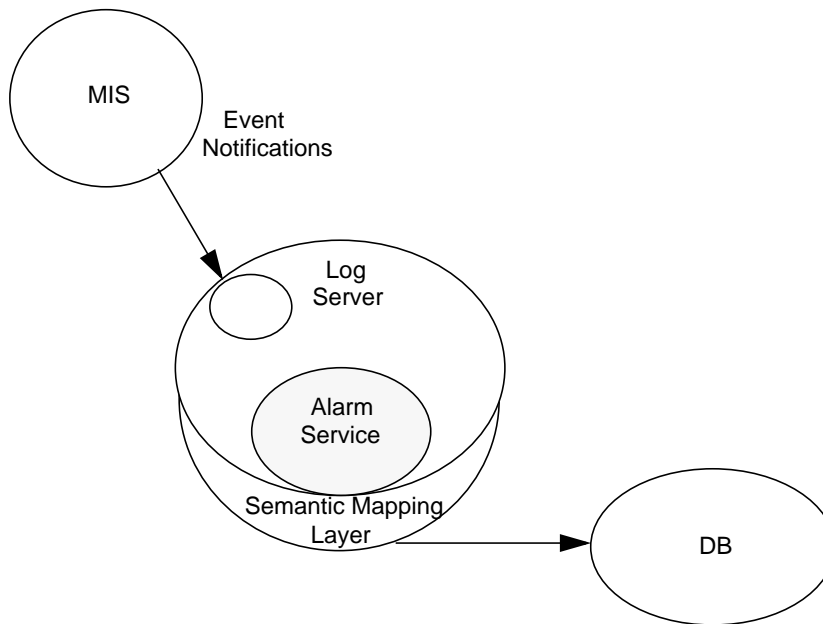


FIGURE 4-1 Network View Nodes

The Alarm Service is notified with the creation of each alarm log record. It maps the MOI value to the corresponding Network View node and updates the alarm counters based on severity. This triggers the severity propagation of `topoNode` in two ways:

- Propagates to parents, according to the attribute `topoNodeParents` (you can have multiple parents). This applies if `topoNodePropagateUp` is set to “true.”
- Propagates to peers, according to the attribute `topoNodePropagatePeers`. This applies if `topoNodePropagateUp` is set to “true.”

In addition, the Alarm Service keeps the `topoNodeSeverity` synchronized so that it represents the highest (most critical) uncleared alarm log record that is posted against the Network View node. When there are no open alarms posted against a Network View node, the `topoNodeSeverity` returns to its “normal” value of cleared.

4.2 Alarm Management

Alarms arrive at the MIS as event notifications. A log is a software entity that collects records (called log records) of event notifications. As many logs as you want can be created using the Event Logs. (For more information about the Event Logs, see Chapter 3. For more information on logs, refer to Chapter 6 in the *Management Information Server (MIS) Guide*.)

Alarms have a default log object called `AlarmLog`, to subscribe alarm notifications. This is the default log shipped with Solstice EM. By default, the Alarm Service monitors the `AlarmLog`, which includes all alarm types *except* for the following:

- `attributeValueChange`
- `objectCreation`
- `objectDeletion`
- `stateChange`
- Any alarm type without a `perceivedSeverity`

It is possible to modify the `AlarmLog` discriminator construct to:

- Log various other types of alarms to the `AlarmLog`
- Point the Alarm Service to another log or multiple logs.

For information on how to do this, see the Section 5.4 “Defining the CMIS Filter .”

The following figure illustrates how `internetAlarms` and other trap notifications, CMIP notifications, and `nerveCenterAlarms` get logged to the `AlarmLog`.

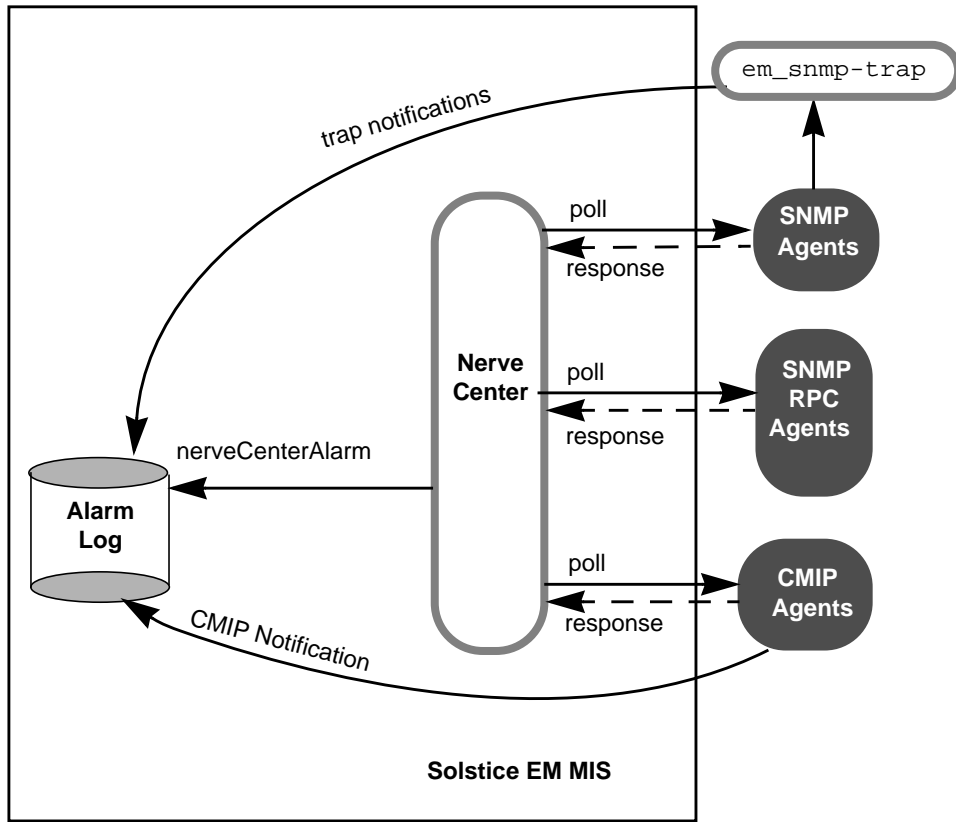


FIGURE 4-2 Logging of Alarms to the AlarmLog

4.3 The Alarm Service

The scenario in this section is designed to show you how the Alarm Service works. You must perform this scenario on a machine where the MIS is running locally, and the machine must be SNMP-manageable. If it is not, you must create an SNMP agent object by using the Network Views-Object Properties window. For information on the tools used in this scenario, refer to Chapter 4 in *Managing Your Network*.

▼ To Populate the MIS

1. **Populate the MIS by running Network Discovery, if a runtime database does not yet exist:**

```
hostname% em_discover -device <hostname>
```

Note – When using command line options, you must first source `emenv.csh`. At the command line, type `source /opt/SUNWconn/em/bin/emenv.csh`

2. **Start the Alarms Window:**

```
hostname% em_alarmmgr &
```

3. **Start the Network Views:**

```
hostname% em_viewer &
```

4. **Find your device in the viewer canvas.**

In the viewer canvas, you should see a Network View node that has the name of your machine. The `topoNodeMOSet` attribute of this Network View node might contain:

```
/systemId=name:"<host>"/internetClassId={1 3 6 1 4 1 42 2 2 2 9 2 4  
1 0}  
/cmipsnmpProxyAgentId="<host>"
```

where `<host>` is the name of your machine. This MOI represents the SNMP agent running on your machine. The Network View node in the Network Views, then, represents the state of this agent running on your machine.

Alarms in the system get posted against MOI. The Alarm Service maps the MOI to the Network View node representing it.

5. **To see how this works, login as root, then kill the SNMP daemon on your machine with the following command:**

```
# /etc/init.d/init.snmpd stop
```

6. Restart the SNMP daemon:

```
# /etc/init.d/init.snmpd start
```

When the SNMP daemon is restarted, a Cold Start trap is sent by default to the local host. This causes a critical alarm to be posted against the MOI representing the SNMP agent. The Alarm Service maps this to zero or more Network View nodes (zero or more because there may be no Network View nodes with this MOI as a member of their `topoNodeMOSet`, or there may be many that contain this MOI in their `topoNodeMOSet`).

When the alarm is posted, the icon color in the Network Views changes according to the severity of the alarm. The color of the alarm in the Alarms Window matches the color of the Network View node in the Network Views.

7. Delete the Alarm using the Alarms Window.

Click once on the alarm in the table, then select Actions ➔ Delete From Log.

If there are several alarms of varying degrees against one node, the icon will appear in the color of the highest severity registered against that node.

The key to understanding the Alarm Service is to understand that a managed object has a state. The Network Views uses icons to represent the state of the managed objects, while the Alarms Window shows the actual managed object. For example, suppose your machine has only one managed object: an SNMP agent, as in this scenario, and that there are already five critical alarms posted against this agent. If a sixth alarm of a severity lower than critical (minor, for example) were posted against this managed object, the Network Views would not reflect this alarm, because of its lower severity. The Alarms Window, however, would show all six alarms.

Both the Network Views and the Alarms Window use the same severity mapping so that the colors are consistent.

4.4 Configuring the Alarm Service

By default, the Alarm Service monitors the `AlarmLog`. However, suppose you want to log some non-default alarms to the `AlarmLog`, or want to monitor only a specific set of alarms, and have created some additional logs in addition to `AlarmLog`. (You can create additional logs by using the Event Logs.) You can use the MIS Objects Object Editor (OBED) to tell the Alarm Service which log(s) to monitor.

The `emAlarmServiceList` object has the attribute `emAlarmLogList`. This attribute, by default, contains the value `AlarmLog`. The Alarm Service automatically monitors any log that is added to `emAlarmLogList`.

4.4.1 Adding Logs to emAlarmLogList

To add a logs to emAlarmLogList, you must use OBED to modify the emAlarmServiceList object.

▼ To Add a Log using OBED

1. Click the folder next to subsystemId="EM_MIS" the selected object's subordinates are displayed.
2. Double-click the listname="emAlarmServiceList" subordinate object.
The Network Views-Object Properties window is displayed.
3. Add the logs you want the Alarm Service to monitor in the emAlarmLogList field.

By default, the emAlarmLogList contains the following:

```
{ string : "AlarmLog" }
```

Suppose you have created a log called WarningLog. To add this log to emAlarmLogList, change the emAlarmLogList field as follows:

```
{ string : "AlarmLog", string : "WarningLog" }
```

As many logs as you want may be added in this fashion. Just be sure to separate each log with a comma.

4. Click Set when you are finished.

4.4.2 Deleting Logs from the Event Logs Window

If you do not want the Alarm Service to monitor a specific log, the log must be removed from emAlarmLogList. You can remove a log by using the Event Logs or OBED.

▼ To Remove a Log Using Event Logs

1. Select the log(s) you want to delete in the Event Logs main window.

2. **Select Actions → Delete from the menu bar to delete the selected log(s).**

▼ To Remove a Log Using OBED

1. **Select the log(s) you want to delete in the OBED main window.**

Logs appear as a subordinate objects beneath the `systemId` object.

2. **Select Object → Delete from the menu bar to delete the selected log(s).**

When you remove a log from the MIS, it is automatically removed from `emAlarmLogList`, thus eliminating the necessity to manually edit this field in the Network Views-Object Properties window.

4.4.3 Turning Off the Alarm Service

Removing all logs from `emAlarmLogList` in effect turns off the Alarm Service because it will not change the `topoNodeSeverity` for a Network View node. However, the Alarm Service still keeps track of the MOI-to-`topoNode` ID mapping, in the hopes that when a log is re-added to `emAlarmLogList`, there is no catching-up penalty.

4.5 Alarm Information Display in Solstice EM Tools

The following subsections explain how the Alarm Service controls the color mapping for alarms displayed in Solstice EM tools.

4.5.1 Alarm Information Display in Alarms Window

The Alarms Window does not display the alarm notifications directly. Instead, it displays the alarm log records contained in the alarm logs (for example, “AlarmLog”) in a tabular format. You can specify filters to ignore unwanted alarms, association groups to summarize alarms, and sorting to prioritize alarms. The Alarms Window also monitors the alarm log for `objectCreation`, `objectDeletion`, `attributeValueChange`, and other events for alarm log records, updating the display as necessary.

The Alarms Window displays each row in the table in colors, based on the severity of the alarm log record. The standard attribute used to denote severity is the `perceivedSeverity` attribute, which is defined in ASN.1 as one of the following enumerated types: `indeterminate`, `critical`, `major`, `minor`, `warning`, and `cleared`.

The alarm record managed object classes with `perceivedSeverity` attributes are as follows:

- `emAlarmRecord`
- `emInternetAlarmRecord`
- `nerveCenterAlarmRecord`

Severity of the alarm log record is translated to a color by the user-configurable color mapping defined by the Nerve Center. The default color mapping is shown in the following table.

Normal	no color
Warning	yellow
Minor	cyan
Major	orange
Critical	red
Indeterminate	blue
Cleared	grey

If you want to change the default color mapping, you must use the Severities window in the Design Advanced Requests by selecting **Edit** ➔ **Severities** from the menu bar. This change, however, is not dynamic, meaning that you need to restart the Network Views and Alarms windows to see this change in color. For more information about changing the color mapping, see Chapter 18.

4.5.2 Alarm Information Display in Network Views

The Network Views also displays alarm information, but in a more indirect and limited way. The Network Views does not read the alarm log or wait for events for alarm log records, such as `objectCreation` or `objectDeletion`. The alarm information displayed by the Network Views are updates by the Alarm Service.

4.6 User-configurable Alarm Log Record Filter for Alarm Service

The following table compares the user options available in the Alarms Window and Network Views for processing alarm log record information.

TABLE 4-1 Alarm Log Record Processing Options

Alarm Log Processing Option	Alarms Window	Network Views
Sorting	By any attribute.	By severity.
Association	By object instance, event type, probable cause, specific problem, additional text, additional info identifier, and additional information.	By object instance (Network View node).
Filtering	All open, cleared, or acknowledged alarms; all within a certain time period, only certain alarms, only alarms on certain objects, only certain severities, only certain event types, and more.	Only uncleared alarms.

Using the Event Logs Tool

The Event Logs tool enables you to create, modify, and delete log objects.

This chapter describes the following topics:

- Section 5.1 “Log Process Overview” on page 5-1
- Section 5.2 “Starting the Event Logs Tool” on page 5-4
- Section 5.3 “Using the Event Logs Tool” on page 5-5
- Section 5.4 “Defining the CMIS Filter” on page 5-13
- Section 5.5 “Sample CMIS Filters” on page 5-18
- Section 5.6 “Event Logs Tool Configuration File” on page 5-21

5.1 Log Process Overview

A log object is a software entity that collects records of event notifications. Agents create event notifications when they detect a change in the state of a managed resource. When a notification arrives at the *Solstice Enterprise Manager* (Solstice EM) Management Information Server (MIS), a record of the notification is appended to a log object if the notification passes a log filter (called a *discriminator construct*), or is discarded.

Note – Log objects are also called “logs” and the terms “log” and “log object” are used interchangeably in this chapter.

You can browse log records at any time using the Log Entries tool. (The Log Entries can be invoked by selecting Actions ➔ Log Entries, or by double-clicking on Event Logs.)

An MIS can have any number of log objects. As shipped with Solstice EM, the MIS has a single log object, `AlarmLog`, which is visible when you invoke the Event Logs tool. The default discriminator for `AlarmLog` ensures that records are logged for

SNMP trap notifications (alarms forwarded from remote SNM managers by Cooperative Consoles), Nerve Center alarms, and OSI standard alarms. By default all event notifications are logged to `AlarmLog` other than the following notifications, which are explicitly excluded:

- `attributeValueChange`
- `objectCreation`
- `objectDeletion`
- `stateChange`

This default log object may be sufficient for your needs. However, Event Logs tool makes it easy to create new log objects as and when required.

A *discriminator construct* is a CMIS filter that determines whether a log record is created under a log object. A log record is defined for each event notification. This mapping from event notification to log record is stored in the MIS at `subsystemID="EM-MIS"/listname="event2ObjectClass"` in attribute `evr2oclist`.

Each notification has an attribute describing its notification type. Common notification types are shown in TABLE 5-5, but the set is open-ended, requiring only that each new type have a registered OID. For information on adding new, user-defined event types to the MIS is discussed in the Chapter 6 of *Management Information Server (MIS) Guide*.

See Section 5.4, "Defining the CMIS Filter," for detailed information about log filters.

Every notification the MIS receives is passed to each log object that has been created. Each log object then applies its own log filter to decide whether to keep a record of the notification or ignore it.

5.1.1 Attributes of a Log

Each log object has the attributes shown in the following table. Some attributes can be modified. All of these attributes are accessible through the Event Logs tool.

TABLE 5-1 Log Object Attributes

Fields	Modifiable?	Description
Log Name	No	Each log object is identified by its FDN (Fully Distinguished Name) in the MIS' Management Information Tree (MIT).
Entries	No	The number of log records under the log object.
Admin State	Yes	When set to locked, the log object cannot be written to. When set to unlocked, allows the log object to be written to.

TABLE 5-1 Log Object Attributes (*Continued*)

Fields	Modifiable?	Description
Size (Bytes)	No	Number of octets the log object and its records now occupy in the MIS (reportable but cannot be modified).
Max Size (Bytes)	No	Each log object has a maximum size and an attribute that indicates whether its maximum size has been reached. A maximum size set to zero indicates no limit. Size is expressed in bytes.
Full Actions	Yes	When the log object is full, it either starts to overwrite the oldest records or stops accepting new records, according to the value of this attribute.

A notification record is logged only if all the following criteria are met:

- The notification is acceptable to the log object's discriminator.
- The log object has the capacity to receive the record or permits overwriting.
- The log object is unlocked.

If a notification is acceptable to the log filters of several log objects, it is recorded in each of them.

Note – Creating multiple logs with overlapping criteria has the potential for storing duplicate copies of notifications. For example, if you create a log, `SNMPLog`, to record `enterpriseSpecificTrap` notifications, you would be storing duplicate copies of enterprise-specific traps since these are, by default, logged to `AlarmLog`. You could avoid this duplication, in this case, if you were to add `enterpriseSpecificTrap` to the event types excluded from the `AlarmLog`.

5.1.2 Log Records Generated by Nerve Center Request Actions

Nerve Center requests are based on templates that are written in Request Condition Language (RCL). Among the functions available in RCL are `alarm()`, `alarmStr()`, and `alarmOi()`, which generate a Nerve Center alarm. When a running Nerve Center request invokes any of those functions, the default `AlarmLog` log object creates log records to record the alarms. The RCL alarm-logging functions are described in Chapter 22. In the Event Logs tool, you can modify the log filter for the default log object to filter out these alarms. Alternatively, you can, for example, create a new log object that accepts only Nerve Center alarms.

5.2 Starting the Event Logs Tool

The Event Logs tool may be started by:

- Selecting the Event Logs tool icon in the Network Tools window
- Selecting Tools ➔ Event Logs in other Solstice EM tools (for example, the Network Views or Log Entries)
- Entering the following command from the command line:

Note – You must source `/opt/SUNWconn/bin/emenv.csh` before running the command.

```
hostname% em_logmgr [-host <hostname>] [-logobj <fdn>]
```

If you start the Event Logs tool from the command line, and you are a non-root user, you might receive a Login window, depending upon whether or not password authentication is turned on. To proceed, enter your password and click OK. Your access to the Event Logs tool functions depends on the permissions granted to you through Access Control. For more information about password authentication and granting permissions, refer to Chapter 6 of *Management Information Server (MIS) Guide*.

The optional parameters of the `em_logmgr` command are described in the following table.

TABLE 5-2 Command-Line Options for the `em_logmgr` Command

Option	Description
-help	Print list of options (with descriptions) for the <code>em_logmgr</code> command.
-host <hostname>	Specify the <hostname> of a remote MIS. You can specify an IP address as the <hostname>.
-c <filename>	Specify the <filename> of the configuration file.
-logobj <fdn>	Display the log objects of the MIS specified in <fdn> in the main window summary table.

Before starting, the Event Logs tool looks for the `.em_logmgr.cf` configuration file in your home directory. If this file is not found, the default Event Logs tool properties are used. The format of the configuration file is described in “Event Logs Tool Configuration File” on page 21.”

5.3 Using the Event Logs Tool

Before using Event Logs tool, decide on the types of event notifications you want to log. To create a new log object, use the Event Logs tool to name the log object and fill in the modifiable attributes described in TABLE 5-1.

All of the log attributes are simple, except for the log filter. Specify the log filter using the ASN.1 syntax for a CMIS Filter. The easiest way to create a log filter is to copy a construct from an existing log and paste it into the log filter field in the Event Logs tool’s Log Creation window, where you can modify the construct to suit your needs.

Log filters are discussed in detail, with examples presented, in Section 5.4, “Defining the CMIS Filter.”

5.3.1 Accessing Logs on a Remote MIS

The Event Logs tool can be used to create or modify logs on the local MIS or a remote MIS.

▼ To Display Logs on a Specified Remote MIS

- **Select File → Customize Tools** menu to retrieve the **Properties** window, shown in the following figure, which is used to choose the locations from which the log objects are to be read.

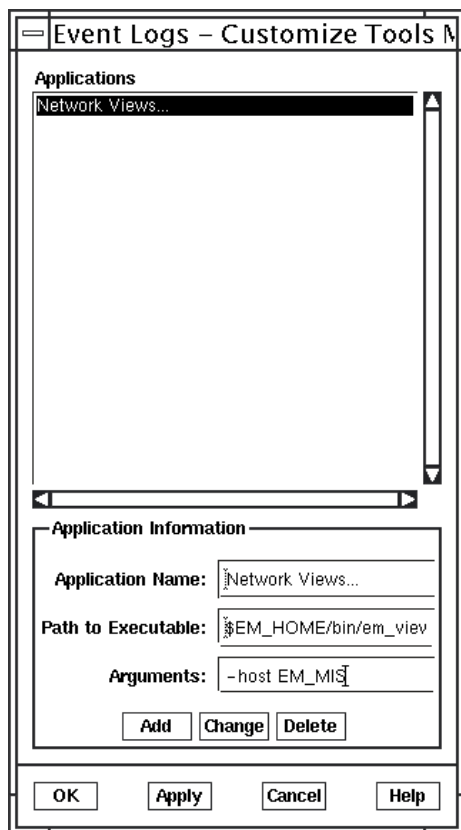


FIGURE 5-1 Customize Tools Menu

From this window, you can:

- Specify the name of a remote MIS, which will be shown as an additional FDN in the Available FDNs list. The logs from the specified MIS will also be available for you to see.
- Specify the location(s) from which the Event Logs tool will read the log objects by selecting it in the Available FDNs list and clicking on the display.

Once you specify the location(s), click either OK or Apply at the bottom of the window, and the log object information will be displayed in the Event Logs tool table.

▼ To Create a New Log

1. **Select Actions → Create Log from the Event Logs window.**

2. **Click Create.**

Fields in this window allow you to select the following log attributes:

- Enter the name of the MIS on which you are creating the log object in the MIS field. To create a log object on a remote MIS, you must first connect to that MIS by using the MIS Connections. For more information, refer to Chapter 6 of *Management Information Server (MIS) Guide*.
- Enter the name of the log object in the Log Name field.
- Enter the maximum log size (in octets) in the Maximum Size field. An entry of 0 indicates no limit to the log size. Such an entry poses the obvious danger of overwhelming your storage space.
- Select either On or Off for the Event Capture parameter. This parameter determines whether the log object can be written to (on) or not (off). Most often, you leave this parameter in its default, on state. Select locked to prevent writing to the log object.
- Select either On or Off for the WrapWhenFull parameter. This parameter determines what the Event Logs tool does when the log object reaches the maximum (maxLogSize) size. In the on state, the log object wraps around to the beginning of the file and overwrites existing log records. When turned off, no new records will be created for this log object.

In the example in the following figure, a new log called SNMLog is being created to log SunNet Manager event notifications (snmAlarmEvents).

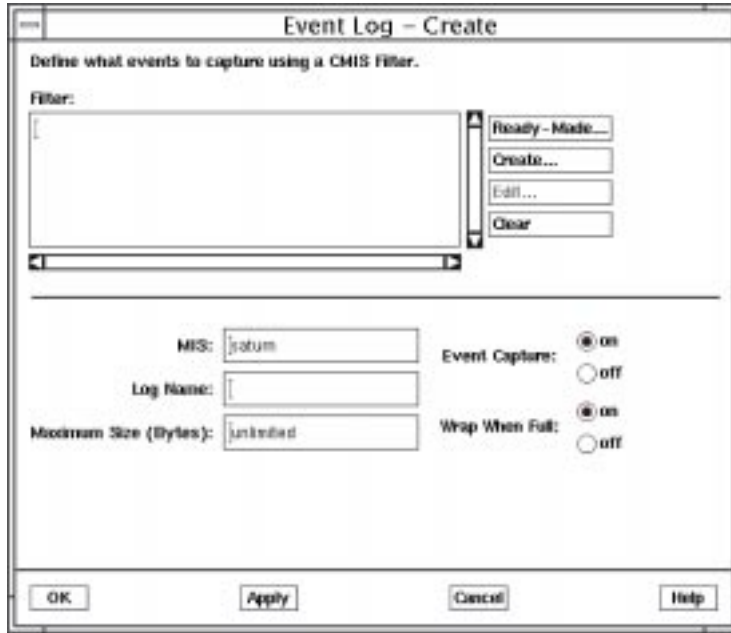


FIGURE 5-2 Creating a New Log

3. To define the log filter, click Create to invoke the CMIS Filter window.

A log filter is a Common Management Information Service (CMIS) filter. A notification is appended to a log object only if it passes the log's CMIS filter. A CMIS filter defines a test that evaluates to either true or false for each notification tested. (Refer to Section 5.4, "Defining the CMIS Filter," for detailed information about creating log filters.) The 'item', 'and', 'not', and 'or' buttons are used to add components to the CMIS filter. In the following figure, the Item button has been selected. This invokes the CMIS Filter Item window, as shown. In this example, a simple discriminator that selects all and only SNM events is the goal. Thus, the equality operator has been selected and snmAlarmEvent is entered as the target attribute value. Clicking OK adds the item to the CMIS Filter window.

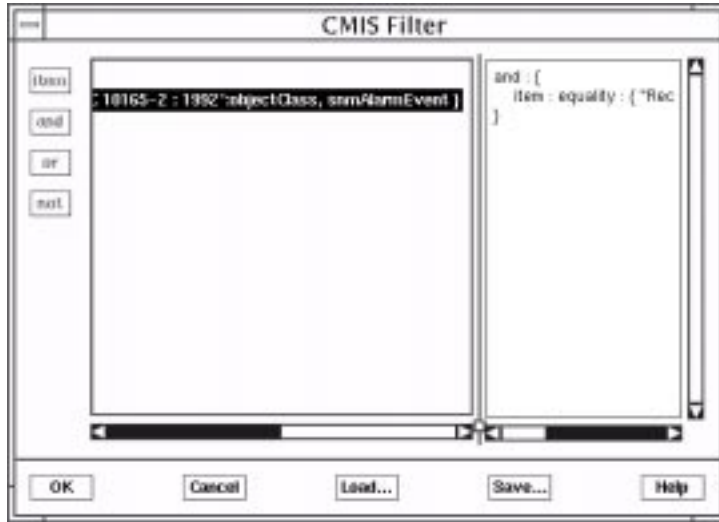


FIGURE 5-3 Defining a Discriminator to Log SNM Events

4. Click **OK** in the **CMIS Filter** window to add the log filter to the **Create Log** window.
5. Enter the **Log Name** and click **OK** or **Apply** in the **Create Log** window to add the new log to the **MIS**.

▼ To Delete a Log

1. To delete a log, select the log object (one or more) from the main window.
2. Select **Actions** → **Delete** to remove the selected log object(s) from the **MIS**.
Deleting a log object also deletes its corresponding log record. You will be prompted to confirm your action before the log object and its corresponding log record are deleted.

▼ To Modify a Log's Properties

1. To modify the properties of a log (such as the maximum size), select the log object from the main window.

2. Select Actions → Properties to modify or view the properties information about the selected log object(s).

This will bring up the Event Logs-Properties window, for that particular Log Object, allowing you to modify the selected log object as necessary. If multiple log objects are selected, then the Log Object window will display only the properties for the first log object selected.

- **To modify the log filter for the selected log, click Edit.**

The CMIS Filter window will be displayed. In the example in the following figure, the default AlarmLog discriminator is being modified to exclude enterpriseSpecificTrap notifications.

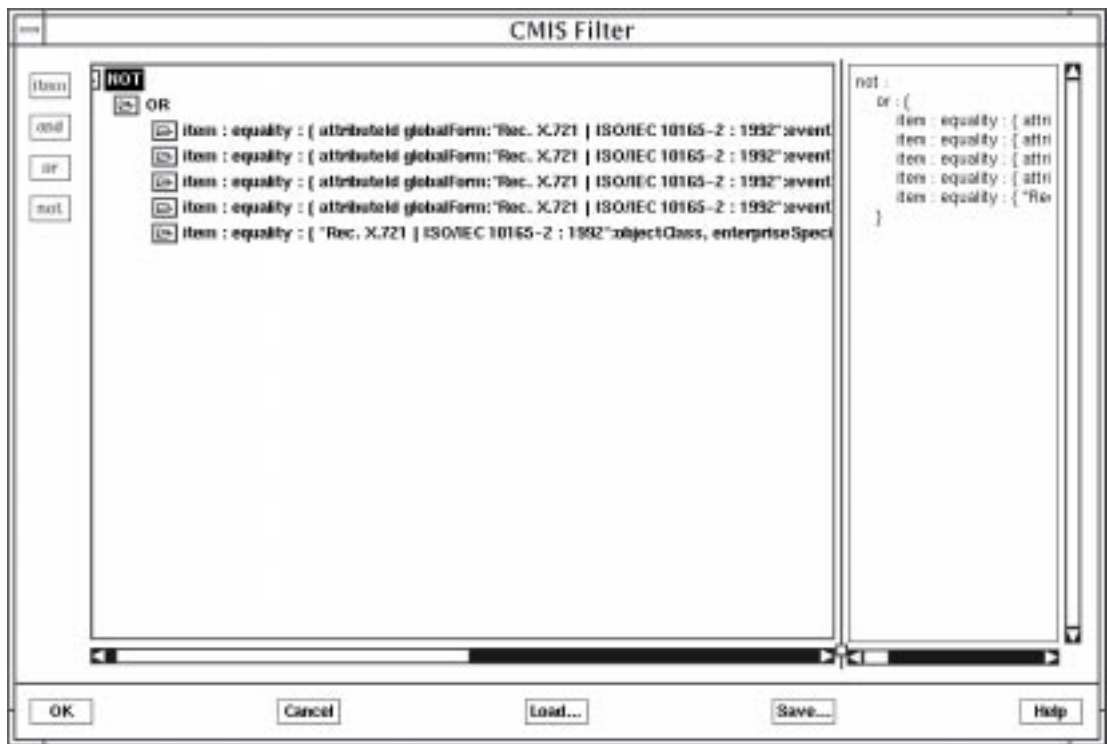


FIGURE 5-4 Modifying a Log's log filter

▼ To Export Logs to a File

- **Select File → Export to a File to save the log records to a file. This brings up a file selection window from which you can do one of the following:**
 - Select the path and file name to which you want to save the log records.
 - Enter the path and file name of the file in the Selection field.

The records are stored in the historical log format specified in Chapter 6 of *Management Information Server (MIS) Guide*.

This option is grayed out unless one or more of the log objects in the main window is selected.

5.3.2 Importing Logs from a File into the Event Logs Tool

You can import log records from other log files, such as the Alarm Log, into the Event Logs tool.

▼ To Import Logs

1. **Select File → Import from the Event Logs window to read exported log records.**

The log records in the file you want to import into the Event Logs window *must* be in the MIS.
2. **Select the desired file from the list or enter the name of the file in the Selection field.**

The log record will be imported into the Event Logs window.

The file must be in the historical log format specified in Chapter 6 of *Management Information Server (MIS) Guide*.

Note – `em_imex` does not create a deleted Log.

5.3.3 Configuring Display of Log Properties

The View Properties window is used to configure the way in which log objects are displayed in the Event Logs tool main window.

- **Select View → Column Headings in the Event Logs window to open the Event Logs - Column Headings window, shown in the following figure.**

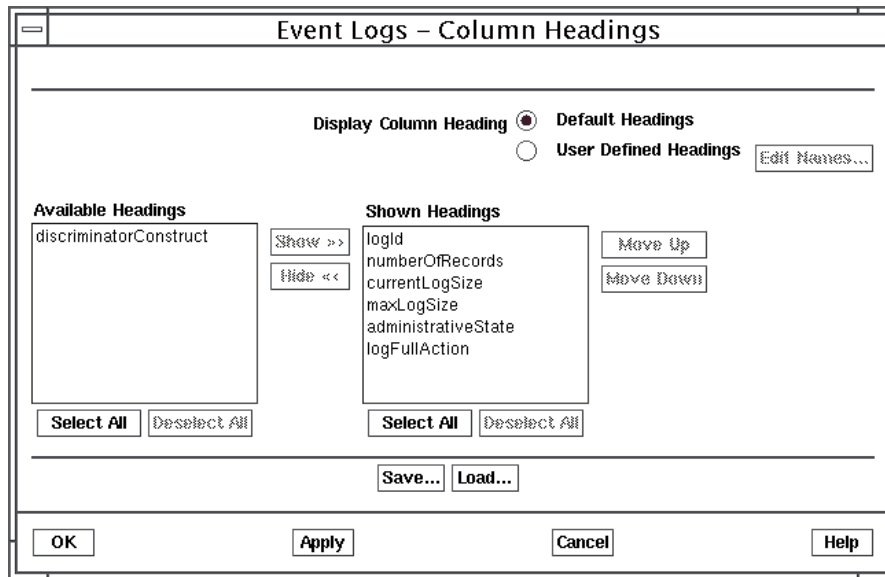


FIGURE 5-5 Viewing Log Objects in the Column Headings Window

5.3.4 Adding Tools to the Event Logs Menu

You can add other tools to the Event Logs window's Tools menu.

- 1. To add another tool, select File → Customize Tools Menu from the Event Logs window's Tools menu.**

This displays the Customize Tools window.

- 2. Select the Tool you would like to add to the Event Logs menu.**
- 3. Click Add, then OK or Apply to complete your addition.**

If you click OK, you dismiss the window. If you click Apply, the window remains and you can add more tools by repeating Step 2 and Step 3.

The Tool Name field is the name that is added to the Tools menu.

In the example in the following figure, clicking Add and then OK or Apply will add the Network Views to the Event Logs window's Tools menu.

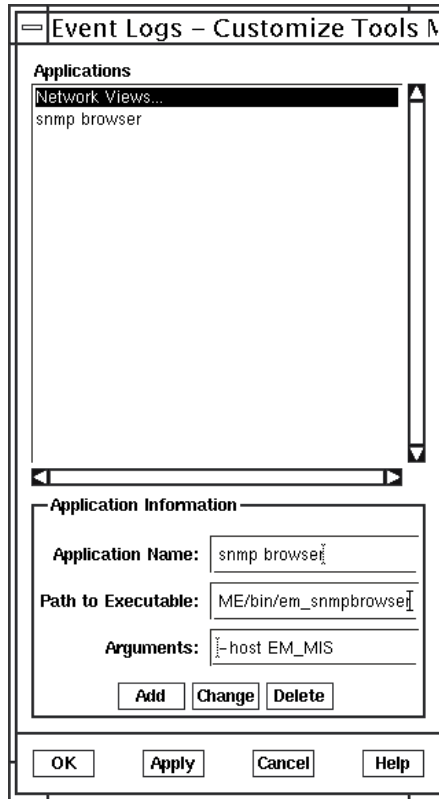


FIGURE 5-6 Customize Tools Menu Window

5.4 Defining the CMIS Filter

A log filter is an expression used to decide which notification types will be accepted by a particular log object. The general rules for such an expression are contained in the ASN.1 definition of a CMIS filter, in ISO 9595.¹

1. ISO/IEC 9595. Information technology – Open systems interconnection – Common Management information Service definition. 1991.

Within a log filter, a single test is called an item. Within an item, a relationship is written with the comparison operator preceding a pair of curly braces, which enclose an operand. The following is the general form:

```
item : <operator> : { <operand> }
```

The term *item* is one of four keywords that can be used to specify the format of a CMIS filter. These format specifiers can be *item*, *and*, *or*, or *not*. The following table defines these terms.

TABLE 5-3 Format Specifier Definitions

Name	Descriptions
item	A single <code>FilterItem</code> that is a choice of a test for equality, less than or equal, greater than or equal, substring matching, or presence in. These filter words are described in TABLE 5-4.
and	The logical AND of a set of CMIS filters.
or	The logical OR of a set of CMIS filters.
not	The negation of the sense of one CMIS filter.

The *<operator>* in the preceding example is the word in a given filter item. The following table defines the operators.

TABLE 5-4 Operator Definitions

Name	Description
equality	Is the item we are filtering equal to this operand?
substrings	Is the string under consideration match the beginning, end, or any part of the string in the operand? This breaks down to the operators <code>initialstring</code> , <code>anystring</code> , and <code>finalstring</code> .
greaterOrEqual	Is the item we are filtering greater than or equal to this operand?
lessOrEqual	Is the item we are filtering less than or equal to this operand?
present	Is the item we are filtering present in the operand?

The *<operand>* is an attribute, which, in GDMO terms, is a name-value pair. So, we can further refine the general form as:

```
item : <operator> : { <name_string>, <value_string> }
```

Using the `and` or `or` format specifier, you can build nested filters. These are of the form:

```
and : (or or)
{
  item : <operator> : { <name_stringA>, <value_stringA> },
  item : <operator> : { <name_stringB>, <value_stringB> }
}
```

As an example of a filter used as a log filter, to test whether the subject of an event notification is a `test`, the expression might be:

```
item : equality : {objectClass, test }
```

In the preceding example, `objectClass` (the attribute name) and `test` (the attribute value) combine to form a single attribute, against which an event notification will be tested for equality.

Use the format specifier `and`, defined in TABLE 5-3, to perform a logical AND on two filter items. For example, to test whether the subject of notification is a `test` and the severity of the notification is `minor`, specify the following:

```
and : { item : equality : {objectClass, test },
        item : equality : {perceivedSeverity,   minor   } }
```

Multiple items are separated by a comma (last character in first line above).

5.4.1 A CMIS Filter That Accepts Notifications of a Specific Type

The following example tests the notification type in an incoming event notification. The OID `{ 2 9 3 2 7 14 }` specifies an event notification. The notification type is identified by the OID `{ 2 9 3 2 10 <x> }`:

```
item : equality : { globalForm : { 2 9 3 2 7 14 }, { 2 9 3 2 10 x} }
```

In this expression, substitute one of the values 1 through 15 for <x> from the list of notification types shown in the following table.

TABLE 5-5 Notification Types and Numbers

Number	Notification type
1	attributeValueChange
2	communicationsAlarm
3	environmentalAlarm
4	equipmentAlarm
5	integrityViolation
6	objectCreation
7	objectDeletion
8	operationalViolation
9	physicalViolation
10	processingErrorAlarm
11	qualityofServiceAlarm
12	relationshipChange
13	securityServiceOrMechanismViolation
14	stateChange
15	timeDomainViolation

Note – Within a log filter, you can express OIDs in text, rather than numeric form.

In addition to the OSI-standard notifications shown in the above table, the alarmLog object accepts alarms of type nerveCenterAlarm, the OID for which is: 1.3.6.1.4.1.42.2.2.2.8.3.111.

5.4.2 CMIS Filter with Multiple ANDs

The following example is a sample log filter that uses the and format specifier to combine three filter items.

CODE EXAMPLE 5-1 Sample Log Filter

```
and :
{
  item : equality :
  {
    attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 :
1992":managedObjectClass,
    attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":log
  },

  item : equality :
  {
    attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 :
1992":managedObjectInstance,
    attributeValue distinguishedName :
    {
      {
        attributeId "Rec. X.721 | ISO/IEC 10165-2 : 1992":systemId,
        attributeValue name : "minerva"
      }
    },
  },

  item : equality :
  {
    attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992":eventType,
    attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 :
1992":attributeValueChange
  }
}
```

The effect of the preceding construct is as follows: If there is an event of the log object type `attributeValueChange` to `minerva`, log that event.

5.4.3 A CMIS Filter That Accepts All Notifications

The following exploits the fact that `and` over an empty set is the identity element for `and`, namely 1. This produces a log filter that accepts all notifications:

```
and : { }
```

This example should never be used in actual practice, because you will likely overwhelm your machine resources in logging data.

5.4.4 A CMIS Filter That Accepts No Notifications

The corresponding identity element for `or` yields a log filter that accepts nothing:

```
or : { }
```

As with the `and` example, this construct is shown for tutorial purposes only, and is not intended for use in actual practice.

5.5 Sample CMIS Filters

The following subsections present the log filters you can enter in the Event Logs tool's Log Creation to create log objects that collect log records for given event notifications.

To complete the creation of a log object, in addition to the log filter, you must fill in values for `maxLogSize`, `logFullAction`, `administrativeState`, and `operationalState`. The choices of these values depend on your specific needs. The value for `maxLogSize` depends on your logging requirements and your storage resources.

5.5.1 Creation of an Object Instance

To create a log object that logs notifications reporting the creation of object instances, use the following log filter:

```
item : equality:
{
    attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 :
1992" :
    eventType,
    attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2
: 1992" :
    objectCreation
}
```

5.5.2 Deletion of an Object Instance

To create a log object that logs notifications reporting the deletion of object instances, use the following log filter:

```
item : equality:
{
    attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 :
1992" :
    eventType,
    attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2
: 1992" :
    objectDeletion
}
```

5.5.3 Attribute Value Change of an Object Instance

To create a log object that logs notifications reporting attribute value changes of object instances, use the following log filter:

```
item : equality:
{
  attributeId globalForm : "Rec. X.721 | ISO/IEC 10165-2 : 1992" :
    eventType,
    attributeValue globalForm : "Rec. X.721 | ISO/IEC 10165-2 :
1992" :
      attributeValueChange
}
```

5.5.4 State Changes Received From Agent

To create a log object that logs notifications reporting the state changes received from agents, use the following log filter:

```
and:
{
  not:
  {
    item : equality:
    {
      eventType,
      objectCreation
    }
  },
  not:
  {
    item : equality:
    {
      eventType,
      objectDeletion
    }
  },
  not:
  {
    item : equality:
    {
      eventType,
      attributeValueChange
    }
  }
}
```



```
}  
}  
}
```

5.6 Event Logs Tool Configuration File

Upon starting, the Event Logs tool looks for the `.em_logmgr.cf` configuration file in your home directory; otherwise, it looks for it in the `$EM_HOME/config` directory. (The file names are the same in the `config` directory except for the absence of the initial dot.) If the configuration file is not found, the Event Logs tool uses the default properties.

The alphanumeric characters in each line of the configuration file must begin at the left edge. Each statement must be on a separate line.

The configuration file has the following format:

CODE EXAMPLE 5-2 Event Logs Tool Configuration File

```
display_name=nickname  
label_name=default_name  
show_doc_names=show  
show_oids=oid  
attr_name=logId  
logId.name=Log Name  
logId.position=1  
logId.displayed=true  
logId.sort_pos=-1  
logId.width=10  
attr_name=numberOfRecords  
numberOfRecords.name=Records  
numberOfRecords.position=2  
numberOfRecords.displayed=true  
numberOfRecords.sort_pos=-1  
numberOfRecords.width=10  
attr_name=administrativeState  
administrativeState.name=Admin State  
administrativeState.position=3  
administrativeState.displayed=true  
administrativeState.sort_pos=-1  
administrativeState.width=12  
attr_name=currentLogSize  
currentLogSize.name=Current Size  
currentLogSize.position=4  
currentLogSize.displayed=true
```

CODE EXAMPLE 5-2 Event Logs Tool Configuration File *(Continued)*

```
currentLogSize.sort_pos=-1
currentLogSize.width=9
attr_name=maxLogSize
maxLogSize.name=Max Size
maxLogSize.position=5
maxLogSize.displayed=true
maxLogSize.sort_pos=-1
maxLogSize.width=9
attr_name=logFullAction
logFullAction.name=Full Action
logFullAction.position=6
logFullAction.displayed=true
logFullAction.sort_pos=-1
logFullAction.width=8
attr_name=discriminatorConstruct
discriminatorConstruct.name=Discriminator
discriminatorConstruct.position=7
discriminatorConstruct.displayed=true
discriminatorConstruct.sort_pos=-1
discriminatorConstruct.width=75
```

The information in this file corresponds to the Properties and View Properties windows. Although this file shouldn't be modified manually, you might want to change the width field, which determines the width of the cells in the table.

PART III Network Management Protocol Support

Managing Devices Using RPC Agents

Solstice Enterprise Manager (Solstice EM) 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.

This chapter describes the following topics:

- Section 6.1 “Types of RPC Agent Management” on page 6-1
- Section 6.2 “Preparing for Device Management with RPC Agents” on page 6-5
- Section 6.3 “RPC Management Protocol Adapter” on page 6-11
- Section 6.4 “RPC MPA Configuration Parameters” on page 6-12

6.1 Types of RPC Agent Management

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 Solstice 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 Solstice EM MIS is running.

As illustrated in the following figure, SNM agents and proxies use Remote Procedure Call (RPC) protocol to communicate with the Solstice EM MIS (via the RPC Management Protocol Adapter). However, an SNM proxy agent may use a different management protocol in gathering information from other agents.

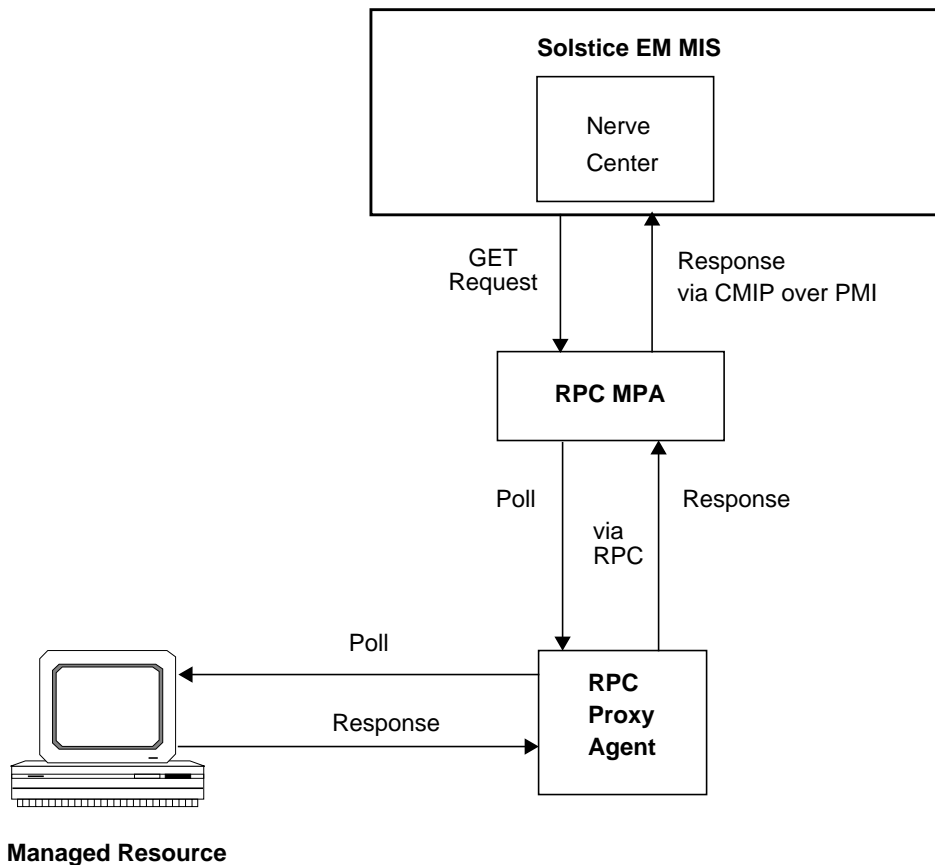


FIGURE 6-1 Communication With RPC Agents in Direct Polling Requests

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 15. Communication between manager and managed resource in a direct polling request is illustrated in FIGURE 6-1.
- **Offload polling activity to the RPC proxy agent**—A Nerve Center request can send a one-shot message, called an *SNM event request*, to a 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 by 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 a SNM event notification arrives at an MIS machine, this information is forwarded to the Solstice EM MIS by Solstice EM's SNM Event Forwarder (`em_snmfwd`). Communication between manager and agents in SNM event requests is illustrated in the following figure.

Building templates that use the Nerve Center's SNM event request capability is described in Chapter 17.

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 view changes. Event and network view information received by these SNM management stations can be forwarded to an Solstice EM MIS using Cooperative Consoles. This type of distributed management scenario is described in Chapter 7.

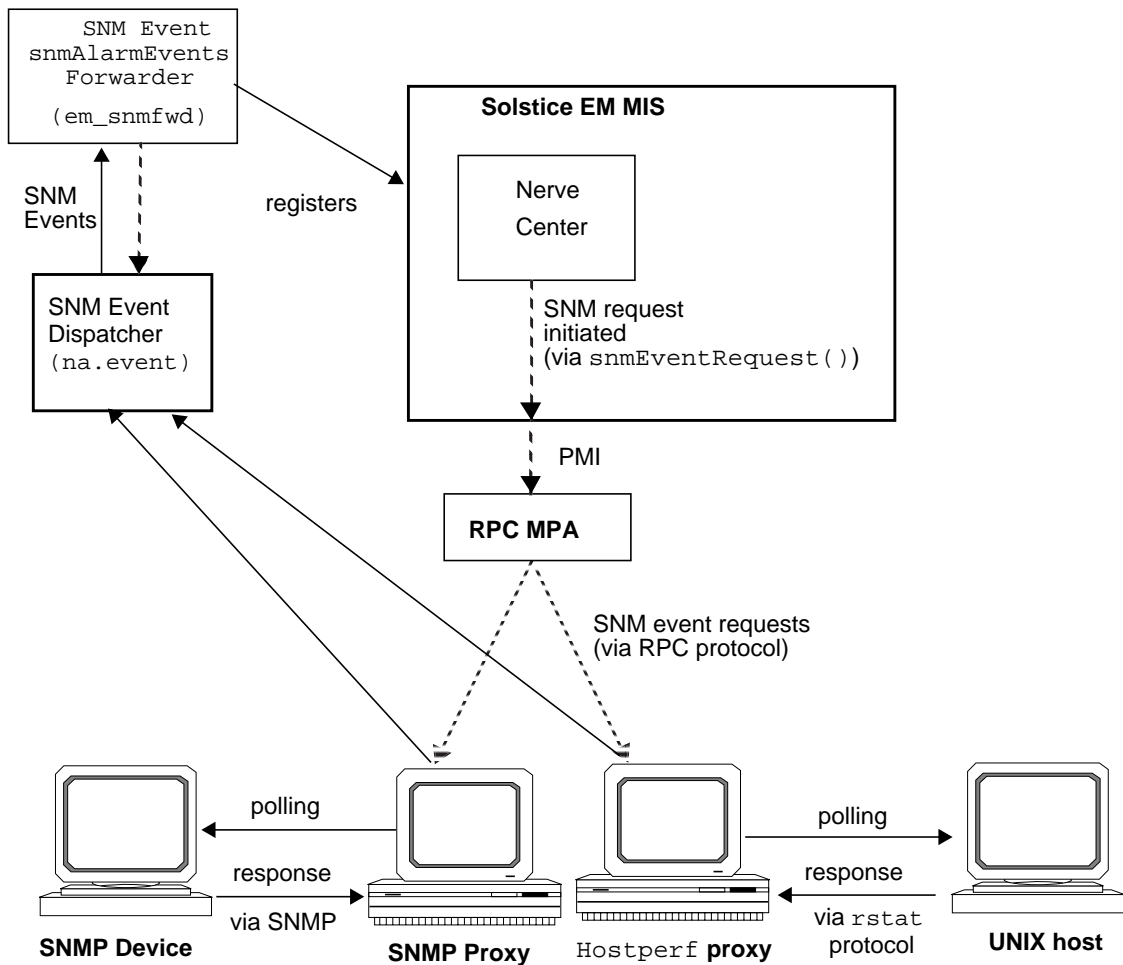


FIGURE 6-2 Using SNMP Event Requests with Solstice EM

6.2 Preparing for Device Management with RPC Agents

The first thing you must do is prepare your device management with RPC agents.

▼ To Prepare the Device Management with RPC Agents

1. Install and configure 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

`SUNWsnmag` is installed by default during installation of Solstice EM 4.1. Use `pkgadd` on other machines to install the RPC agents package (machines to be managed or proxy systems).

To install agents on a machine 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 Chapter 6 in *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`), see Chapter 10.

2. Add 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 4.

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 vendor, 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 8 of *Management Information Server (MIS) Guide*.

However, you will only need to do a schema-to-GDMO conversion if you have SNM schemas not provided with Solstice EM. For all SNM schemas shipped with Solstice 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, see Chapter 10.

3. Install and configure the RPC MPA.

The RPC Management Protocol Adapter (MPA) is a protocol translator that enables communications between the MIS SunNet Manager proxy agents using Remote Procedure Call (RPC) protocol. By default, the RPC MPA is installed on the same machine as the MIS. However, to improve performance you may want to distribute the RPC MPA to another machine. For information on installation of the RPC MPA, refer to Chapter 6 in *Installation Guide*. For more information on the RPC MPA, refer to Section 6.3 “RPC Management Protocol Adapter.”

4. Configure the managed object in the MIS.

Note – This is applicable only for proxy agents.

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 Network Discovery to populate your MIS, you can configure Network Discovery to query hosts for RPC agents and automatically configure objects to indicate RPC agent support when it adds them to the database. The RPC agent selection sheet in the Network Discovery Properties window is shown in FIGURE 6-3. You can also select the host that will be configured as the proxy system for RPC-manageable devices that are added to the MIS by Network Discovery. When SNM event requests are targeted at these devices, the RPC proxy agents on the “default proxy” are used to conduct the threshold-checking of the target device. If `localhost` (the default) was selected during discovery, the RPC proxy agents on the MIS machine are used to carry out threshold-checking. In the example in the following figure, the machine `localhost` is selected to be the

proxy host for all devices added to the MIS through this use of Network Discovery. For more information on using the Network Discovery tool, refer to *Managing Your Network*.

- You can also manually configure RPC agent support for objects in the MIS using the Network Views Object Properties/Create Object. It is invoked from the Solstice EM Network Views. For more information, refer to Chapter 3 in *Managing Your Network*.

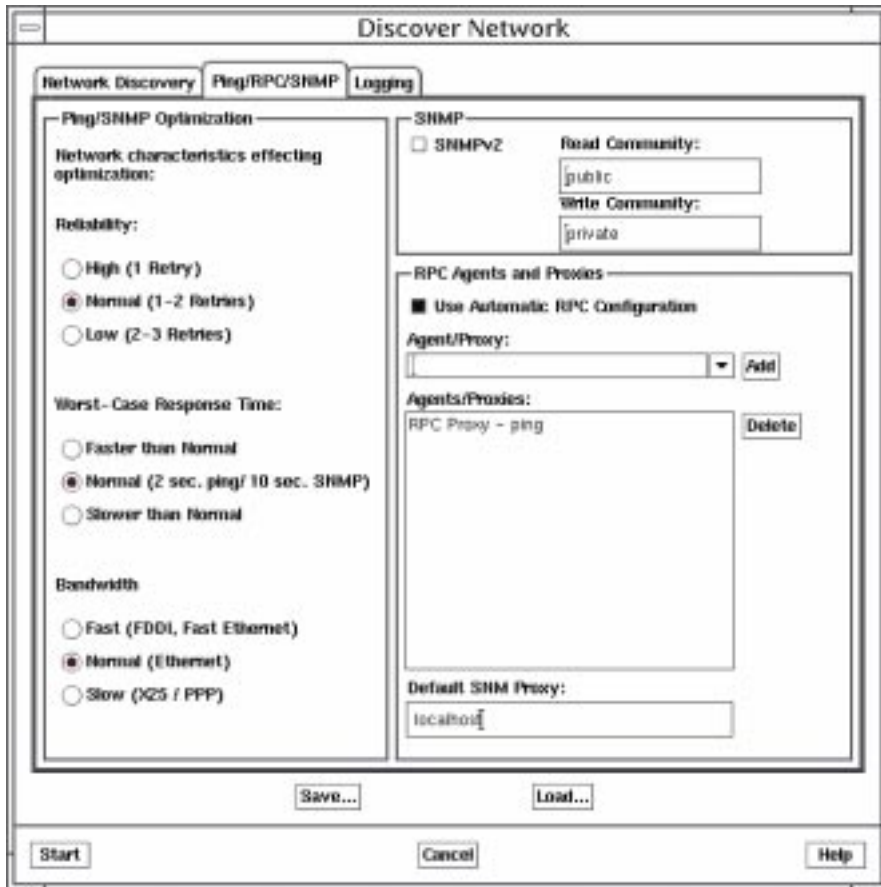


FIGURE 6-3 Selecting RPC Agents to be Configured During Network Discovery

5. Build request templates for RPC agents.

The Nerve Center module in the MIS contains the request-handling capabilities of Solstice EM. Nerve Center requests are based on request templates, which are built using the Design Advanced Requests tool. A key building block in request templates are request conditions—sets of instructions defined using the Solstice 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 in the following table. 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 15.
- For information on the Design Advanced Requests, see Chapter 18.
- For information on Request Condition Language, see Chapter 20.
- 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 17.
- For additional information on RCL functions that can be used in building request templates, see Chapter 22.

TABLE 6-1 Ready-to-Use RPC Request Templates

Template Name	Description
AdminOperStatusUp	AdminOperStatusUp sends SNM event requests that use the RPC SNMP proxy agent (<code>na.snmp</code>) to poll the interfaces on the target router. When <code>ifOperStatus</code> is down (indicating the interface is not operational) and <code>ifAdminStatus</code> is up (has not been intentionally downed by an administrator), the request posts a critical alarm. If a subsequent SNM event indicates that <code>ifOperStatus</code> is up or <code>ifAdminStatus</code> is down, a warning alarm is posted to indicate the interface is up after having been down.
CheckCPU	CheckCPU sends an SNM event request that uses the RPC <code>hostperf</code> proxy agent (<code>na.hostperf</code>) to check for CPU usage greater than 50%. If an SNM event arrives, a minor alarm is posted and another SNM event request is started to check the target host for CPU usage greater than 80%. If no SNM event arrives within 3 minutes, the second event request is killed. If a greater-than-80%-usage event arrives within the three minute interval, a major alarm is posted and the request continues to check for CPU usage greater than 80%.
DeviceReachablePing	DeviceReachablePing initiates SNM event request to check for device reachability and subscribes for incoming <code>snmAlarmEvents</code> . Posts a major alarm if the device is not reachable and a warning alarm when device is available after having been unreachable.

TABLE 6-1 Ready-to-Use RPC Request Templates *(Continued)*

Template Name	Description
DiskPartitionsFull	DiskPartitionsFull directly polls the RPC diskinfo agent (na.diskinfo) to obtain the diskSpace table for the target host. The request checks the capacity attribute for locally-mounted disk partitions (other than /proc, /tmp, and /fd) and posts a critical alarm for each disk partition that has greater than 95% of its capacity in use.
PingUpOrDown	PingUpOrDown directly polls for reachability of device using the RPC ping proxy agent (na.ping). It sends a critical alarm if the device is not reachable. If the device becomes reachable after not being reachable, a clear alarm is sent to clear the previous critical alarm.
RPC_Diskinfo_ DiskPartitionsFull	RPC_Diskinfo_DiskPartitionsFull sends multiple SNM event requests to the SNM Diskinfo agent. A capacity request is sent for each partition that is contained in the following disk_list of RCL attributes. Each partition in the list may have two threshold values—one for a minor alarm, and one for a critical alarm. (Note: Each partition may have different threshold values for the minor and critical alarms. In order to change the alarm severities and alarm messages, the user must modify the condition process_capacityEvents).
RPC_Diskinfo_ WatchAllPartitions	This template reads the entire disk table from the target agent by polling the diskinfo agent. An SNM event request is then sent to each partition checking for a capacity of greater than 90%.
RPC_Hostperf_VerifyProxyA gent RPC_Ping_VerifyProxy Agent RPC_SNMP_VerifyProxy Agent	These three templates are used to verify that the remote proxy agents are servicing requests. These templates should be launched against the devices which are running the SNM proxy agents. Every poll period the template will send an SNM event request which forces an event. If the proxy does not respond or if an event is never returned, a NerveCenter alarm is posted. The template will continue to check for a response and will post an alarm if the proxy starts servicing again.
RPC_Hostperf_WatchCPU	This template sends an SNM event request to the SNM Hostperf proxy agent. The proxy agent polls the target for cpu percentage and posts an alarm when it exceeds 70%. (Note: This template will also inform Solstice EM via an event when the target agent is down.)

TABLE 6-1 Ready-to-Use RPC Request Templates *(Continued)*

Template Name	Description
RPC_MibII_Interface CollisionDetection	This template polls the RPC SNM SNMP proxy agent for the ifOutput table every poll period. When the collision percentage is calculated, an alarm may be posted if the collisions exceed the minor or critical rate.
RPC_MibII_InterfacePing Triptime	This template polls the target and reads in the interface and IP tables. One RPC event request is sent to each IP address, checking for an ICMP round trip time of greater than 300ms. If the round trip time exceeds 300ms a NerveCenter alarm is posted. When the round trip time drops back below 300ms a NerveCenter alarm of warning is posted.
RPC_MibII_InterfaceStatus	RPC_MibII_InterfaceStatus sends SNM event requests that use the RPC SNMP proxy agent (na.snmp) to poll the interfaces on the target router. This template has some user-configurable parameters which are found in the condition mibII_interface_watch_variables.
RPC_MibII_sysUpTime_Agent Up	This template sends a SNM event request to the SNM SNMP proxy agent. The proxy agent will poll the target to make sure that the SNMP agent is responding and to check whether the sysUpTime has not increased, indicating that the system has rebooted. The parameters below are configurable for the template and may be found in the condition MIBII_SNMP_AgentUp_variables.

TABLE 6-1 Ready-to-Use RPC Request Templates *(Continued)*

Template Name	Description
RPC_PingReachableRetry	This template sends a SNM event request to SNM Ping proxy agent. The proxy agent will check the target agent for ping reachability every poll period.
RPC_SunMIB_Process Watch	This template uses the Sun process table extensions to MIBII. The template reads in the process table from the remote agent using the SNMP proxy agent. One SNM event request is then sent for each process found in the <code>process_list</code> RCL variable. When that process is no longer found on the remote agent, the template will post a NerveCenter alarm and execute the action in the <code>action_list</code> RCL variable associated to the process name. Note that <code>\$hostname</code> and <code>\$process_name</code> may be substituted in the action. This allows a process to be restarted if desirable.
SnmpPingBackoffReachable	<code>SnmpPingBackoffReachable</code> directly polls the SNMP daemon for the <code>sysUpTime</code> attribute to determine if the SNMP daemon is running. If there is no response to the initial poll, the request polls at a longer interval. If there is still no response, the request directly polls for reachability, using the RPC ping proxy agent (<code>na.ping</code>), to determine if the nonresponse is due to the SNMP agent being down or the unavailability of the host machine. The request generates a major alarm if the device will respond to a ping but there is no response from the SNMP agent. The request generates an “SNMP daemon back up” warning alarm if the SNMP agent responds after being down. The request generates a critical alarm if the target device does not respond to either the SNMP or ping polls. If the device becomes reachable after a period of unreachability, the request posts a “device back up” warning alarm.

6.3 RPC Management Protocol Adapter

The Remote Procedure Call (RPC) Management Protocol Adapter (MPA) provides the mechanism to get data and set attribute values for devices that are managed via RPC-based agents. The RPC MPA works as a proxy agent between the Solstice EM MIS and any device on the network having RPC agents installed.

The RPC MPA serves a major role in providing compatibility with SunNet Manager 2.2 or later products. Agents that were written for SNM 2.2 or later can be registered with the Solstice EM MIS, so that when a request is made, the RPC MPA will be called by the MIS to route the request via RPC calls to the appropriate SNM 2.2 or later agent on the appropriate host.

Note – SNM 2.2 or later schema files get compiled by the Schema compiler (em_snm2gdm) into GDMO descriptions which then get loaded into the Solstice EM MIS. This is described in Chapter 8 of *Management Information Server (MIS) Guide*.

When an RPC request is received by the Solstice EM MIS, the MIS will route the request to the RPC MPA. When the new request arrives from an MIS, the RPC MPA translates the CMIS-like message it received into a corresponding RPC request. It then sends the request to the specified SNM agent on the specified device and waits for a response. Upon receiving a response from the SNM agent, the RPC MPA translates the RPC message into a CMIS-like message, and sends it on to the MIS.

All communication between the MIS and the RPC MPA are CMIS-like. All communication between the RPC MPA and the manageable SNM agents are via RPC.

6.4 RPC MPA Configuration Parameters

The following configuration parameters for the RPC MPA are set during installation:

- **Default MIS host**—The name of the machine running the MIS that the MPA is to connect to.
- **Default MIS port**—The port used in communicating with the MIS (by default this is port 5555).
- **Default RPC MPA port**—The port on which the RPC MPA listens for incoming messages (by default this is port 5577).
- **RPC request timeout**—The length of time the RPC MPA waits for a response to a request sent to an RPC agent (by default, this is 15 seconds).
- **RPC retries**—The number of times the RPC MPA retries a request to an RPC agent if there is no response (by default this is zero).

The request and retry parameters determine when the RPC MPA determines that an RPC agent is unavailable.

Using Cooperative Consoles with Solstice EM

Cooperative Consoles provides the ability to forward information about critical network events from management stations running Site/SunNet/Domain Manager (SNM) to one or more *Solstice Enterprise Manager* (Solstice EM) managers.

This chapter describes the following topics:

- Section 7.1 “Cooperative Console Forwarding” on page 7-1
- Section 7.2 “Filtering Criteria for Information Forwarding” on page 7-3
- Section 7.3 “Cooperative Consoles Configuration and Operation” on page 7-4
- Section 7.4 “Receiving SunNet Manager Alarms” on page 7-7

7.1 Cooperative Console Forwarding

The supported configuration of Cooperative Consoles information forwarding between SNM and Solstice EM management stations is periphery-to-center. 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 Solstice EM MIS. The Solstice EM MIS thus functions as a central office “manager of managers.” A periphery-to-center configuration is illustrated in the following figure.

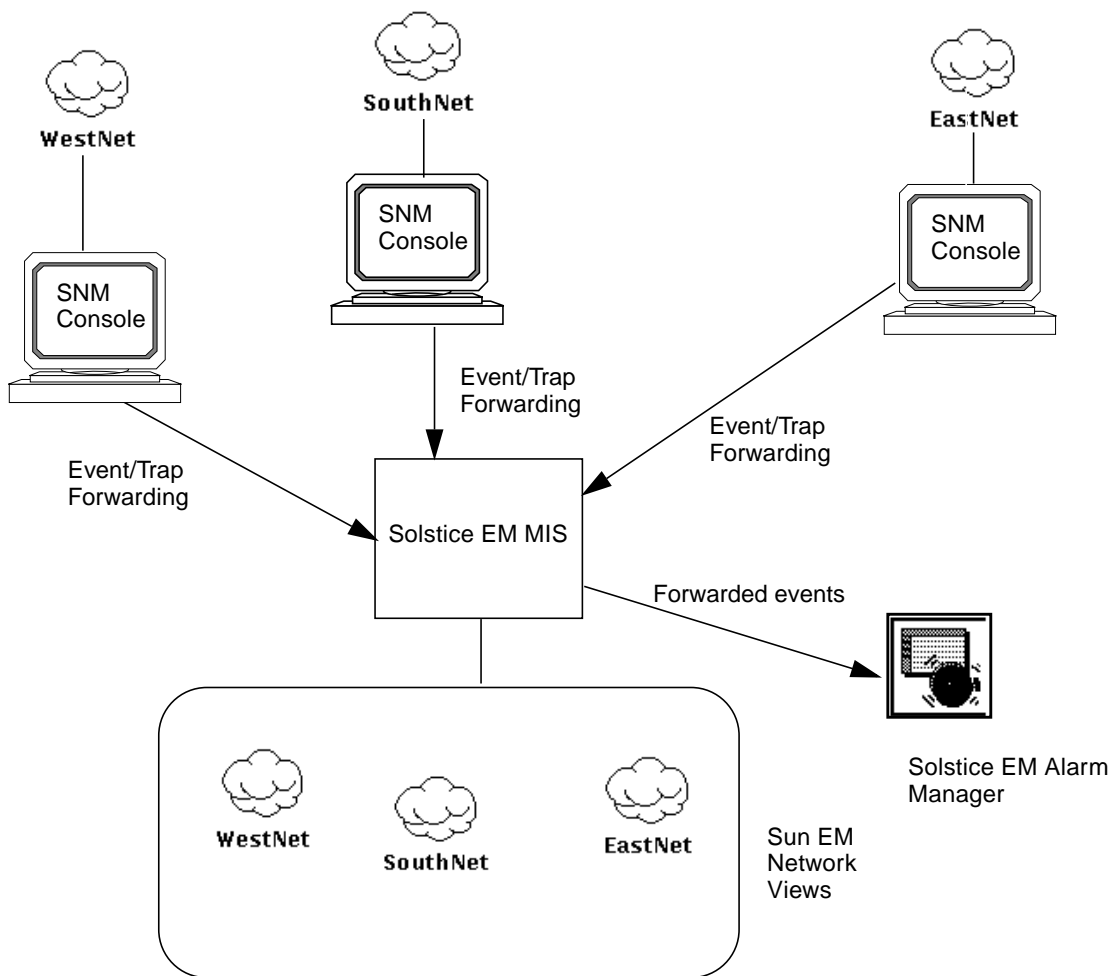


FIGURE 7-1 Forwarding of Information to Central Management Station

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 Solstice EM's SNMP trap daemon (`em_snmp-trap`) for distributed SNMP trap forwarding and configurable event type conversion. The Solstice EM SNMP trap daemon can forward SNMP traps to SNM Consoles as well as Solstice EM.

7.2 Filtering Criteria for Information Forwarding

The flexible filtering capabilities of Cooperative Consoles allow 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.

7.3 Cooperative Consoles Configuration and Operation

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

- **Receiver Application**—A Receiver is installed on the Solstice 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. Use the Cooperative Consoles 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 Solstice EM MIS to update the Solstice EM MIS to reflect changes in the views on the remote SNM Consoles. Solstice EM's support for the SunNet Manager database API is described in Chapter 8. If Cooperative Consoles forwarding of information has been set up to create a "mirror" on the Solstice 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 Solstice EM Network Views.
- **Sender Daemon**—A Sender daemon is installed on each SunNet Manager host that forwards event and topology information to the Solstice EM MIS. Cooperative Consoles' event and topology filters are used by the Sender daemon. The Cooperative Consoles Configuration Tool is used for sending SNM stations to configure these filters. The periphery-to-center configuration is the only configuration currently supported for Solstice EM. In this configuration, no Sender daemon is installed on the Solstice EM MIS machine.
- **Configuration Tool**—This is the user interface for configuring operation of the Sender and Receiver processes. Configuring the Receiver on the Solstice EM MIS machine also requires installation of the Cooperative Consoles Configuration Tool.

▼ To Set Up Cooperative Consoles on the Solstice EM MIS Machine

1. **Install the Cooperative Consoles Configuration Tool and Receiver packages.**

See the *Installation Guide* for detailed information.

2. Set your LD_LIBRARY_PATH to support the Receiver application.

Enter a command such as the following, to set the environment variable correctly:

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

Since the Cooperative Consoles Receiver is an SNM application, you should refer the general instructions for use of SNM applications, see Chapter 8.

3. Add the Cooperative Consoles Configuration Tool and the Cooperative Consoles Receiver to the Solstice EM Network Tools.

This is described in *Managing Your Network*. You will need to tell the Launcher the path to the Cooperative Consoles executables. The default path to the Cooperative Consoles Receiver is as follows:

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

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

5. Use the Cooperative Consoles Configuration Tool to set up the Receiver's Registration List on the Solstice EM MIS machine.

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

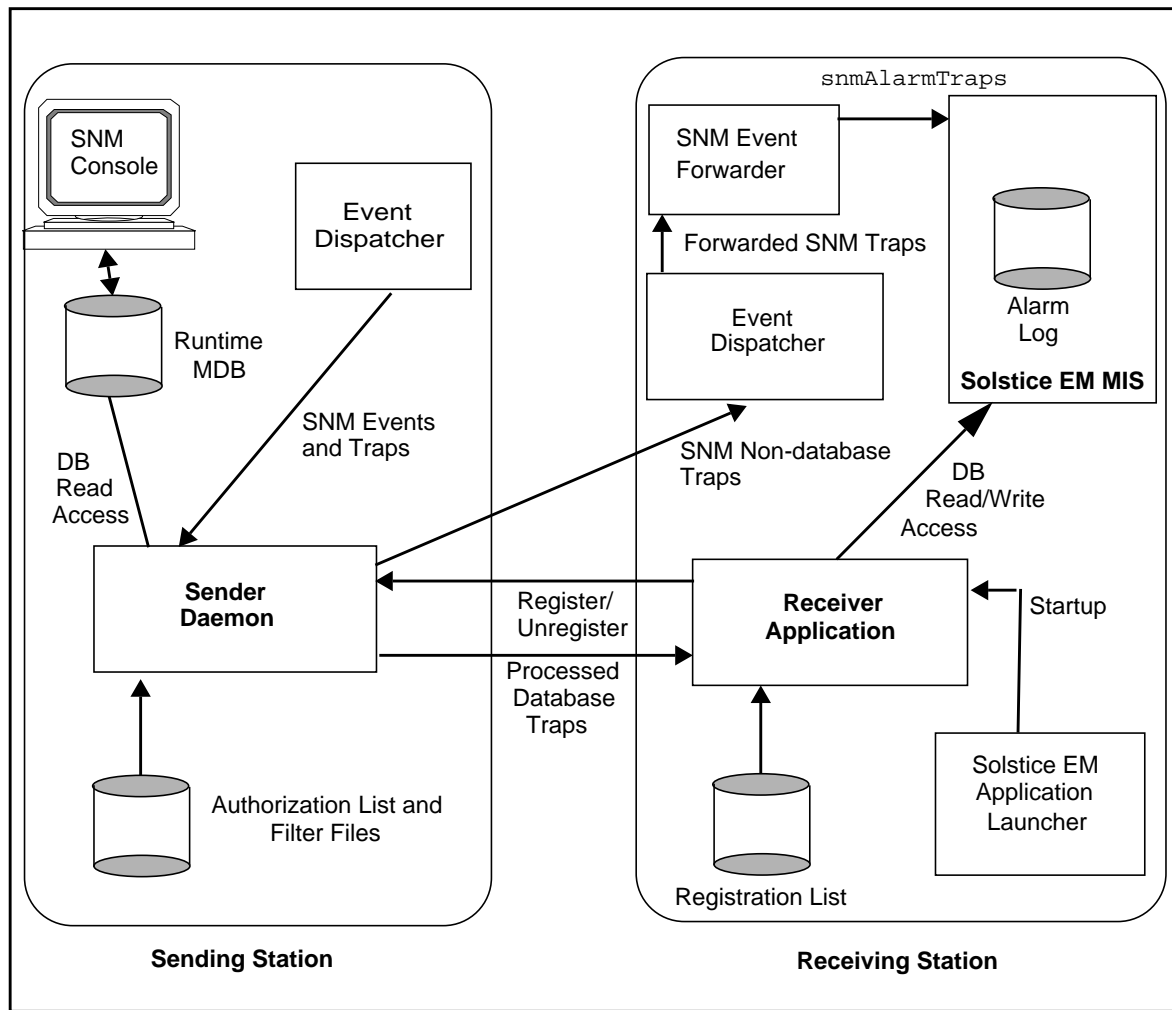


FIGURE 7-2 Information Forwarding From SNM Console to Solstice EM MIS

7.4 Receiving SunNet Manager Alarms

The Cooperative Consoles 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 the following table. The SNM Event Forwarder also translates dimming or blinking of glyphs into `perceivedSeverity` values, as indicated in the following table.

TABLE 7-1 Mapping of SNM Console Fault Indications to `perceivedSeverity` Values

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

The Alarm Service, which controls the fault status color of icons in the Network Views, 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 Network Views.

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 Solstice 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, see Chapter 17.

SunNet Manager Application Support

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

This chapter describes the following topics:

- Section 8.1 “Solstice EM Compatibility with SunNet Manager” on page 8-1
- Section 8.2 “Access to Solstice EM Features from SNM Applications” on page 8-4
- Section 8.3 “Adding an SNM Application to Solstice EM” on page 8-5
- Section 8.4 “Information for Configuring Specific SNM Applications” on page 8-11
- Section 8.5 “Importing an SNM Database into Solstice EM” on page 8-25
- Section 8.6 “Access to SNM Agents by SNM Applications” on page 8-25
- Section 8.7 “Access to SNM Agents by Solstice EM Applications” on page 8-28

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.

8.1 Solstice EM Compatibility with SunNet Manager

For the purpose of describing SNM/Solstice EM interoperability, we define an *SNM application* as an application that uses the SNM Application Programming Interface (API) to access the SNM database 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 prepend 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 Management Protocol Adapter (MPA) (see FIGURE 8-1). 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 8.3, “Adding an SNM Application to Solstice EM.”

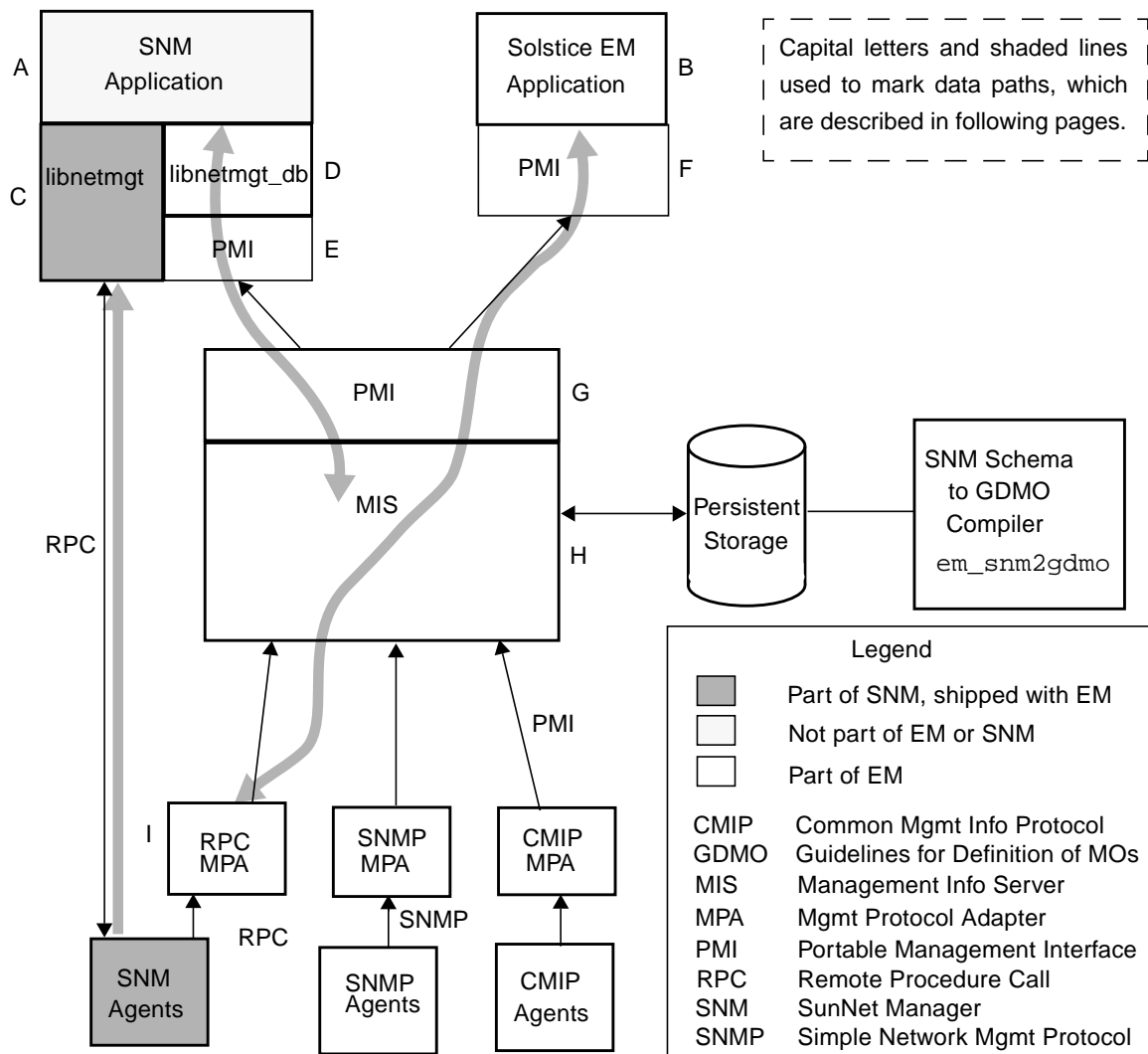


FIGURE 8-1 SNM-Solstice EM Compatibility

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

The areas of compatibility described in the bullets on page 8-2 are illustrated in FIGURE 8-1 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 8-1, 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.

8.2 Access to Solstice EM Features from SNM Applications

Solstice EM support for SNM applications accessing Solstice EM features is illustrated in the following figure.

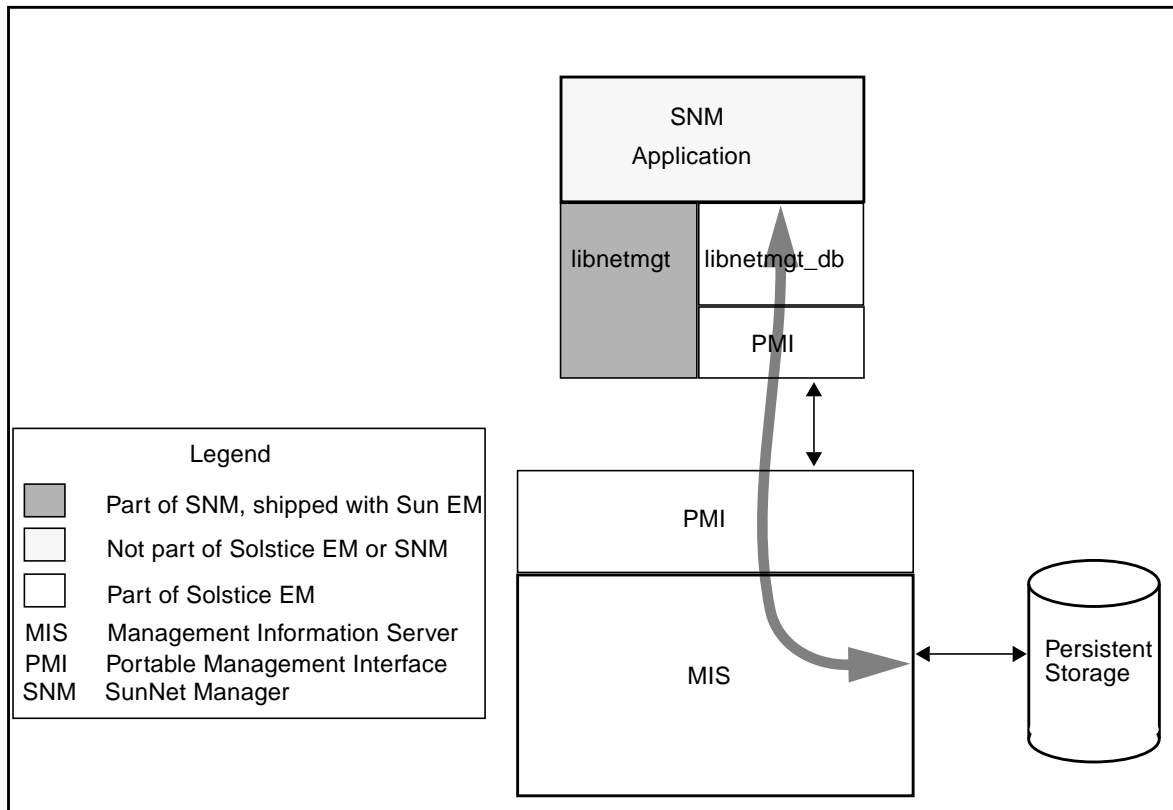


FIGURE 8-2 SNM Application Accessing Solstice EM Features

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 Applications 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.

8.3 Adding an SNM Application to Solstice EM

Note – This section describes the general procedure to be followed for setting up an SNM application to work with Solstice EM. However, specific third-party SNM applications may require additional steps or modifications to this procedure. Before carrying out the steps outlined below, see Section 8.4, “Information for Configuring Specific SNM Applications ” to determine what modifications or additions to this procedure are necessary for the particular SNM application you wish to use with Solstice EM. If the SNM application is the Solstice Cooperative Consoles Receiver, for additional information see Chapter 7.

▼ To Run Your SNM Application with Solstice EM

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 Chapter 6 of *Installation Guide*.

2. **Convert third-party SNM icons to Solstice EM glyph format.**

Element types shipped with Site/SunNet/Domain Manager have already been mapped to Solstice 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 Solstice EM features. These icons must be converted from SNM Xview format to Solstice EM X-pixmap (pm) format.

▼ To make this conversion

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 Solstice 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. Check their website (www.x.org) for information on obtaining `netpbm` and `convert` from their `ftp` site.

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 Solstice 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 8 in *Management Information Server (MIS) Guide*.

4. Use the `em_snm_type_import` utility (as root) to incorporate new SNM element types, defined in SNM schema files, into the Solstice EM environment.

You only need to do this step if you have custom or third-party SNM element types that you want to incorporate into Solstice EM. When Solstice EM is shipped to you, the default element types included with Site/SunNet/Domain Manager are already mapped to Solstice EM topology types. 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
```

This utility updates entries in `$EM_HOME/config/SNM2EM_type_mapping`, which maintains the mapping of SNM element types to Solstice 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:

- `$EM_HOME/install/em_platform/bc_map`
- `$EM_HOME/config/em_viewer.cf`

However, if a user has already run the Network Views, it is possible that a personalized copy of the Network Views configuration file has been created in their home directory. In that case, their Network Views configuration file will not have been updated with the new topology information generated by `em_snm_type_import`. To update their individual topology type information, a user can copy the master Network Views configuration file (`$EM_HOME/config/em_viewer.cf`) to `~/.em_viewer.cf`

When `em_snm_type_import` is run, 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/glyphs directory
Warning: Need syn-novell-server.pm file in $EM_HOME/glyphs directory
Warning: Need baystack-100-conc.pm file in $EM_HOME/glyphs directory
Warning: Need syn-fddi-segment.pm file in $EM_HOME/glyphs directory
```

5. Run `em_services -reload` 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 -reload` 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 Network View 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_db` 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. You will also want to add this command to your `.cshrc` file.

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 Solstice EM Launcher.

Invoke the Configure Applications window in the Network Tools to add the SNM application to the launcher. For information on adding applications to the Solstice EM Network Tools, refer to Chapter 2 in *Managing Your Network*. If you select “Yes” in the Solstice EM Application field in the Configure Applications window, the icon will be grayed out whenever the MIS is disconnected.

8.3.1 Forwarding Event and Topology Information from SunNet Manager to Solstice EM

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

If you are already using Site/SunNet/Domain 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 the following figure. For a more detailed discussion of Cooperative Consoles, see Chapter 7.

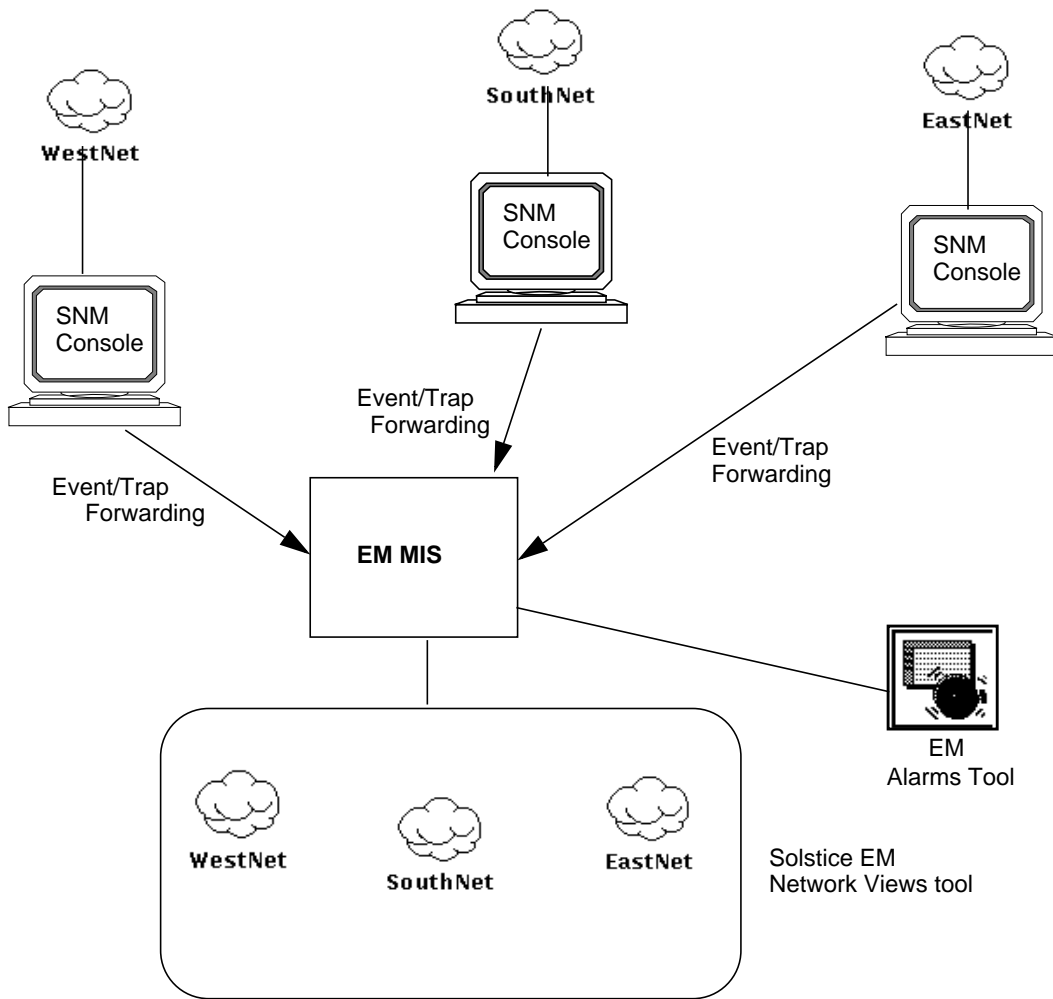


FIGURE 8-3 Forwarding of Information to Central Management Station

8.3.2 SunNet Manager Application Support

Solstice EM's ability to interoperate with Cooperative Consoles is an illustration of Solstice EM's support for applications that have been developed for use with Site/SunNet/Domain Manager. There are numerous third-party applications developed for SNM that can also be used with Solstice EM. For more information, see Chapter 8.

8.4 Information for Configuring Specific SNM Applications

This section provides information needed to set up specific SNM applications to work with Solstice EM. The specific SNM applications are discussed in alphabetical order.

8.4.1 Running Solstice EM and Applications on Hosts With a New IP Address or Name

Applications may be installed where the hostname or IP address has been changed. This section provides procedures for getting Solstice EM and any third party applications you plan to use with Solstice EM working properly.

▼ To Set up Solstice EM to be Used on a Host With a New Name

1. **Change <oldname>_emdb to <newname>_emdb in the directory:** /opt/SUNWemrdb/etc/onconfig.em
2. **Change all occurrences of <oldname> to <newname> in the directory:** /opt/SUNWemrdb/etc/sqlhosts
3. **Change all occurrences of <oldname> to <newname> in the directory:** /opt/SUNWconn/em/build/acct/onconfig.em
4. **Replace the old name with the new name in LM_LICENSE_FILE path of the following license files:**
/etc/opt/SUNWconn/em/conf/rpc_mpa_config
/etc/opt/SUNWconn/em/conf/snmp_mpa_config
/etc/opt/SUNWconn/em/conf/trap_forward

Note – To find other files that use the <oldname>, execute the grep command at the prompt (oldname is the old name used for the host): `grep <oldname> *`

5. **Type `em_services -reload` at the prompt to run Solstice EM.**

▼ To Set up CiscoWorks to be Used on a Host With a New IP Address or Name

1. **Replace the hexadecimal representation of the old IP address with the hexadecimal representation of the new IP address in the file `/opt/CSC0syb/interfaces`.**

The IP address is embedded in a string that begins with `\x000208ae...` You must add skip the first eight characters after the `'\x'`, then add the new hexadecimal address after the `'\x'`.

For example, the hexadecimal form of the IP address 129.146.183.19 is 8192b713. Locate the old hexadecimal string and replace it with the new one.

Note – You can calculate hexadecimal representations using `/usr/openwin/bin/calctool`.

2. **Replace the old name with the new name in the file `/opt/CSC0syb/interfaces`.**
3. **Replace the hexadecimal representation of the old IP address with the a hexadecimal representation of the new IP address in the file `/opt/CSC0syb/interfaces.001` if it exists. (See Step 1 for more information)**
4. **Replace the old name with the new name in the file `/opt/CSC0syb/interfaces.001` if it exists.**

▼ To Set up Remedy ARS to be Used on a Host With a New IP Address or Name

- **Replace all occurrences of the old name with the new name in the following files:**

```
/etc/ar  
/arHome/arcmds/emxdemo.arq  
/arHome/arcmds/emxsearc.arq  
/arHome/arcmds/emupdat.arq
```

▼ To Set up Landmark Performance Works to be Used on a Host With a New IP Address or Name

1. **Replace all occurrences of the old name with the new name in the following files:**

```
/usr/landmark/landmark/license.dat  
/arHome/landmark/data/pcws*/device.ini
```

2. Replace the old IP address in the file `/usr/landmark/data/pwcs*/ns.ini`.
3. Replace all occurrences of the old name with the new name in the file `/usr/landmark/data/pwcs*/trapgen/PWcore*.ini`.

▼ To Complete Application Set up on Hosts With New IP Addresses or Names

- After completing the steps to set up Solstice EM, CiscoWorks, Remedy ARS, and Landmark Performance applications, reboot the machine.

8.4.2 Configuring Remedy's Action Request System (ARS) to Work with Solstice EM

The instructions in this section cover any third-party SunNet Manager application that alters the default SNM `elements.schema` file during installation. In particular, this applies to Remedy's Action Request System (ARS). The default SNM `elements.schema` file has already been converted to GDMO and ASN.1 files for loading into the Solstice EM MIS as shipped to you. Also, Solstice EM is shipped to you with mappings of SNM element types to Solstice EM topology types. However, if the installation of a third-party SNM application alters the `elements.schema`, then you will need to do the following:

1. After installing the third-party SNM application, run `em_snm_type_import` on the modified `elements.schema` file.

Use the following command:

```
# $EM_HOME/bin/em_snm_type_import -file elements.schema
```

For more information on the `em_snm_type_import` utility, refer to Section 8.3, "Adding an SNM Application to Solstice EM."

2. Run the `em_snm2gdmo` compiler on the `elements.schema` file to convert it to GDMO documents.

```
# $EM_HOME/bin/em_snm2gdmo /opt/SUNWconn/snm/struct/  
elements.schema 1
```

Refer to the *Management Information Server (MIS) Guide* for more detailed information.

3. Remove the extra menu entries from the Network Views configuration file.

The `elements.schema` shipped with Remedy ARS has various menu entries that do not apply to Solstice EM. Edit the Network Views configuration file (`$EM_HOME/config/em_viewer.cf`) to remove these extra entries. The extra entries are the following for the Network Views menu:

- Browser
- Graphs
- Snapshot
- Network Discovery
- IPX Discover

An extra entry for Network Discovery is also created in the element icon menu.

4. Copy the updated global Solstice EM Network Views configuration file to your home directory.

The `em_snm_type_import` utility updates the global Solstice EM Network Views configuration file. To copy the changes to your home directory, enter:

```
%cp $EM_HOME/config/em_viewer.cf ~/.em_viewer.cf
```

5. Exit and restart the Network Views.

The ARS commands should be incorporated into the Network Views menu.

8.4.3 Configuring Konfig 2.4 to Work with Solstice EM

▼ To Use Konfig 2.4 With Solstice EM

1. As root, install Solstice EM.

Refer to the *Installation Guide* for instructions.

2. Each user needs to set the SNMHOME environment variable.

For example:

```
% setenv SNMHOME /opt/SUNWconn/snm
```

Also add the same line to the user's `.cshrc` file:

```
setenv SNMHOME /opt/SUNWconn/snm
```

3. Create a link from \$SNMHOME/bin to \$EM_HOME/bin.

Remove the `$SNMHOME/bin` directory and create a link pointing `$SNMHOME/bin` to `$EM_HOME/bin`:

```
#rm -r $SNMHOME/bin
#ln -s $EM_HOME/bin $SNMHOME/bin
```

4. Add \$SNMHOME/bin to the user's path.

For example:

```
% setenv PATH $SNMHOME/bin:${PATH}
```

Also, add the same line to the user's `.cshrc` file; for example:

```
setenv PATH $SNMHOME/bin:${PATH}
```

5. Convert the config SNM schemas to GDMO and ASN.1 files.

Refer to Step 3 under Section 8.3, "Adding an SNM Application to Solstice EM."

6. Convert the config types and menus into Solstice EM topology types.

Use the `em_snm_type_import` utility to convert the types and menus in the `konfig_snm.schema` file following the instructions in Step 4 and Step 5 under Section 8.3, "Adding an SNM Application to Solstice EM."

7. For objects added using Solstice EM Network Discovery, use Network Views-Object Properties window to set the usrType field.

Go to each object icon and choose Object Properties from the icon menu to invoke the Network Views-Object Properties window. Fill in the attribute corresponding to `usrType[1]` with the SNM component type name—for example, `component.router`

Note – It is not necessary to carry out Step 7 for objects imported to Solstice EM from an SNM database using `em_snmdb_import` or added to the Solstice EM database by running `snm_discover`.

8.4.4 Configuring Optivity 7.0 to Work with Solstice EM

▼ To Use Optivity 7.0 With Solstice EM as Root

1. Install Solstice EM.

Refer to the *Installation Guide* for installation instructions.

2. Create a link for the `snm.conf` file.

Optivity expects to find the `snm.conf` file. Create the appropriate link as follows:

```
# ln -s /etc/opt/SUNWconn/snm/snm.conf /opt/SUNWconn/snm/snm.conf
```

3. Create an `snm_version` file that contains the following line:

```
echo "/opt/SUNWconn/snm/bin/snm:Release 2.3 FCS - Solaris X86 Patch  
Level 0"
```

This file needs to be located in `/opt/SUNWconn/em/bin` and needs to have executable file permissions.

4. Each user needs to set the `SNMHOME` environment variable.

For example:

```
% setenv SNMHOME /opt/SUNWconn/snm
```

Also add the same line to the user's `.cshrc` file:

```
setenv SNMHOME /opt/SUNWconn/snm
```


5. Move executables from \$SNMHOME/bin directory to \$EM_HOME/bin.

For example:

```
#mv $SNMHOME/bin/* $EM_HOME/bin
```

6. Create a link from \$SNMHOME/bin to \$EM_HOME/bin.

Remove the \$SNMHOME/bin directory and create a link pointing \$SNMHOME/bin to \$EM_HOME/bin:

```
#rmdir $SNMHOME/bin  
#ln -s $EM_HOME/bin $SNMHOME/bin
```

7. Each user needs to set the SNMDBDIR environment variable.

For example:

```
% setenv SNMDBDIR /var/opt/SUNWconn/snm
```

Also, add the same line to the user's .cshrc file; for example:

```
setenv SNMDBDIR /var/opt/SUNWconn/snm
```

8. Add \$SNMHOME/bin to the user's path.

For example:

```
% setenv PATH $SNMHOME/bin:${PATH}
```

Also, add the same line to the user's .cshrc file; for example:

```
setenv PATH $SNMHOME/bin:${PATH}
```

9. Install Optivity.

- Follow the instructions steps documented in *Getting Started with Optivity LAN 7.0 for UNIX* by Bay Networks.
- After installation, run `/opt/lnms/bin/LNMS_ENABLE` as instructed by the Optivity documentation but do not do any of the other post-installation steps described in the Optivity manual. In particular, do *not* invoke `/opt/lnms/optivity_snm`, as instructed by the Optivity manual.

- Source your `.cshrc` file. For example:

```
% source ~/.cshrc
```

10. Create an SNM database directory and files.

Optivity looks for certain SNM database files when it starts up. Create these directories and files if they do not already exist.

- a. Create the directory `$SNMDBDIR/db.<user-id>`.

- b. Create the required database files using the following commands:

```
% touch $SNMDBDIR/db.<user-id>/nc.rec
% touch $SNMDBDIR/db.<user-id>/snm+lock
```

where `<user-id>` is the user's login name.

11. Follow the steps outlined above in Section 8.3, "Adding an SNM Application to Solstice EM," modified as follows:

- a. Copy the Solstice EM Optivity icons (the `.pm` files) into `$EM_HOME/glyphs`.

The Optivity icons have already been converted into Solstice EM icons for you. They are installed in `$EM_HOME/glyphs/optivity`. Substitute the following step for Step 2 in Section 8.3, "Adding an SNM Application to Solstice EM".

Copy the `.pm` files from `$EM_HOME/glyphs/optivity` to `$EM_HOME/glyphs`.

- b. Edit the `$LNMSHOME/snm/struct/synoptics-menus.schema` file to correct formatting errors.

Before running the schema-to-GDMO compiler or `em_snm_type_import` utility on the `synoptics-menus.schema` file, it is necessary to correct certain formatting errors in that file. The following four application entries in the schema file are incorrect in that they have component entries broken into two lines by a carriage return:

```
(component.2810conc "Box Status" "boxstatus -H %Name -I %IP_Address
-R %SNMP_RdCommunity")
(component.2810conc "Nmm Status" "nmmstatus -R %Name -I %IP_Address
-R %SNMP_RdCommunity")
(component.2810conc "Activity" "boxactivity -H %Name -I %IP_Address
-R %SNMP_RdCommunity")
(component.2810conc "Diagnostics" "boxdiags -H %Name -I %IP_Address
-R %SNMP_RdCommunity")
```

You will need to remove the carriage return so that each entry is a single line. The following is an example of a correct entry:

```
(component.2810conc "Box Profile" "boxprofile -H %Name -I
%IP_Address -R %SNMP_RdCommunity")
```

c. Convert the Optivity SNM schema files to GDMO documents.

Optivity ships two directories that contain SNM schema files that need to be converted into GDMO and ASN.1 files. The directory `$LNMSHOME/snm/struct-base` contains three schemas:

- `switch-elements.schema`
- `synoptics-elements.schema`
- `synoptics-stackprobes.schema`

The schema file, `synoptics-lcell.schema` in `$LNMSHOME/snm/struct` directory needs to be converted to GDMO.

Note – There are a number of schemas in the `$LNMSHOME/snm/struct` directory but only the `synoptics-lcell.schema` file needs to be converted to GDMO. The record definitions in the other schema files in this directory are already known to Solstice EM and do not need to be converted to GDMO.

Use the schema-to-GDMO compiler shipped with Solstice EM to create ASN.1 and GDMO files from these schemas. The syntax for using the compiler is:

```
#em_snm2gdmo <schema-file> <oid>
```

where *<schema-file>* is the name of the SNM schema file to be converted and *<oid>* is a number that is unique within the Solstice EM MIS for each schema compiled. For example:

```
#$EM_HOME/bin/em_snm2gdmo /opt/lnms/snm/struct-base/synoptics-
elements.schema 200
```

For more information, refer to Chapter 8 in *Management Information Server (MIS) Guide*.

d. Move the GDMO and ASN.1 files.

For example:

```
# mv synoptics-elementsschema.gdmo /opt/SUNWconn/em/etc/gdmo/.
# mv synoptics-elementsschema.asn1 /opt/SUNWconn/em/etc/asn1/.
```

e. Import the Optivity types into the Solstice EM MIS.

Each schema file within `$LNMSHOME/snm/struct-base` and `$LNMSHOME/snm/struct` needs to have its SNM element types and menus converted into Solstice EM topology types. The `em_snm_type_import` utility is used for this purpose; this utility updates all the relevant Solstice EM configuration files. The syntax for this utility is:

```
# em_snm_type_import -file <schema-file>
```

For example:

```
#em_snm_type_import -file /opt/lnms/snm/struct/baynet-rtop.schema
```

`em_snm_type_import` updates the global Network Views configuration file (`em_viewer.cf`) located in `$EM_HOME/config`. This utility also creates a file `bc_map` in `$EM_HOME/install/em_platform` to create the appropriate topology type mappings for the Optivity devices.

Note – You must run `em_snm_type_import` on the `$LNMSHOME/snm/struct-base` schema files before importing the types from the `$LNMSHOME/snm/struct` schemas.

f. Add `$EM_HOME/lib` to your `LD_LIBRARY_PATH`.

For example:

```
% setenv LD_LIBRARY_PATH $EM_HOME/lib:${LD_LIBRARY_PATH}
```

Add the same line to your `.cshrc` file:

```
setenv LD_LIBRARY_PATH $EM_HOME/lib:${LD_LIBRARY_PATH }
```

If you have also installed SNM, make sure that `$EM_HOME/lib` occurs before `$SNMHOME/lib` in your `LD_LIBRARY_PATH`.

- g. Create a `$SNMHOME/bin/snm` shell script, with execute permissions, containing the following:**

```
#!/bin/sh
/opt/SUNWconn/em/bin/em_viewer
```

- h. Modify the `/opt/lnms/bin/trap_server.sh` file as follows:**

- Replace occurrences of `$SNMHOME/agents/na.snmp-trap` with `$SNMHOME/bin/em_snmp-trap`.
- Replace occurrences of `/opt/SUNWconn/snm/agents/na.snmp-trap` with `/opt/SUNWconn/snm/bin/em_snmp-trap`.
- Replace all other occurrences of `na.snmp-trap` with `em_snmp-trap`.

- i. Reboot the system.**

- j. Run `em_services -r` to compile and load the new GDMO and ASN.1 documents.**

- k. Start `em_snmp-trap` on port 412.**

For example:

```
#$EM_HOME/bin/em_snmp-trap -p 412 &
```

- l. Start Optivity with the following command:**

```
% $LNMSHOME/bin/optivity_snm
```

- m. Optivity types can now be created either by using Network Views-Object Properties window to change a device to an Optivity type, or by using the Solstice EM Network Views menu to create a new object with an Optivity type.**

Filling in the Object Description fields for a device in Network Views-Object Properties window is similar to entering object descriptions in the SNM properties sheet for the new element, as described in Chapter 3 of the Optivity manual, on Post-Installation Tasks. However, there is one field in the Network

Views-Object Properties window Object Description window that is not present in the SNM Properties sheet. You must fill in the `usrType` field, in Network Views-Object Properties window, with the SunNet Manager type name.

For example, the Optivity type that has the Solstice EM type name `syn-internet` has the following SNM type name:

```
view.syn-internet
```

Note – When creating objects using Network Views-Object Properties window, Optivity device types rely on the Object Descriptions that you fill in.

You should now be able to invoke Optivity tools and commands from the Solstice EM Network Views menu or from an icon popup menu just as in SunNet Manager.

Note – To populate the database by running discovery from the Optivity tools, it is necessary to run the Optivity tools as root.

8.4.5 Configuring Landmark's Performance Works to Work with Solstice EM

This section describes procedures for setting up Performance Works for UNIX 4.0, Performance Works for Sybase 2.0, and Performance Works Monitor 1.3.

Note – The default SNMP daemon supplied with Solstice EM (`snmp`) must be replaced with Landmark's SNMP agent (`xsnmpd`). `xsnmpd` is placed in the `/etc` directory along with `snmpd.conf`, `snmpd.defs`, and `snmpd.peers`. These four files need to be downloaded from Landmark's anonymous ftp site.

▼ To Set Up Performance Works for UNIX 4.0

1. Each user needs to set the SNMHOME environment variable.

For example:

```
% setenv SNMHOME /opt/SUNWconn/snm
```

Also add the same line to the user's .cshrc file:

```
setenv SNMHOME /opt/SUNWconn/snm
```

2. Move executables from \$SNMHOME/bin directory to \$EM_HOME/bin.

For example:

```
#mv $SNMHOME/bin/* $EM_HOME/bin
```

3. Create a link from \$SNMHOME/bin to \$EM_HOME/bin.

Remove the \$SNMHOME/bin directory and create a link pointing \$SNMHOME/bin to \$EM_HOME/bin:

```
#rmdir $SNMHOME/bin  
#ln -s $EM_HOME/bin $SNMHOME/bin
```

4. Add \$SNMHOME/bin to the user's path.

For example:

```
% setenv PATH $SNMHOME/bin:${PATH}
```

Also, add the same line to the user's .cshrc file; for example:

```
setenv PATH $SNMHOME/bin:${PATH}
```

5. Each user needs to set the SNMDBDIR environment variable.

For example:

```
% setenv SNMDBDIR /var/opt/SUNWconn/snm
```

Also, add the same line to the user's `.cshrc` file; for example:

```
setenv SNMDBDIR /var/opt/SUNWconn/snm
```

6. Install the Landmark products—Performance Works, UNIX agent, and SNMP polling agent.

7. Use the MIB-to-GDMO compiler shipped with Solstice EM to create ASN.1 and GDMO files from the Landmark MIBs.

For example:

```
#em_cmib2gdmo pwunix.mib
```

Refer to Chapter 8 in *Management Information Server (MIS) Guide* for information on running the MIB to GDMO compiler.

8. Move the GDMO and ASN.1 files.

The GDMO and ASN.1 files created by the compiler must be moved to the appropriate directory under `$EM_HOME/etc`. For example:

```
#mv pw*.gdmo /opt/SUNWconn/em/etc/gdmo
#mv pw*.asn1 /opt/SUNWconn/em/etc/asn1
```

9. Convert the Landmark SNM element types and menus to Solstice EM topology types.

The schema file `lsc_snmintegration.schema` needs to have its SNM element types and menus converted into Solstice EM topology types. Follow the instructions in Step 4 and Step 5 under Section 8.3, “Adding an SNM Application to Solstice EM.” After running the SNM type import utility, you will need to copy the updated global Solstice EM Network Views configuration file to Landmark’s home direction. For example:

```
%cp $EM_HOME/config/em_viewer.cf ~landmark/.em_viewer.cf
```

10. Run Solstice EM Network Discovery to populate the MIS.

You can display the values of attributes for objects configured with Landmark agents by selecting the object in the Network Views and then invoking the SNMP Data from the object icon menu. The `pwadmin` utility can be used to set the thresholds for traps generated by the Landmark agents. The Alarms can be used to view trap notifications generated by the agents.

8.5 Importing an SNM Database into Solstice 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 predefined event requests or event request templates in the SNM database are not loaded into the Solstice EM MIS. This means that SNM event request templates that have been defined for use in link or router management, for example, are not imported into the Solstice EM MIS.

8.6 Access to SNM Agents by SNM Applications

Solstice EM support for SNM applications to access SNM agents over Solstice EM is illustrated in the following figure.

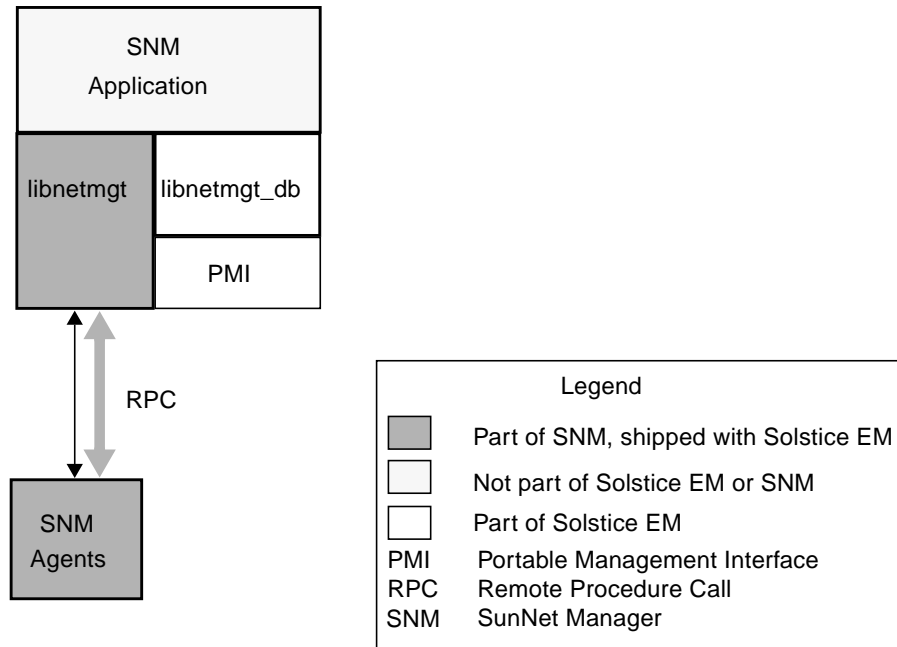


FIGURE 8-4 SNM Application Accessing SNM Agents Over Solstice EM

Under Solstice EM, an SNM application accesses an SNM agent just as it would under SNM, through the `libnetmgt` 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 as follows:

```
/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 Solstice 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 8.3, “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 8.3, “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 8.3, “Adding an SNM Application to Solstice EM .”

8.7 Access to SNM Agents by Solstice EM Applications

Solstice EM support for Solstice EM applications accessing SNM agents is illustrated in the following figure.

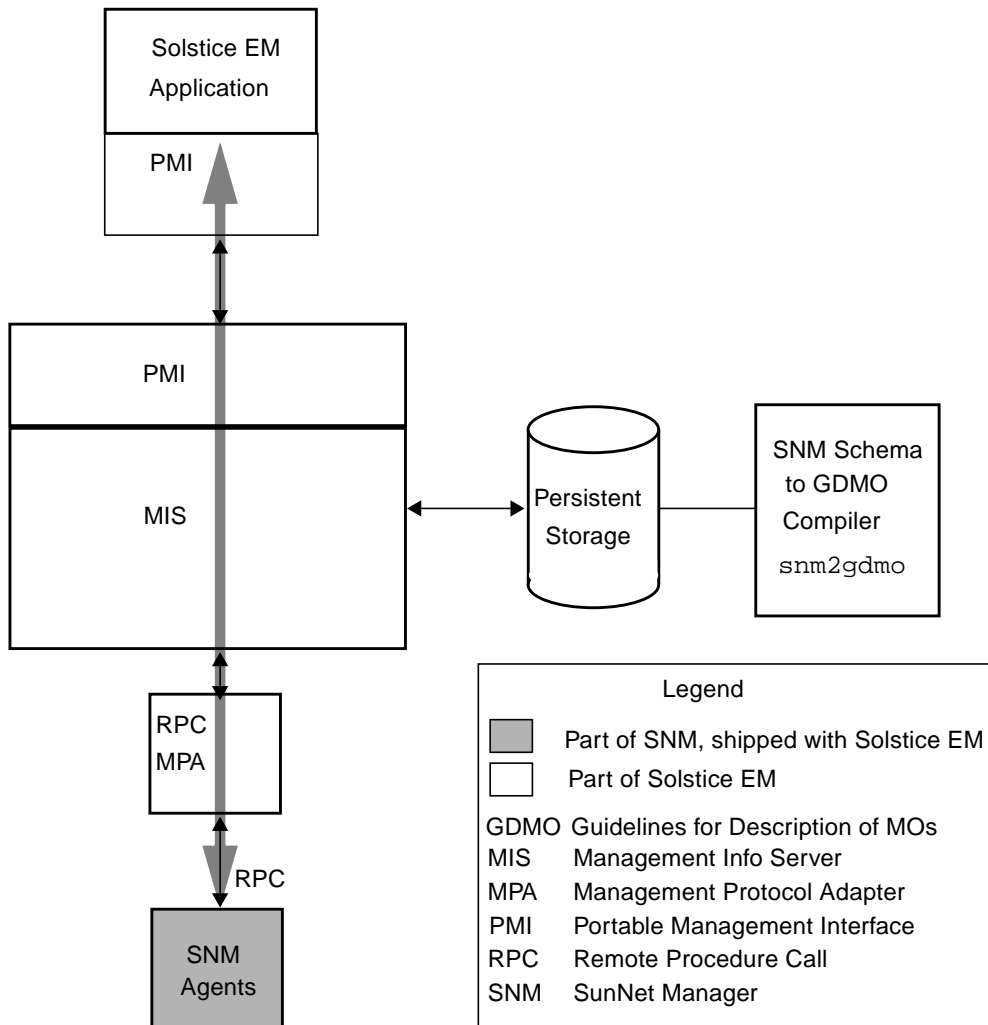


FIGURE 8-5 Solstice EM Applications Accessing SNM Agents

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 RPC/CMIP Data window, Network Views-Object Properties, Event Logs window, and Alarms window 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 Event Logs and the Alarms, and Nerve Center request capability, for manipulating data in ways not possible in SNM.

8.7.1 Configuration

Solstice EM's application support for SNM agents is seamless. 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`.

8.7.2 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 vendor. The product has an `snm2gdm` 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 8 of *Management Information Server (MIS) Guide*.

8.7.3 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, communicating to MIS with the SNM RPC protocol and communicate 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 Solstice EM, refer to Chapter 17 in *Developing C++ Applications*.

SNMP Management

Simple Network Management Protocol (SNMP) is a protocol for exchanging information between network managers and agents processes within various managed objects that are able to report their status on request. It is a connection-less protocol, with the view of continuing to receive information from managed objects even when network performance is degraded and a connection-based reliable transport may fail.

This chapter describes the following topics:

- Section 9.1 “SNMP Managed Components” on page 9-1
- Section 9.2 “SNMP Management Protocol Adapter” on page 9-3
- Section 9.3 “SNMP MPA Configuration” on page 9-3
- Section 9.4 “Specifying the Version of SNMP Used” on page 9-4

9.1 SNMP Managed Components

Solstice Enterprise Manager (Solstice EM) provides the following components for management of network resources that are manageable using SNMPv1 and SNMPv2c:

- An SNMP daemon (`snmpd`) which can be installed on Sun workstations.

For information about the SNMP daemon refer to the *Site/SunNet/Domain Manager Reference Manual*. Procedures for installing the SNMP daemon is available in Chapter 6 of *Installation Guide*.

- Distributable SNMP proxy agents (`na.snmp` and `na.snmpv2`) that use Remote Procedure Call (RPC) protocol to handle communication (such as threshold-checking) between the Solstice EM management station (where the MIS resides) and SNMP agents.

The RPC-based SNMP proxy agents are described in Chapter 10.

- A distributable SNMP trap daemon (`em_snmp-trap`) which listens for SNMP event notifications (traps) and translates these into CMIP event notifications for forwarding to the Solstice EM MIS. The trap daemon can also be configured to forward unprocessed SNMP traps to other managers.

The SNMP trap daemon is discussed in Chapter 11.

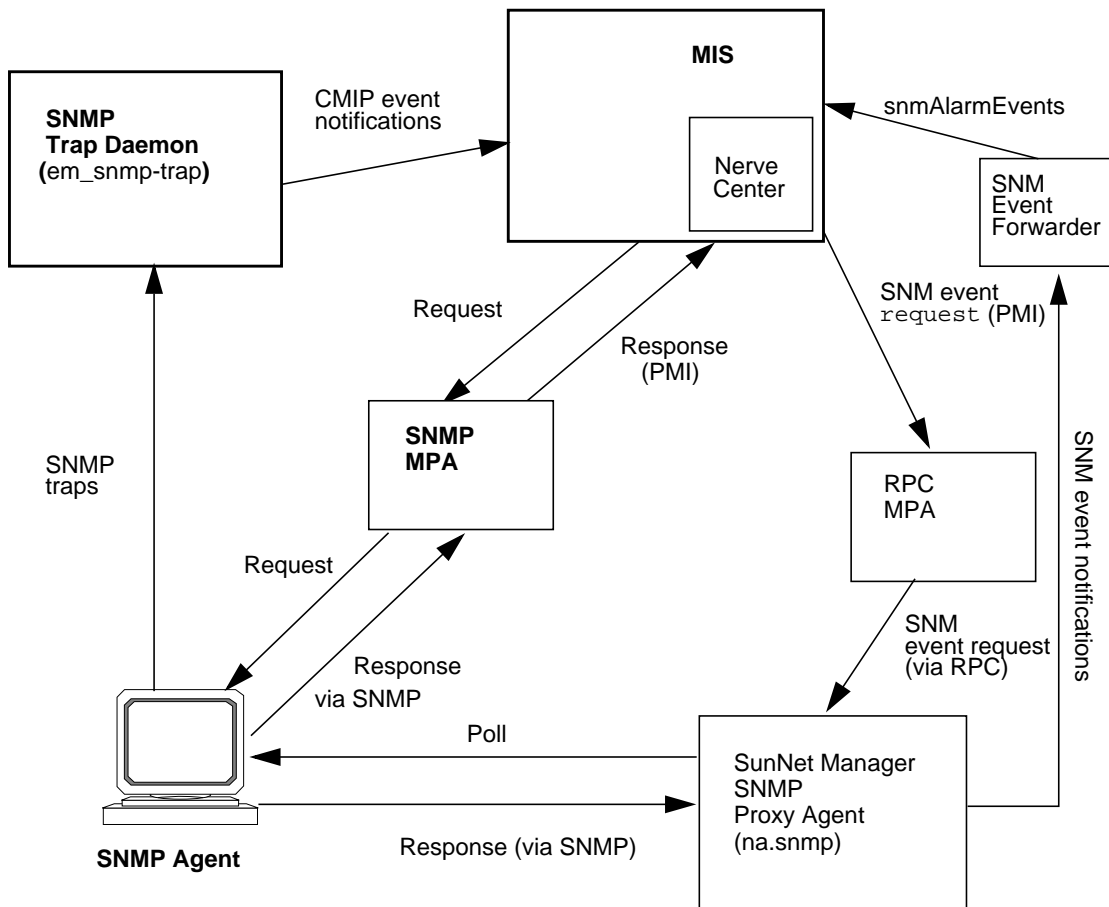


FIGURE 9-1 Components of Solstice EM's SNMP Management Support

- A distributable SNMP Management Protocol Adaptor (MPA), which handles communication of management requests and agent responses between the Solstice EM MIS and SNMP agents.

9.2 SNMP Management Protocol Adapter

The Simple Network Management Protocol (SNMP) Management Protocol Adapter (MPA) allows for the retrieval of data and the setting of attribute values for SNMP managed devices. The SNMP MPA works as a proxy agent between the Solstice EM MIS and any device on the network which is SNMP-manageable. The proxy agent allows you to manage any number of Management Information Bases (MIBs), where either standard SNMP MIB objects or enterprise-specific objects can be defined. SNMP MIBs get compiled by the Concise MIB compiler (`em_cmib2gdmo`) into GDMO and ASN.1 descriptions which then get loaded into the Solstice EM MIS.

The SNMP MPA processes Common Management Information Service (CMIS)-like requests received from the MIS, translates them into SNMP requests, and sends the requests out to the intended device. The translation is based on the OMNIPoint I Internet-ISO Management Coexistence (IIMC) standard. The SNMP MPA then receives the response from the SNMP device and forwards it back to the MIS.

9.3 SNMP MPA Configuration

The following configuration parameters for the SNMP MPA are set during installation:

- Default MIS host—The name of the machine running the MIS that the MPA is to connect to.
- Default MIS port—The port used in communicating with the MIS (by default this is port 5555).
- Default SNMP MPA port—The port on which the SNMP MPA listens for incoming messages (by default this is port 5575).
- SNMP request timeout—The length of time the SNMP MPA waits for a response to a request sent to an SNMP agent (by default, this is 20 seconds).
- SNMP retries—The number of times the SNMP MPA retries a request to an SNMP agent if there is no response (by default this is three).

The request and retry parameters determine when the SNMP MPA determines that an SNMP agent is unavailable.

9.4 Specifying the Version of SNMP Used

Solstice EM supports the SNMPv1 and SNMPv2c protocols. When you are performing management operations on objects in an SNMP agent, you have to specify the version of SNMP used for communications between the SNMP agent and your application.

Specify the version of SNMP used in either of the following ways:

- Calling the `set_management_protocol` function of the `EMSnmpAgent` class
- Using the PMI to set the `managementProtocol` attribute

9.4.1 Calling the `set_management_protocol` Function of the `EMSnmpAgent` Class

Call the `set_management_protocol` function if you are using the `EMSnmpAgent` class to represent your SNMP agent. In the call to the `set_management_protocol` function, set the argument as follows depending on the version of SNMP:

- SNMPv1: `EMSnmpAgent::snmp_v1`
- SNMPv2c: `EMSnmpAgent::snmp_v2`

A call to the `set_management_protocol` function is shown in CODE EXAMPLE 9-1.

CODE EXAMPLE 9-1 Calling the `set_management_protocol` Function

```
...  
EMSnmpAgent::set_management_protocol (EMSnmpAgent::snmp_v2);  
...
```

In this example, SNMPv2c is used for communications between the SNMP agent and the application.

9.4.2 Using the PMI to Set the managementProtocol Attribute

If you are using the `Image` class to represent your SNMP agent, set the `managementProtocol` attribute when you initialize the instance of `Image`. To set the `managementProtocol` attribute, call the `set_str` function. You must set the `managementProtocol` attribute to the object identifier (OID) of the version of SNMP that you are using.

The versions of SNMP are identified by the following OIDs:

- SNMPv1: 1.3.6.1.4.1.42.2.2.2.9.2.4.3.1
- SNMPv2c: 1.3.6.1.4.1.42.2.2.2.9.2.4.3.2

CODE EXAMPLE 9-2 shows code for setting the `managementProtocol` attribute.

CODE EXAMPLE 9-2 Setting the managementProtocol Attribute Directly

```
...  
Image im;  
...  
im.set_str ("managementProtocol", "{1 3 6 1 4 1 42 2 2 2 9 2 4 3 1}");  
...
```

In this example, SNMPv1 is used for communications between the SNMP agent and the application.

SunNet Manager SNMP Proxy Agents

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 Chapter 6 in *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 Enterprise Manager* (Solstice EM) with versions of SNM prior to 2.2.

This chapter describes the following topics:

- Section 10.1 “Proxy Agents” on page 10-1
- Section 10.2 “SNMP Proxy Agent Operation” on page 10-4
- Section 10.3 “SNMP Trap Daemon (em_snmp-trap) Operation” on page 10-7
- Section 10.4 “Schema Files” on page 10-7
- Section 10.5 “SNMP Version 2 Support” on page 10-9

10.1 Proxy Agents

The SunNet Manager (SNM) 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.

SunNet Manager requests can be launched from the Solstice EM MIS using the request-handling capabilities of the Solstice EM Nerve Center (as described in Chapter 17). 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 10.5, “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 in 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 RPC-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

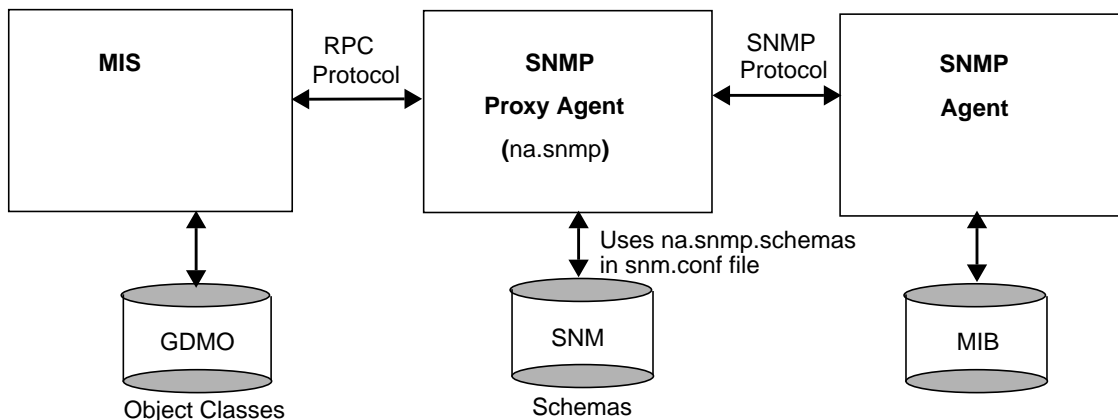


FIGURE 10-1 MIB, GDMO, and Schema Definitions

Generating GDMO object classes from SNMP MIBs is described in Chapter 8 of *Management Information Server (MIS) Guide*. 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, as shown in the above figure.

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, 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 `snm.conf` entry in the *Site/SunNet/Domain Manager Reference Manual* for information on the keywords that are related to the SNMP proxy agent.

10.2 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 the following figure.

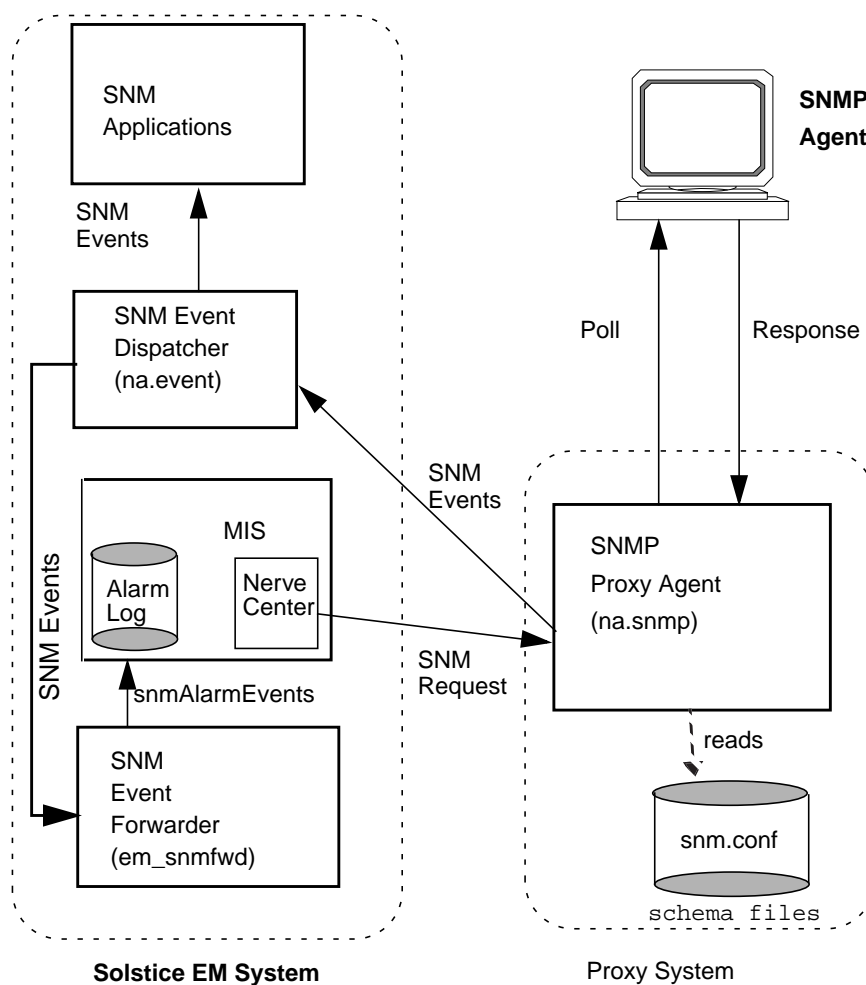


FIGURE 10-2 SNMP Proxy Agent Operation

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 schema 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.

10.3 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 Solstice EM SNMP trap daemon is described in Chapter 11.

10.4 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 10.5.3, “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 as follows:

- 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 1.3.6.1.2.1.2.1.6 -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.

10.5 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 *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 *Solstice 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.

10.5.1 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

10.5.1.1 Structure of Management Information

The SMI for SNMPv2 is based on the SMI for SNMP. The SNMPv2s 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 includes a `MAX-ACCESS` clause which allows you to specify the maximum level of access.

10.5.1.2 Protocol Operations

SNMPv2 has three new protocol data units (PDU). The SNMPv2 trap PDU works in a way similar to that of the SNMPv1 trap PDU, but it uses a different format from the SNMPv1 trap PDU. Its format is changed to be the same 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. This PDU is presently not used by Solstice EM for its operation.

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 Protocol Data Unit (PDU) provides management information to an application using the second SNMPv2 manager.

10.5.2 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 as follows:

- `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.

10.5.3 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 `v2mib2schema(5)` man page (or `v2mib2schema` entry in the *Site/SunNet/Domain Manager Reference Manual*) for more details.

Mapping SNMP Traps to CMIP Event Notifications

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 Enterprise Manager* (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.

Solstice EM supports both SNMPv1 and SNMPv2c. For more information, see Section 11.1 “SNMP Support” on page 11-1.

This chapter describes the following topics:

- Section 11.1 “SNMP Support” on page 11-1
- Section 11.2 “Trap Daemon Operation” on page 11-2
- Section 11.3 “The Structure of SNMP Traps” on page 11-5
- Section 11.4 “Default Trap Mapping” on page 11-8
- Section 11.5 “Trap Daemon Behavior When No Mapping is Provided” on page 11-13
- Section 11.6 “Format of trap_maps File” on page 11-14
- Section 11.7 “Customizing the Mapping of SNMP Traps” on page 11-25
- Section 11.8 “Distributed Trap Handling” on page 11-37

11.1 SNMP Support

Solstice EM supports both SNMP version 1 (SNMPv1) and SNMP version 2c (SNMPv2c). The SNMPv2c standard is based on the community string model. Solstice EM does not support SNMPv2 `InformRequest` (confirmed traps).

Supporting both SNMPv1 and SNMPv2c protocol offers you the flexibility to interact with SNMPv1 and SNMPv2c agents on your network. Configuration and operation of the SNMP versions differ in some areas, and they are explained in detail in the remaining sections of this chapter.

11.2 Trap Daemon Operation

The Solstice EM trap daemon (`em_snmp-trap`) supports both SNMPv1 and SNMPv2c.

The trap daemon for SNMPv1 and SNMPv2c does the following with incoming SNMP traps:

- SNMP traps are converted to Common Management Information Protocol (CMIP) event notifications and sent to the MIS. Like other Solstice EM applications, `em_snmp-trap` uses the Portable Management Interface (PMI) 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`). Procedures for customizing the trap daemon are described in Section 11.7, "Customizing the Mapping of SNMP Traps".
- SNMP traps are also translated into SunNet Manager traps for use by SunNet Manager (SNM) 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 filter raw SNMP traps to be forwarded to other managers or discarded. Configuring this capability is described in Section 11.8.1, "Filtering SNMP Traps for Other Managers." The SNMP manager(s) is configured when you install the Solstice EM trap daemon. The installation script prompts you for the host name and port number of the manager that is to receive the forwarded SNMP traps. If you want to discard raw SNMP traps for other managers, add DISCARD to the trap-filtering-record in the `trap_forward` file.

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 the following figure.

For information on how to install `em_snmp-trap`, refer to Chapter 6 in *Installation Guide*.

Note – In the current release, `em_snmp-trap` is supported only on Sun workstations running Solaris 2.6 through 2.8 software.

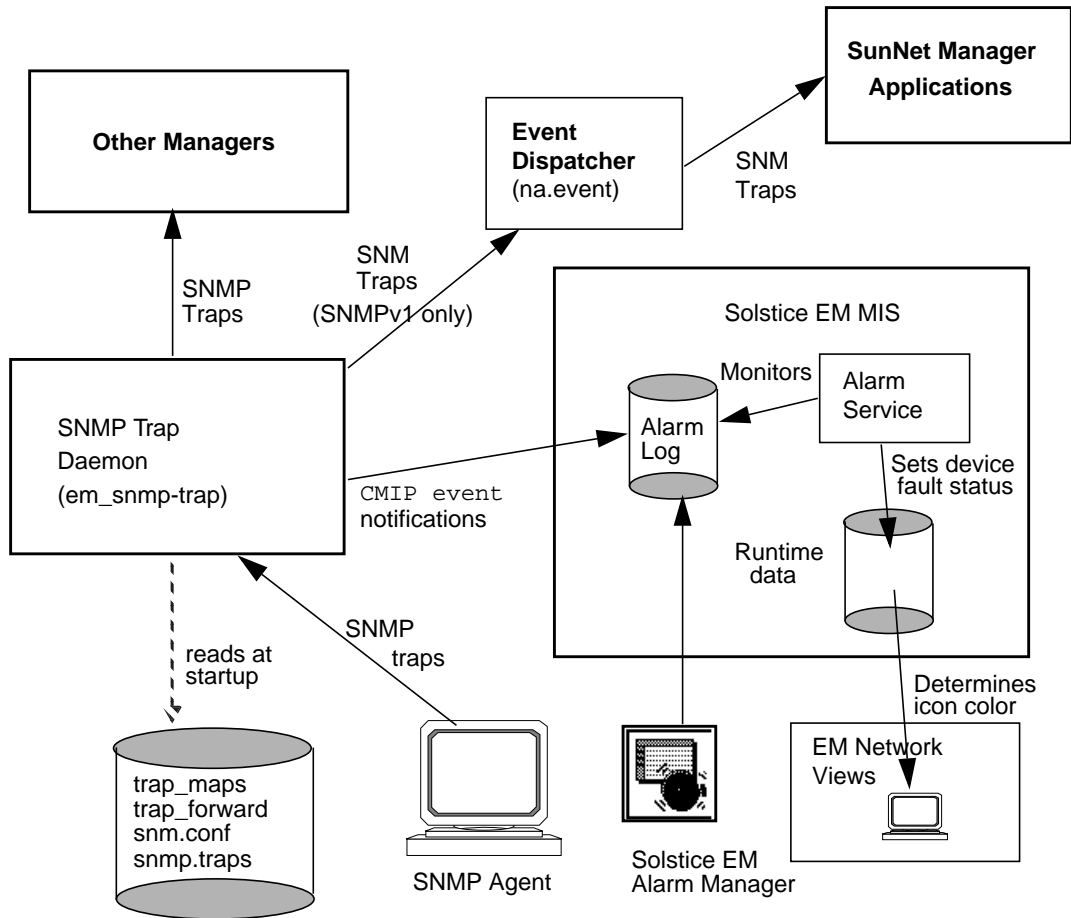


FIGURE 11-1 `em_snmp-trap` Operation

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 Solstice EM daemons at once, including the trap daemon and the MIS. (For information on the `em_services` command, refer to Chapter 2 in *Management Information Server Guide*.)
- If you want to start or stop the trap daemon by itself, you can use the `em_trapd` script.

11.2.1 Starting the Trap Daemon

▼ To Start the Trap daemon

Type the following:

```
# em_trapd start [<port_number>]
```

Port number is optional. You can use this option to start `em_snmp-trap` on a non-default port. By default, `em_snmp-trap` listens for incoming traps on port 162. For example, to start the trap daemon on port 412, enter the following command:

```
# em_trapd start 412
```

11.2.2 Stopping the Trap Daemon

▼ To Stop the Trap Daemon

The following command stops the trap daemon:

```
# em_trapd stop
```

You should use these commands if you want to stop the trap daemon on an MIS machine without 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.

11.3 The Structure of SNMP Traps

Throughout this chapter we will be making reference to the various fields that comprise the SNMP trap Protocol Data Unit (PDU). The structure of the SNMP traps for SNMPv1 and SNMPv2c differ. See the following sections for information specific to each SNMP version.

11.3.1 SNMPv1

The SNMP trap PDU for SNMPv1 has the fields indicated in the following figure.

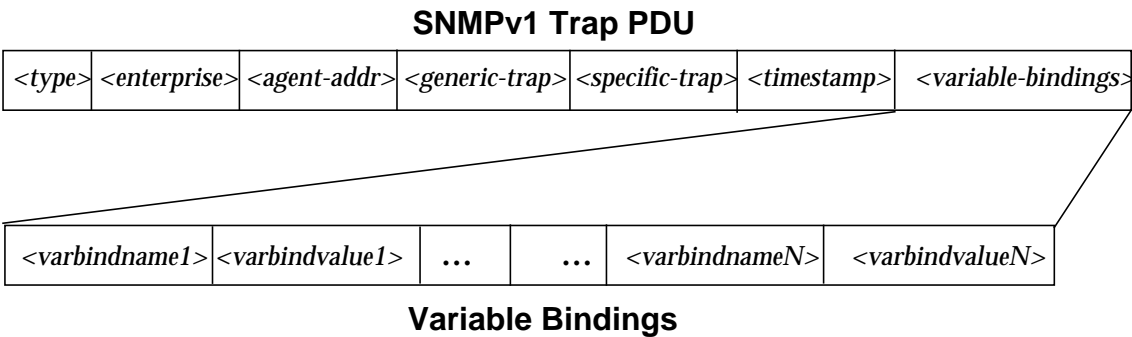


FIGURE 11-2 SNMPv1 Trap PDU Structure

The fields are as follows:

TABLE 11-1 SNMPV1 Field Descriptions

Field	Description
<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 <code>sysObjectID</code> attribute.
<agent-addr>	This is the IP address of the source of the trap.
<generic-trap>	This is an integer value in the range of 0 to 6 indicating the standard trap type. The standard trap types are listed in TABLE 11-2.
<specific-trap>	A device-specific value providing more information concerning to 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 vendor. 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 the following table.

TABLE 11-2 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 <code>ifIndex</code> for the interface.

TABLE 11-2 Standard SNMP Trap Types

Value of <generic-trap>	Trap Type	Description
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.

11.3.2 SNMPv2c

The SNMPv2c trap PDU differs substantially from the SNMPv1. The SNMPv2c PDU does not contain the following information:

- <enterprise>
- <agent-addr>
- <generic-trap>
- <specific-trap>
- <timestamp>

For SNMPv2c traps, some of this information is contained in the variable bindings. The <error-status> and the <error-index> fields are always set to 0 for SNMPv2c, as shown in the following figure.

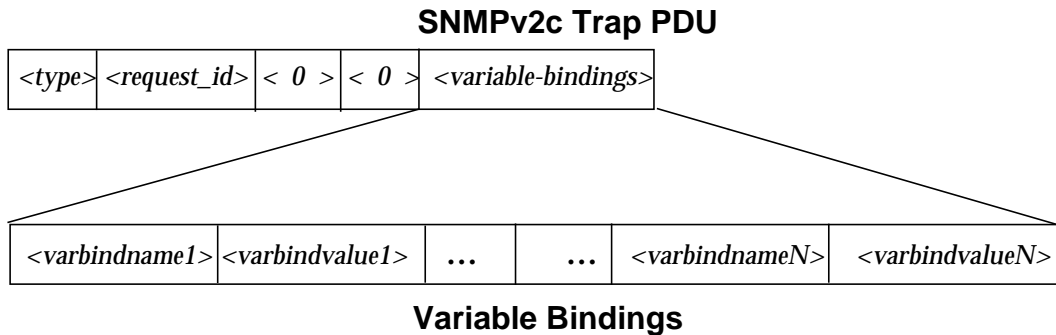


FIGURE 11-3 SNMPv2c Trap PDU Structure

The fields are as follows:

TABLE 11-3 SNMPV2c Field Descriptions

Field	Description
<code><type></code>	Indicates the type of SNMP message. (In this case, it indicates that this is a trap PDU.)
<code><request_id></code>	The sequential number of the trap.
<code><error_status></code>	This field is always set to 0 for SNMPv2c traps.
<code><error-index></code>	This field is always set to 0 for SNMPv2c traps.
<code><variable-bindings></code>	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.

SNMPv2c requires the first two variable bindings to be `sysUpTime` and `TrapOid`. If present, the enterprise Oid must be the last variable binding.

11.4 Default Trap Mapping

A default trap-mapping is configured automatically when you install the Solstice EM trap daemon. When an SNMP trap arrives, Solstice EM uses this default mapping for converting the SNMP trap into a CMIP event notification. For information on customizing the Solstice EM trap daemon, see Section 11.7 “Customizing the Mapping of SNMP Traps” on page 11-25. The default method for

specifying the source of the alarm is different for SNMPv1 and SNMPv2c. See the following sections for information specific to each version. The following default information is the same for both SNMPv1 and SNMPv2c:

- default `perceivedSeverity` values
- default `probableCause` values
- default `additionalText` information
- default event notification type

11.4.1 Default Method for Specifying the Source of an Alarm

11.4.1.1 SNMPv1

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. The following methods specify the alarm source for SNMPv1.

- 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 daemon 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.

11.4.1.2 SNMPv2c

When an SNMPv2c trap arrives, `em_snmp-trap` extracts the source IP address from the UDP header information 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. The following methods specify the alarm source for SNMPv2c.

- If there is a `cmipsnmpProxyAgent` object in the MIS corresponding to the source IP address in UDP header information, `em_snmp-trap` daemon's default 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 daemon 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.

11.4.2 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 Network Views by icon color. The default mapping of severities to colors is described in the following figure. If event notifications are to affect fault status indication, they need to have a `perceivedSeverity` value.

TABLE 11-4 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.

For SNMPv1, the default trap-mapping uses the *<generic-trap>* value to make severity assignments as follows:

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

Note – LinkUp traps automatically clears previous linkDown traps from the same router.

For SNMPv2c, the default severity is indeterminate.

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 11.7, “Customizing the Mapping of SNMP Traps.” For information on the Alarm Service, see Chapter 4.

11.4.3 Default probableCause Values

For SNMPv1, 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.

For SNMPv2c, the default `probablyCause` value is set to `TrapOid`.

11.4.4 Default additionalText Information

For SNMPv1 and SNMPv2c, 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)
 - This is used for SNMPv1 only
- The `<trapoid>` value - This is used for SNMPv2c only
- The attribute/value pairs from the trap variable bindings

For an SNMPv2c trap, the trap mapping mechanism tries to derive the enterprise and specific trap field values by using methods defined in RFC 2089.

An SNMPv2c trap may also contain the `TrapOid` value.

11.4.5 Default Event Notification Type

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

TABLE 11-5 Default IP Management Trap Event Types

Generic Trap Type	Event Notification
coldStart	coldStartTrap
warmStart	warmStartTrap
linkDown	linkDownTrap
linkUp	linkUpTrap
authentication Failure	authenticationFailureTrap
EGP Neighbor Loss	egpNeighborLossTrap
Enterprise specific	enterpriseSpecificTrap

The default trap-mapping options ensures that every incoming SNMP trap matches 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 vendors. How to do this is discussed below in Section 11.7, “Customizing the Mapping of SNMP Traps.”

11.4.6 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. This behavior is the same for both SNMPv1 and SNMPv2c.

11.5 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.

Solstice EM handles SNMPv1 and SNMPv2c traps that have no explicit mapping in the same manner.

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 `additionalText` 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.

11.6 Format of `trap_maps` File

The `trap_maps` file is a daemon that listens for incoming SNMP traps for forwarding to management stations. This file contains different parts including:

- Enterprise mapping blocks
- Mapping records
- `<attr-value>` definitions

11.6.1 Enterprise Mapping Blocks

The trap mapping file consists of blocks of records, with each block identified by the keyword `enterprise`. The form of these blocks of records is different for SNMPv1 and SNMPV2c. See the following sections for SNMPv1 and SNMPV2c enterprise mapping block information.

11.6.1.1 SNMPv1

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 `<enterprise-object-identifier>` in the block heading matches the `<enterprise>` field in the trap. Three important aspects of the enterprise heading:

- 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 enterprise block in the `trap_maps` file whose enterprise heading is most specific to the trap.

Traps are checked against the enterprise blocks in the `trap_maps` file. The more specific enterprise values are matched first.

11.6.1.2 SNMPv2c

Each block is in the following form:

```
enterprise <enterprise-oid> | NO-ENTERPRISE
{
  <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 `<enterprise-object-identifier>` in the block heading matches the `<enterprise>` field in the trap. Three important aspects of the enterprise heading:

- 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.
- NO-ENTERPRISE - If no enterprise information is found in the trap variable binding for an SNMPv2c trap, the block specified with NO-ENTERPRISE will be mapped.
- A trap is mapped by the enterprise block in the `trap_maps` file whose enterprise heading is most specific to the trap.

Traps are checked against the enterprise blocks in the `trap_maps` file. The more specific enterprise values are matched first.

11.6.2 Mapping Records

The trap-mapping records are defined by the event notification. You can use either standard notifications (see TABLE 11-6) or create your own custom event types. For more information, refer to the *Management Information Server (MIS) Guide*.

TABLE 11-6 Standard Event Notifications

Standard	Event Notification
ISO X.722	objectCreation
	objectDeletion
	attributeValueChange
	relationshipChange
	stateChange
	communicationsAlarm
	environmentalAlarm
	equipmentAlarm
	integrityViolation
	operationalViolation
	physicalViolation
	processingErrorAlarm

TABLE 11-6 Standard Event Notifications (*Continued*)

Standard	Event Notification
	qualityofServiceAlarm
	securityServiceOrMechanismViolation
	timeDomainViolation
IIMC	internetAlarm
Solstice EM-specific	snmAlarmEvent
	snmAlarmTrap
	nerveCenterAlarm
	coldStartTrap
	warmStartTrap
	linkDownTrap
	linkUpTrap
	authenticationFailureTrap
	egpNeighborLossTrap
	enterpriseSpecificTrap

The trap-mapping records are different for SNMPv1 and SNMPV2c. Both versions of SNMP use:

- <attr-value> Definitions
- CMIP notification managedObjectClass
- Attribute value types
- Wild cards for trap_mapping
- FDN templates

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. Logging the event type to the alarm log will ensure that the incoming traps cause appropriate changes in Network Views icon color. By default, only the following event types are excluded from the AlarmLog:

- objectCreation
- objectDeletion
- attributeValueChange
- stateChange

See the following sections for SNMPv1 and SNMPV2c enterprise mapping block information.

11.6.2.1 SNMPv1

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 11.5, “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.

Each SNMPv1 record in an enterprise block in the *trap_maps* file has the format shown below:

```
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 11-4 SNMPv1 Trap Mapping Record Format

Where

- *<generic-trap>* is an integer in the range of 0–6.
- *<specific-trap>* is an optional entry. Typically, *<specific-trap>* will be specified only when *<generic-trap>* is 6, indicating an enterprise-specific trap.
- *<alarm-type>* is any defined notification in the platform; a keyword DISCARD can be used to discard this trap.
- *<attr-name>* is a form [CONVERT] attribute=value;
See Section 11.6.3 “<attr-value> Definitions” on page 11-20 for more information.

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
    interfacesId=NULL/ifTableId=NULL/ifEntryId={varbindvalue1};;}
```

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>*.

11.6.2.2 SNMPv2c

The SNMMPv2c trap map record is similar to SNMPv1, except that trapOid is matched instead of generic and specific trap values. Once the match is found, that mapping record is applied.

Each SNMPv2c record in an enterprise block in the trap_maps file has the format shown in the following example.

```
TRAPOID <trapoid_list>
NOTIFICATION <alarm-type> | DISCARD
[ ATTRIBUTE-MAP <attr-name>=<attr-value>; ]
[                                     <attr-nameN>=<attr-valueN>;   ]
  FDN-MAP[ <FDN-template> ];;
```

Where:

- `<trapoid_list>` is a list of object identifiers, expressed in dot notation, which should match the `snmpTrapOID.0` varbind in SNMPv2c traps. A list of comma-separated trapoids can be used, and each trapoid can contain wildcards. See Section 11.7.5 “Wild Cards for trap_mapping” on page 11-32 for more information.
- `notification` is any defined notification in the platform; a keyword `DISCARD` can be used to discard this trap.
- `attribute-map` is a form `[CONVERT] attribute=value;`
See Section 11.6.3 “<attr-value> Definitions” on page 11-20 for more information. This field is optional.
- `managed_object_class` is optional. See Section 11.7.3 “Configuring CMIP notification managedObjectClass” on page 11-27 for more information.

For example:

```
enterprise 1.3.6.1.4.1.42
TRAPOID <trapoid_list>
NOTIFICATION communicationsAlarm
ATTRIBUTE-MAP
    perceivedSeverity=varbindvalue3;
    probableCause=varbindvalue2;
FDN-MAP
    interfacesId=NULL/ifTableId=NULL/ifEntryId={varbindvalue1};;
```

11.6.3 <attr-value> Definitions

The <attr-value> definitions are used by both SNMPv1 and SNMPv2c.

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

- a constant
- a trap variable binding value
- the keyword `$ALLVARS`

<attr-value> definitions are the same for SNMPv1 and SNMPv2c.

11.6.3.1 Constant

In the following example, the severity of the alarm is set to critical and a string constant is passed as `additionalText`.

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

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.

11.6.3.2 Trap Variable Binding Value

In the following example, `varbindvalue2` indicates the value of the second variable binding name/value pair in the SNMP trap.

```
probableCause=varbindvalue2;
```

11.6.3.3 Trap Variable Binding Name

Varbind names can be referenced in the `trap_maps` file using a `varbindNameN` variable analogous to a `varbindValueN` variable, where N is the varbind number.

In the following example, `varbindname2` indicates the value of the second variable binding name/value pair in the SNMP trap.

```
GENERIC-TRAP 6
    SPECIFIC-TRAP 13-25, 27, 29, 31-40
    NOTIFICATION enterpriseSpecificTrap
    ATTRIBUTE-MAP
        probableCause=varbindName2;
        perceivedSeverity=indeterminate;
        additionalText=$ALLVARS;
    FDN-MAP ; ;
```

11.6.3.4 Trap Variable Binding Index

Varbind names can be referenced in the `trap_maps` file using a `varbindIndexN` using variable analogous to a `varbindValueN` variable. In the following example, `varbindIndex2` indicates the value of the second variable binding name/value pair in the SNMP trap.

```
GENERIC-TRAP 6
    SPECIFIC-TRAP 13-25, 27, 29, 31-40
    NOTIFICATION enterpriseSpecificTrap
    ATTRIBUTE-MAP
        probableCause=varbindIndex2;
        perceivedSeverity=indeterminate;
        additionalText=$ALLVARS;
    FDN-MAP ;
```

11.6.3.5 Embedding Strings in varbind Expressions

Solstice EM supports mapping of notification attributes to strings embedded in SNMP trap varbind variables (for example, `$varbindNameN`, `$varbindValueN`, and `$varbindIndexN`). Solstice EM also allow you to embed simple arithmetic expressions in varbind variables. These simple arithmetic expressions contain basic arithmetic operations on integer varbind values and integer constants. The arithmetic expressions are embedded by enclosing them between backquotes (`). The following operations are supported in the arithmetic expressions.

- Addition, for example, "`...'$varbindValue1 + 10'...`"
- Subtraction, for example, "`...'$varbindValue2 - $varbindValue1'...`"
- Multiplication, for example, "`...'$varbindValue3 * 16'...`"
- Division, for example, "`...'$varbindValue2 / $varbindValue1'...`"
- Unary minus, for example, "`...'- $varbindValue1'...`"

The following is an example of an embedded arithmetic expression.

```
enterprise 1.3.6.1.4.1 {  
    GENERIC-TRAP 6  
    SPECIFIC-TRAP 50-55  
    NOTIFICATION enterpriseSpecificTrap  
    ATTRIBUTE-MAP  
        probableCaus=varbindValue3;  
        perceivedSeverity=minor;  
        additionalText="Board number '$varbindValue1 * 10 +  
$varbindValue2' has a problem of type $varbindValue3";  
    FDN-MAP ;;  
}
```

11.6.3.6 Defining and Using varbind-to-substring Tables

Tables that map integer SNMP trap varbind values to substrings can be defined by a table construct in the `trap_maps` configuration file. Such substrings can be embedded in strings that map CMIP notification attributes (in an `enterprise ATTRIBUTE-MAP` clause) by indexing the corresponding varbind-to-substring table with an integer varbind value or an integer constant. The syntax of the table construct is as follows:

```
table <table-id> {  
    <index1>: <substring1>  
    [<indexN>: <substringN>]  
}
```

where

- `<table-id>` is the table identifier comprised of a case-sensitive sequence of alphanumeric characters beginning with a letter (for example, `ProblemDescription`, `NetworkCardType`, `contacts`). The table identifiers must be unique in the scope of its `trap_maps` file. All subsequent duplicate identifier tables found are discarded. A proper message is logged to indicate this parsing error.
- `<indexN>` is the table entry index, a 32-bit signed integer constant in decimal notation, e.g., 0, 64, -900. This table entry index must be unique within the scope of the table where it is defined.
- `<substringN>` is the table entry substring, a sequence of printable characters enclosed by a pair of double quotes ("), which cannot be part of the string.

After being defined, a table can then be indexed by integer varbind values—for example, \$varbinValueN, or integer constants (32-bit signed integer constants in decimal notation) from within a CMIP notification attribute mapping entry (in the enterprise ATTRIBUTE-MAP clause). Such table references have the following syntax:

```
ATTRIBUTE-MAP
...
<attr-name> = " ...$<table-id>[<index>]..." ;
...
```

where *<index>* is one of the following:

- An integer varbind value (for example, \$varbindValueN whose type is SNMP INTEGER, Counter, Gauge, or TimeTicks, converted to 32-bit signed integer)
- A 32-bit signed integer constant in decimal notation (for example, 0, 16, -120)

For example:

```
table BoardType {
    10: "14.4K modem",
    20: "28.8K modem",
    30: "56K modem"
}

table ProblemDescription {
    1: "data overrun",
    3: "power fluctuation",
    7: "CRC error"
}

enterprise 1.3.6.1.4.1 {
    GENERIC-TRAP 6
    SPECIFIC-TRAP 8-10
    NOTIFICATION enterpriseSpecificTrap
    ATTRIBUTE-MAP
        probableCause=varbindValue1;
        perceivedSeverity=major;
        additionalText="Board $Boardtype[30] of modem pool has a
problem of type $ProblemDescription[$varbindValue1].";
    FDN-MAP ;;
}
```

11.7 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.

11.7.1 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.

11.7.2 How to Customize SNMP Trap Mapping

This section provides the steps to configure `em_snmp-trap` for SNMPv1 and SNMPv2c trap mapping.

▼ To Customize SNMP Trap Mapping

1. Collect information on enterprise-specific traps.

If you want to add mapping blocks to map enterprise-specific traps, consult the vendor 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 value for SNMPv1 and trapoid for SNMPv2c.

This is discussed in Section 11.5, “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 11.7, “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 11.7.6, “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 Event Logs 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 the above example.

5. Save the file.

`/etc/opt/SUNWconn/em/conf/trap_maps` is the file location.

6. Restart the trap daemon.

- a. To stop the trap daemon, enter the following command (as `root`):

```
#em_trapd stop
```

- b. 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.

11.7.3 Configuring CMIP notification managedObjectClass

The `managedObjectClass` of a notification of an SNMP trap-to-CMIP notification mapping defaults to one of the following:

- If the FDN is defined in the FDN-MAP clause of the trap mapping entry, it defaults to `actualClass`, for example, `globalForm:{2 9 3 4 3 42}`
- If the FDN is not defined in the FDN-Map clause, it defaults to `"iimcManagementProxyMIB":cmipsnmpProxyAgent class`

You can override the `managedObjectClass` default if the FDN is defined in the FDN-MAP clause of the trap mapping entry. To override the default value (`actualClass`), specify the managed object class of the notification from within a trap mapping entry in the `trap_maps` file. The syntax of the `CLASS-MAP` clause is:

```
enterprise <enterprise-object-id> {  
    ...  
    [CLASS-MAP <managed-object-class>]  
    FDN-MAP <FDN-template> ;;  
}
```

where

- `<managed-object-class>` is the GDMO managed object class to be mapped to the notification `managedObjectClass`, for example, `"IIMCCISCO-MIB":lsystem`, or `circuit`
- In case `CLASS-MAP` clause is present, a FDN is expected in the `<FDN-template>` of the corresponding FDN-MAP clause

The following illustrates the usage of CLASS-MAP clause:

```
enterprise 1.3.6.1.4.1 {  
    GENERIC-TRAP 6  
    SPECIFIC-TRAP 100  
    NOTIFICATION enterpriseSpecificTrap  
    ATTRIBUTE-MAP  
        probableCause=varbindValue1;  
        perceivedSeverity=critical;  
        additionalText="$ALLVARS;  
    CLASS-MAP "OP1 Library Vol. 1":computerSystem  
    FDN-MAP /systemId=name:"mars"/computerSystemId="paris" ;;  
}
```

11.7.3.1 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 for SNMPv1 or <trapoid> for SNMPv2c
- All of the attribute/value pairs comprising the trap variable bindings

Note – The SNMPv2c trap does not usually have the enterprise and specific trap fields. When \$ALLVARS is used for an SNMPv2c trap, the trap mapping mechanism tries to derive these values by using methods defined in RFC 2089. If the snmpTrapEnterprise.0 variable binding is in the list of variable bindings, it is ignored because it would be a duplicate of the enterprise field.

In this example,

```
additionalText = $ALLVARS;
```

Sample output from this mapping would be an additionalText field that looks like the following:

```
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 Solstice 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 Network Views icon colors that represent the importance of a network event. The valid severities and their default associated icon colors are listed in TABLE 11-4.

The mapping of severity to displayed color is controlled by the Solstice EM Nerve Center; this mapping is user-configurable via the Design Advanced Requests application. See Chapter 18 for more information.

The interpretation of *<specific-trap>* values for enterprise-specific traps depends upon the particular implementation by the product vendor. 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 11.5, “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.

11.7.3.2 The Keyword \$NORFC2089

This keyword is used only with a text field and used only for SNMPv2c. The \$NORFC2089 keyword is similar to the \$ALLVARS keyword in that it specifies that the text field is to receive the attribute/value pairs comprising the trap variable bindings.

When the \$NORFC2089 keyword is used, the varbinds are printed out in the order received instead of printing out in a format similar to that of SNMPv1.

In this example,

```
additionalText = $NORFC2089;
```

Sample output from this mapping would be an `additionalText` field that looks like the following

```
additionalText = sysUpTime = 142100, snmpTrapOID = {1 3 6 1 6 3 1
1 5 3 }, ifIndex = 1
```

Consider the example in case of `$ALLVARS`,

```
additionalText = $ALLVARS;
```

Sample output from this mapping would be an `additionalText` field that looks like the following:

```
additionalText = "enterprise = 1.3.6.1.6.3.1.1.5,
specificTrap = 0, sysUpTime = 128100, snmpTrapOID = {1 3 6 1 6 3 1
1 5 3}, ifIndex = 1"
```

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

11.7.4 Attribute Type Mapping

Attribute type mapping specifies how attribute types in a trap are mapped to attribute types in a notification. The permitted mapping between types is provided in the following table.

TABLE 11-7 Attribute Value Type Conversions

Varbind Type	Permitted Event-Report Attribute Types
INTEGER (+ Counter, Gauge, TimeTicks	OCTET STRING, DisplayString, Opaque
OCTET STRING and DisplayString	INTEGER, OBJECT IDENTIFIER, NetworkAddress, DisplayString
OBJECT IDENTIFIER	OCTET STRING, Opaque, DisplayString
NetworkAddress	OCTET STRING, Opaque, DisplayString
Opaque	Opaque, OCTET STRING

Note – If you want to convert an OCTET STRING to an INTEGER, the OCTET STRING must not contain text.

To specify the attribute type mappings use the following:

```
[ CONVERT ] <attribute> = <value>
```

where:

- <attribute> is any valid attribute type for the corresponding notification
- <value> is one of the following:
 - a varbind variable, which can be one of the following:
 - i. varbindvalue - yields the value of a varbind
 - ii. varbindindex - yields the index of the SNMP object the varbind refers to
 - iii. varbindname - is the OID of the object the varbind refers to
 - ASN.1 string format constant. If *value* is an ASN.1 string format constant, the value must be the same as the attribute.

Specify the CONVERT keyword only if:

- <value> is a varbind
- the <value> and <attribute> types are different

If you want value and attribute types to be the same, you must omit the CONVERT keyword. Otherwise, the translation will fail.

The following illustrates the usage of attribute type conversion:

```
GENERIC-TRAP 6
  SPECIFIC-TRAP 101
  NOTIFICATION internetTrapINTEGERv1Alarm
  ATTRIBUTE-MAP
    probableCause = localValue : 101;
  CONVERT
    objINTEGERv1 = varbindvalue1;
  FDN-MAP ;;
}
```

11.7.5 Wild Cards for trap_mapping

The trap_mapping wild cards are similar for SNMPv1 and SNMPv2c. See the following table for version specific information.

TABLE 11-8 Wild Cards for trap_mapping

Wild Card	Description	Examples	
		SNMPv1	SNMPv2c
Match All	An asterisk wildcard can used to match all traps with a specific enterprise OID.	GENERIC-TRAP *	TRAPOID *
Ranges	A dash between two numbers can be used to match a range of values. For example, a numeric range can be listed which assigns different severities based on a trap's value.	SPECIFIC-TRAP 0 - 12	TRAPOID 0 - 12
Open-Ended Ranges	An asterisk can be used to match a specific number and higher.	SPECIFIC-TRAP 26 - *	TRAPOID 26 - *
Sets	A comma-separated set of values can be given to match a non-continuous set of values. The set may contain sub-ranges.	SPECIFIC-TRAP 13 - 25, 27, 29, 31 - 40	TRAPOID 13 - 25, 27, 29, 31 - 40

TABLE 11-8 Wild Cards for `trap_mapping` (Continued)

Wild Card	Description	Examples	
		SNMPv1	SNMPv2c
Varbind Names	Varbind names refer to the part of a varbind instance OID that names the object being referenced, minus index information. Varbind names can be referenced in the <code>trap_maps</code> file using a <code>varbindNameN</code> variable analogues to a <code>valueValue</code> variable, where <i>N</i> is the varbind number.	1.3.6.1.2.1.2.2.1	1.3.6.1.2.1.2.2.1
Varbind Indexes	Varbind indexes are the index value of varbind instances. They are 0 for scalar instances and index numbers for table entries. Varbind names can be referenced in the <code>trap_maps</code> file using a <code>varbindIndexN</code> using variable analogues to a varbind <code>valueValue</code> variable.		
Strings Containing Varbind Values	A varbind value, name, or index is placed within a string by preceding its name with a dollar sign. Varbind values are converted into <code>DisplayString</code> representation for inclusion in a string.	<code>causeText = "The varbind value is \$varbindValue";</code>	<code>causeText = "The varbind value is \$varbindValue";</code>

11.7.6 Using FDN Templates to Specify the Source of a Trap

For SNMPv1, when SNMP traps, `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. For SNMPv2c, when SNMP traps arrive, `em_snmp_trap` extracts the source IP address from the UPD header information in the SNMP trap and uses this information to determine if there is an object configured in the MIS to present the 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"
```

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.

11.7.6.1 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>
```

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`:

```
internetSystemId=NULL
```

Thus, if this RDN is appended to the FDN for bigguy's cmipsnmpProxyAgent in FIGURE 11-5, the result is an FDN that points to the internetSystem object instance for this SNMP agent:

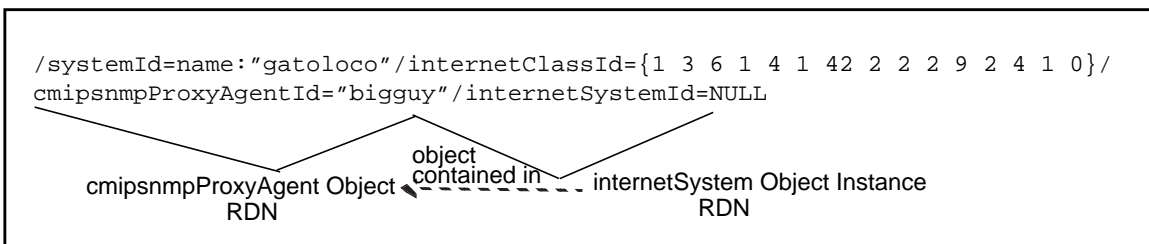


FIGURE 11-5 Sample FDN for internetSystem Group Object Instance

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

`systemId`, `cmipsnmpProxyAgentId`, and `internetSystemId` 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).

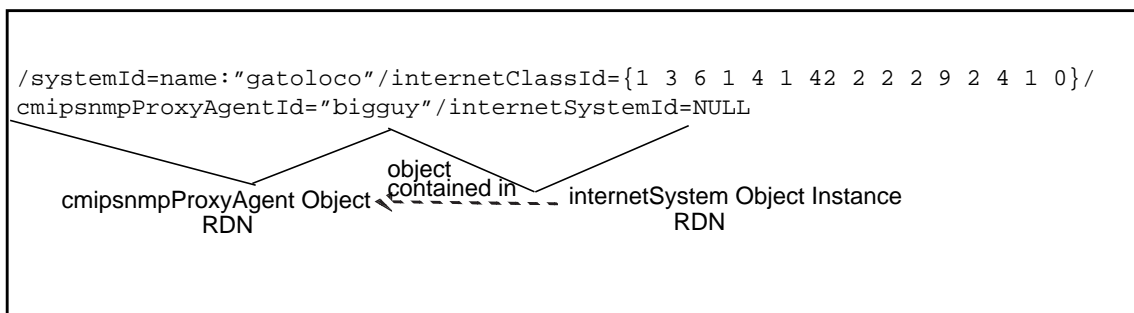


FIGURE 11-6 Sample ifTable FDN

The value of a naming attribute depends on whether the object is a scalar object or a columnar object. (i.e., whether the object can have only one instance or multiple instances in an SNMP agent). For scalar objects, the value of the naming attribute is NULL. For columnar objects, the value is a sequence of the values of the indices for that entry. For example, there can be only one ifTable in an SNMP agent, hence `ifTableId=NULL`, but `ifEntryId={ifIndex2}`, `ifIndex` being the index for ifTable.

11.7.6.2 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. FDN templates can follow one of two formats:

Standard Format:

```
<naming-attribute>=NULL / <naming-attribute>=NULL / <naming-attribute>={ <instance-indices>}
```

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* RDN 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 11-7 shows an FDN template with the standard format. Instance indices 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 10-4 has the following FDN template:

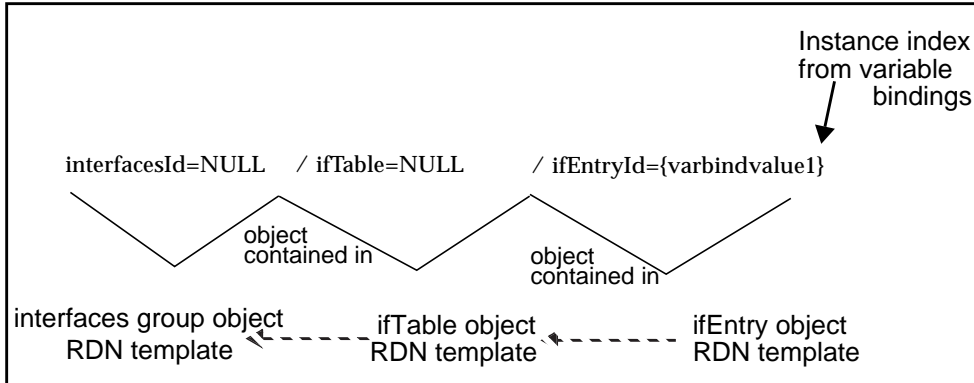


FIGURE 11-7 Sample FDN Template

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

`varbindvalueN`

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

11.8 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 forwarding 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.

▼ To Connect the Trap daemon to the MIS

1. **Edit the `/var/opt/SUNWconn/em/conf/EM-config` file on the MIS machine.**

Add the name of the trap daemon 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.

11.8.1 Filtering SNMP Traps for Other Managers

The Solstice EM SNMP trap daemon has the ability to filter raw (unprocessed) SNMP traps to other managers. The trap daemon can forward or exclude/discard the raw SNMP traps (both SNMPv1 and SNMPv2c). 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; an ASCII text file that the `em_snmp-trap` daemon reads whenever it starts.

The `trap_forward` file has the following format. The keyword is different for SNMPv1 and SNMPv2c.

```
MIS_HOSTS: <MIS-host1>,<MIS-host2>
SNMP_HOSTS: <mgr_hostname1>:<port1>,<mgr_hostname2>:<port2>
or
SNMPV2_HOSTS: <mgr_Host>:<port1>,<mbr_hostname2>:<port2>
```

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

The `SNMP_HOSTS/SNMPV2_HOSTS` line contains the hostname and port number for each SNMP manager that is to receive the raw SNMPv1/SNMPv2c traps. Entries for multiple managers are separated by commas.

To exclude/discard the filtered raw SNMP traps, enter DISCARD in the trap-filtering-record in the trap_forward file. The trap-filtering-record for SNMPv1 has the following format:

```
GENERIC-TRAP gen_trap_range  
SPECIFIC-TRAP spec_trap_range OPTIONAL  
DISCARD: <mgr_hostname1>:<port1>, <mgr_hostname2>:<port2>
```

The trap-filtering-record for SNMPv2c has the following format:

```
TRAPOID trapoid_list  
DISCARD
```


Configuring Communication With CMIP Agents

This chapter provides detailed instructions for installation and configuration of the components required to manage Common Management Information Protocol (CMIP) agents.

This chapter describes the following topics:

- Section 12.1 “Tasks for Setting Up Your System to Manage CMIP Agents” on page 12-1
- Section 12.2 “Preparing the System for CMIP Configuration” on page 12-4
- Section 12.3 “Compile and Load CMIP Agent Object Types into MIS” on page 12-6
- Section 12.4 “Starting and Configuring SunLink OSI” on page 12-7
- Section 12.5 “Access Control” on page 12-8
- Section 12.6 “Starting and Configuring SunLink CMIP 9.0” on page 12-8
- Section 12.7 “Starting and Configuring the CMIP MPA” on page 12-10
- Section 12.8 “Runtime Parameters” on page 12-13
- Section 12.9 “Configuring Multiple MPAs on One System” on page 12-19

12.1 Tasks for Setting Up Your System to Manage CMIP Agents

The following list summarizes the activities that you must complete before your system can manage a CMIP Agent. These procedures must be performed as `root`.

▼ To Prepare Your System to 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 Enterprise Manager* (Solstice EM). For those not shipped with Solstice EM, you must compile the definitions and load them into the MIS.

3. Start up and configure SunLink OSI 8.1 /9.0.

SunLink OSI provides access to 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 9.0.

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 Solstice EM CMIP MPA.

The Solstice EM CMIP MPA translates CMIP requests and responses to and from Solstice EM. It uses the services provided by SunLink CMIP. Before Solstice EM can access the objects in the CMIP Agent, the CMIP agent must be configured in the MIS. You can use the Solstice EM Object Properties/Create Object to configure CMIP objects. Refer to *Managing Your Network* 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/cmip/sbin` and `/opt/SUNWconn/sbin/`. See Chapter 4 in *Installation Guide* for instructions on setting your `PATH` environment variable.

The following figure illustrates the configuration procedure.

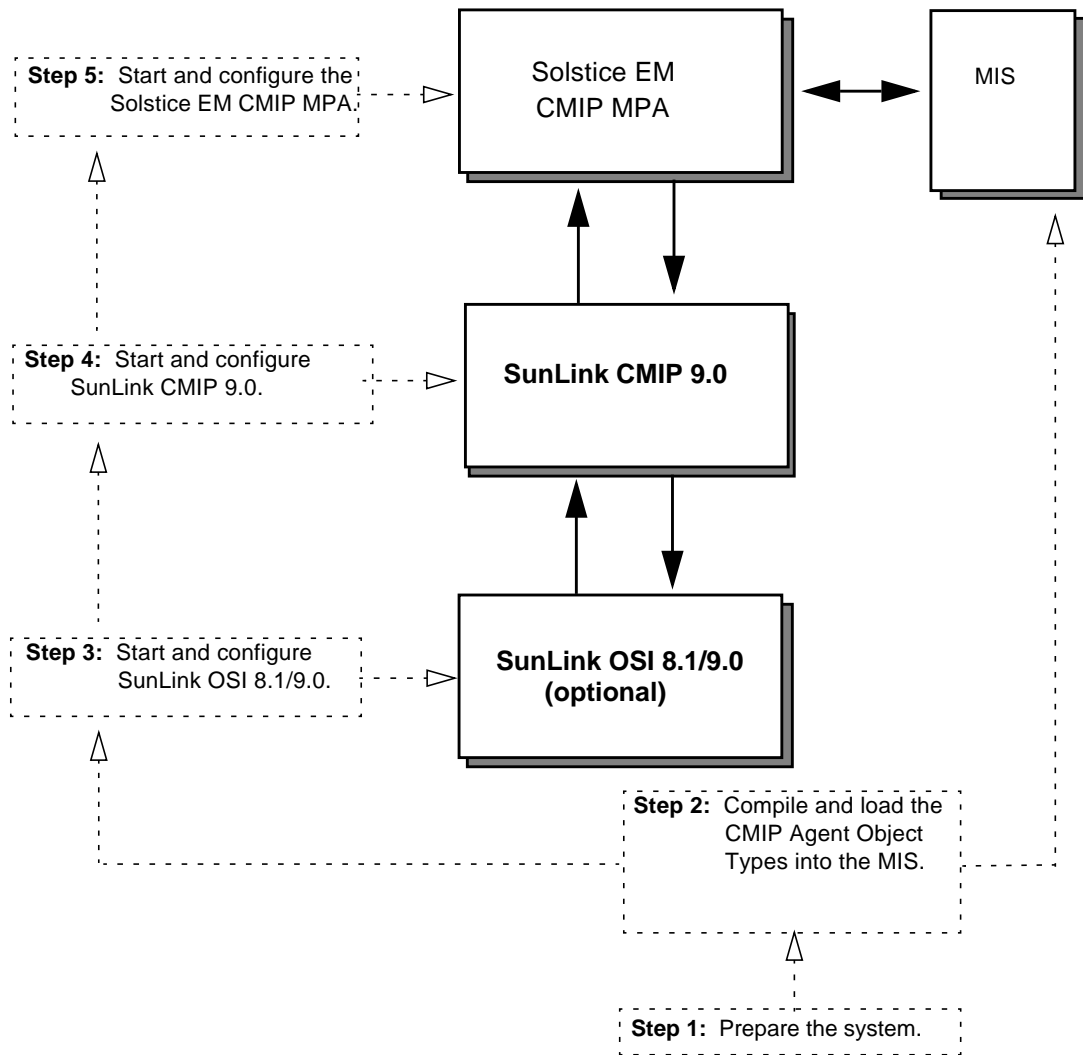


FIGURE 12-1 Configuring Solstice EM for Communication with CMIP Agents

The following sections provide detailed instructions to configure Solstice EM for CMIP Agent communication.

12.2 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.

▼ To Prepare Your System for Configuration

1. Determine the distribution model.

See Chapter 11 for complete instructions. Before you begin, you must determine the distribution model you will use for CMIP communications.

2. Install the required SunLink products.

See Chapter 12 for instructions. Install the following products, as required for your environment:

a. **SunLink CMIP 9.0 RT or SunLink CMIP 9.0 SDE**

b. **SunLink OSI 8.1/9.0 (optional)**

c. **SunLink X25 9.0 or above (optional)**

Note – You can also use CMIP 8.1.2 and CMIP 8.2.2.

3. Gather your configuration information.

See Chapter 12 for instructions. Before proceeding, you should gather all the information that you will need to complete the configuration process.

12.2.1 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:

- Which MISs will the MPA be communicating with
- How many agents are there and how are they configured
- Address of the MPA
- Address of the agents

12.2.2 Installing the Required SunLink Products

You can use one of the following configurations of SunLink products for CMIP communications, depending on your target environment:

- SunLink CMIP 9.0, if using RFC1006.
- SunLink CMIP 9.0 & SunLink OSI 9.0, if using CLNP/LLC.
- SunLink OSI 9.0 & SunLink CMIP 9.0 SDE & SunLink X25 9.0, if using CONS/X.25.

You should have the appropriate patches for your installation 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.

▼ To install the lowest layer of the protocol stack

1. SunLink OSI 9.0 (Optional)

First, install SunLink OSI 9.0. See the provided documentation for details on installation. A typical installation for OSI 9.0 will include the following product packages:

- SUNWcorpc
- SUNWcosia
- SUNWcosib
- SUNWcosic
- SUNWcosid
- SUNWlicsw
- SUNWlit

2. SunLink CMIP 9.0 (Required)

Install either SunLink CMIP 9.0 RT or SunLink CMIP 9.0 SDE. See the provided documentation for details on installation. A typical installation for SunLink CMIP 9.0 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 have already installed a version for SunLink OSI 9.0.

3. SunLink X.25 9.0. (Optional)

You will be required to install SunLink X.25 if you are communicating 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.

12.2.3 Gathering Your Configuration Information

You will need the following information to configure 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

12.3 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 the *Management Information Server (MIS) Guide* for instructions on using these compilers.

12.4 Starting and Configuring SunLink OSI

To configure SunLink OSI 9.0 to communicate with CMIP agents, do the following.

Note – If you will be using RFC1006, you need not set up the Network Layer Address.

▼ To Configure SunLink OSI 9.0 to Communicate with CMIP Agents

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 9.0 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 9.0 Communication Platform Administrator's Guide*.

5. Restart `osinetd`

After you have entered all of your configuration information, use `ositool` to restart `osinetd`.

12.5 Access Control

Access control for associations, requests, and notifications processed by the CMIP MPA is enforced by using a username that is made available at start-up. If a username is specified at the command line, it is used as a fallback value.

12.6 Starting and Configuring SunLink CMIP 9.0

To configure SunLink CMIP 9.0 to communicate with CMIP agents, do the following.

▼ To Configure SunLink CMIP 9.0 to Communicate with CMIP Agents

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 displays.

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 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. The parameter value is then multiplied times 10 to calculate the actual length of the timeout in seconds. The SunLink CMIP timeout is 50 seconds.

If you intend to issue requests to CMIP Agents, which require a longer timeout, you should increase the value of this parameter. The maximum allowable Timeout value is 127, which equals 1270 seconds.

If you do not want to see the communication alarm, you can set the inactivity timer on the `cmiptool` to 0 on the agent side. In this case, the agent does not issue CMI-RELEASE-Rq if it idles for a specified time.

When the communication between manager and agent idles for an amount of time, which is specified on the inactivity timer from `cmiptool` on the agent side, the CMIP stack of the agent will issue CMI-RELEASE_Rq to the CMIP MPA. When the CMIP MPA receives this request, it sends MCS-RELT-Rs response to the request to the agent and at the same time the CMIP MPA needs to notify the user that the association is down between manager and agent. That’s why the CMIP MPA reports a communication alarm.

5. Select Apply.

6. Exit `cmiptool`

12.7 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. 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.

When a CMIP MPA is started, a unique transient CMIP MPA object (containing the CMIP MPA runtime parameters) is created in the Auxiliary Server Container. The object is named by the `auxServerId` attribute. The value of the name is a concatenation of the MIS host and the CMIP MPA port number.

```
/systemId=name:"<hostname>"/auxServerType="cmip_mpa" /  
auxServerId="<host-name>":<port-number>
```

```
/systemId=name:"<hostname>"/auxServerType="cmip_mpa" /  
auxServerId="<host-name>":<port-number>
```

Note – When the CMIP MPA is externally terminated, it does not delete its objects and terminate its connections as it usually does when it terminates in an orderly manner. If you restart the CMIP MPA after an external termination and reconfigure it to start at a different port, a new CMIP MPA object is created in the Auxiliary Server Container. This action invalidates (but does not delete) the older object that was created when the CMIP MPA was first started. In this case, if you perform a scoped get request operation that includes the container's objects, the get request returns two objects rather than one.

The Solstice EM CMIP MPA performs association management. You can choose what Solstice EM will do when a previously established association goes down. The two choices are the default CMIP MPA association recovery algorithm and an agent silent recovery-free algorithm.

- The default recovery algorithm generates a `communicationsAlarm` notification indicating an association is found broken only after all attempts to re-establish the association fail. If a failing agent/association recovers fast enough for the association to be re-established before or by the last retry, no `communicationsAlarm` is emitted.
- The agent silent recovery-free algorithm makes the CMIP MPA generate a `communicationsAlarm` when an association is found broken and before retrying to re-establish the association.

In both cases, if a `communicationsAlarm` is sent to the MIS, it is cleared by the MPA when the connection is re-established.

To choose the agent silent recovery-free management algorithm, set the `EM_CMIP_MPA_SEND_ALARM_EARLY` variable to `YES` in the shell from which the CMIP MPA daemon will be started.

```
EM_CMIP_MPA_SEND_ALARM_EARLY <YES/NO>
```

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

1. **Start the CMIP MPA by entering the following command at the operating system prompt:**

```
host# /etc/rc2.d/S98cmipmpa start
```

2. **Start the Object Properties/Create Object (`em_oct`) by typing the following command at the command line:**

```
hostname% em_oct -cmip [options] &
```

The Object Properties/Create Object enables you to configure objects managed under Solstice EM. For detailed instructions on using OCT, refer to *Managing Your Network*.

3. Enter the appropriate information into the appropriate fields (see the following table).

TABLE 12-1 Object Properties/Create Object Fields

Entity Name	Specify the name of the remote agent with which you want to communicate. The value can be a string, object identifier (OID), or distinguished name (DN). Click the down arrow to see a list of known agents. To delete an agent, click Delete, select an agent from the resulting list, then click OK. After specifying an agent, all the other fields in this window for which information exists in the agent 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 agent. For example, if the remote agent is located on a machine called poignant, enter /systemId=name:"poignant" in the MO DN field and click Add./systemId=name:"poignant" will appear in the Agents DNs field. To delete a DN, select the one you want to delete and click Delete.
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	You must enter the appropriate Presentation Selector, Session Selector, Transport Selector, and Network SAP for the agent you are configuring. Click Apply to create the object, or click OK 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: 4700040006000108002011e7f001

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.

12.8 Runtime Parameters

Access control for associations, requests, and notification processed by the CMIP MPA is enforced by using a username that is made available at start-up. If a username is specified at the command line, it is used as a fallback value.

12.8.1 Auxiliary Server Container

The Auxiliary Server Container is a persistent object created with `cmip_mpa` as its naming attribute value. It enables you to access the CMIP MPA runtime parameters programmatically.

The containment of the Auxiliary Server defined under its name binding is as follows:

```
/systemId=name:"<hostname>"/auxServerType="cmip_mpa"
```

Every time a CMIP MPA is started, a unique CMIP MPA object is created in the Auxiliary Server Container (which is a subtree of the MIT) as illustrated in the figure.

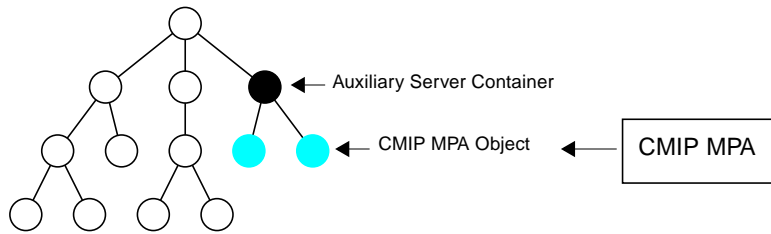


FIGURE 12-2 Auxiliary Server Container

12.8.2 CMIP MPA Object

When a CMIP MPA is started a unique transient CMIP MPA object (containing the CMIP MPA runtime parameters) is created in the Auxiliary Server Container. The object is named by the `auxServerId` attribute. The value of the name is a concatenation of the MIS host and the CMIP MPA port number.

```
/systemId=name:"<hostname>"/auxServerType="cmip_mpa"/  
auxServerId="<host-name>":<port-number>
```

Note – When the CMIP MPA is externally terminated, it does not delete its objects and terminate its connections as it usually does when it terminates in an orderly manner. If you restart the CMIP MPA after an external termination and reconfigure it to start at a different port, a new CMIP MPA object is created in the Auxiliary Server Container. This invalidates (but does not delete) the older object that was created when the CMIP MPA was first started. In this case, if you perform a scoped get operation that includes the container’s objects, the get function returns two objects rather than one.

12.8.3 em_cmip Parameters

Each of the parameters of the CMIP MPA has a corresponding environment variable. Some parameters can be set by both command line parameters and environment variables. In this case, the command line value takes precedence. For example, if the environment variable is set to NO and the command line is set to YES, `em_cmip` will run with the YES value for the option.

Because useful names are used in all cases, you must match the CMIP MPA command line parameters to the CMIP MPA environment variables.

The optional parameters are shown in the following table:

TABLE 12-2 em_cmip Parameters

Command Line Parameter	Corresponding Environment Variable	Description
-access_on	EM_CMIP_MPA_ACCESS_ON	An access control flag whose presence enables access control (turns on agent-role access control) in the CMIP MPA
-apcnt <dotted-OID-string>	EM_CMIP_MPA_APCNT <dotted-OID-string>	Identifies the set of application service elements required by the distributed application initiating the communication.
-apname <AETitle inOID:AeQualifier:APIInvoke Id:AEInvokeId>	EM_CMIP_MPA_APNAME <AETitle in OID:AeQualifier:APIInvokeId:AEInvokeId>	Requires the specification of a presentation address, in addition to the application entity title (AE title).
-debug	EM_CMIP_MPA_DEBUG	This is a backward compatibility option which turns on the cmip mpa debug output.
-debug mpa_access	EM_CMIP_MPA_DEBUG_ACCESS_CONTROL	Turns on access control functionality debug information. This provides information on whether or not cmip messages pass access control rules, how the response will be processed, and whether or not fallback users are being used.
-debug mpa_debug	EM_CMIP_MPA_DEBUG	Turns on em_cmip debug output. This information shows incoming and outgoing cmip messages, association establishment and control, and error conditions.
-fallback_user <user-name>	EM_CMIP_MPA_FALLBACK_USER	An access control identifier string assigned to an association and placed into PDUs sent to the MIS. It is a fallback value to be used only if a designated user name cannot be found in the UAM.
-help	N/A	Prints list of options and environmental variables for the em_cmip mpa command.
-host <mis-host-name>	EM_MIS_DEFAULT_HOST <host-name> EM_SERVER <host-name>	Specifies the host name of the system on which the MIS is running.

TABLE 12-2 em_cmip Parameters *(Continued)*

Command Line Parameter	Corresponding Environment Variable	Description
-i <idle_timeout>	EM_CMIP_MPA_IDLE_TIMEOUT <milliseconds>	
-log_file <log-file>	EM_CMIP_MPA_LOG_FILE <log-file>	Allows the user to specify a log file to output debug information to. The default location is /var/opt/SUNWconn/em/debug/em_cmip.log
-max_retrys <max-retrys>	EM_CMIP_MPA_ASSOC_MAX_RETRIE S <retrycount>	The number of times the cmip mpa will attempt to resend a request to a remote object.
-mis		Configures an MIS object.
-n <nsap>	EM_CMIP_MPA_NSAP <nsap>	Allows the network address to be specified when the cmip mpa binds to the cmip stack. The default is specified by using the cmiptool in /opt/SUNWconn/cmip/sbin/cmiptool and is installed as hex value of host ip address where Solstice EM is installed.
-p <psel>	EM_CMIP_MPA_PSEL <psel>	Allows the presentation selector address to be specified when the cmip mpa binds to the cmip stack. The default is specified by using the cmiptool in /opt/SUNWconn/cmip/sbin/cmiptool and is installed by Solstice EM as dflt.
-port <port-number>	EM_CMIP_MPA_DEFAULT_PORT <portnum>	The cmip mpa connects to the MIS via a TCP connection. The default value of this connection is on port 5557 for the cmip mpa side.
-r <req_timeout>	EM_CMIP_MPA_REQUEST_TIMEOUT <milliseconds>	The length of time the cmip mpa will wait after sending a request to a remote object for a response to return.
-retry_delay<milliseconds>	EM_CMIP_MPA_ASSOC_RETRY_DELA Y <retrydelay>	The time the cmip mpa will wait before resending a request.
-rpc		Configures an RPC object.
-s <ssel>	EM_CMIP_MPA_SSEL <ssel>	Allows the session selector address to be specified when the cmip mpa binds to the cmip stack. The default is specified by using the cmiptool in /opt/SUNWconn/cmip/sbin/cmiptool and is installed by Solstice EM as Prs.

TABLE 12-2 *em_cmip Parameters (Continued)*

Command Line Parameter	Corresponding Environment Variable	Description
-t <tset>	EM_CMIP_MPA_TSEL <tset>	Allows the transport selector address to be specified when the cmip mpa binds to the cmip stack. The default is specified by using the cmiptool in /opt/SUNWconn/cmip/sbin/cmiptool and is installed by Solstice EM as CMIP.
N/A	EM_CMIP_MPA_PLATFORM_CONNECT_TIMEOUT	The timeout setting for CMIP attempts to connect to the MIS. The default is 20 seconds.
N/A	EM_CMIP_MPA_RECONNECT_POLL_INTERVAL	The poll interval setting for CMIP MPA attempts to reconnect to the MIS after it disconnects. This is typically used when the CMIP MPA is installed on a remote machine. The default is 15 seconds.
N/A	EM_CMIP_MPA_APPLICATION_CONTEXT	The application context in OID format.

TABLE 12-3 *em_oct Parameters*

Command Line Parameter	Corresponding Environment Variable	Description
-cmip		Configures a CMIP object. Replaces em_cmipconfig.
-id <id>...		Specifies topology IDs. Multiple IDs are delimited by a space.
-link <id1> <id2>		Create a link between <id1> and <id2>.
-name <name>...		Specifies the name of an object. Multiple names are delimited by a space.
-parent <parent_id>		Specifies the parent of the object you want to create.
-snmp		Configures an SNMP object.
-type <type>		Specifies the topology type of the object you want to create.

12.8.4 Sample Program to Retrieve Runtime Parameters

The sample program `get.cc` is an example of how to retrieve the CMIP MPA's runtime parameters from the MIT. The `get.cc` program is provided with Solstice EM and is located in the `$(EM_HOME)/src/pmi_hi/` directory. The following code example shows the usage and output of the `get.cc` sample program.

CODE EXAMPLE 12-1 Usage and Output of `get.cc` sample program

```
tekgr1:josie(173) ./get -dn '/systemId=name:"tekgr1"/
auxServerType="cmip_mpa"/auxServerId="tekgr1:5557"' -
object_class
'cmipMpaServer'
The object name is
      /systemId=name:"tekgr1"/auxServerType="cmip_mpa"/
auxServerId="tekgr1:5557"
Attribute                               Value
-----                               -
agentRoleAccessControl                  off

Sun Proprietary/Confidential: Internal Use Only CMIPMPASDD
applicationContext                      2.9.0.0.2
applicationEntityTitle                  objectIdentifier : { 1 1 1 1 }
applicationProcessName                  1.1.1.1:-1:-1:-1
associationTimeout                      300
auxServerId                            tekgr1:5557
debugOptionsList { }
destinationProtocol                    cmip
fallbackUserId
idleTimeout                            300
logFile                                em_cmip.log
maximumRetryCount                      1
misHost                                tekgr1
mpaPort                                5557
networkSAP                             81924bed
presentationSelector                   dflt
requestTimeout                         120
retryDelay                             30
sessionSelector                        Prs
sourceProtocol                         lpp
transportSelector                      CMIP
administrativeState                    unlocked
nameBinding "EM AUXILIARY SERVER":cmipMpaServer-
auxServerConfigContainer
objectClass globalForm : "EM AUXILIARY SERVER":cmipMpaServer
operationalState                      enabled
tekgr1:josie(174)
```

12.9 Configuring Multiple MPAs on One System

The following is an example of configuring multiple MPAs on a single system.

1. **Set the PSEL, SSEL, TSEL, and NSAP environment variables using the `setenv` command:**

```
host# setenv EM_CMIP_MPA_PSEL <rfc1>
host# setenv EM_CMIP_MPA_SSEL Prs
host# setenv EM_CMIP_MPA_TSEL CMIP
host# setenv EM_CMIP_MPA_NSAP 0x8192b92b
host# setenv EM_CMIP_MPA_PLATFORM_CONNECT_TIMEOUT 30
host# setenv EM_CMIP_MPA_RECONNECT_POLL_INTERVAL 20
host# setenv EM_CMIP_MPA_APPLICATION_CONTEXT 0.0.13.3100.1.0.1
host# setenv EM_CMIP_MPA_SEND_ALARM_EARLY Yes
```

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.**

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 EM_CMIP_MPA_PSEL rfc1
host# setenv EM_CMIP_MPA_SSEL Prs
host# setenv EM_CMIP_MPA_TSEL CMIP
host# setenv EM_CMIP_MPA_NSAP 0x8192b92b
host# setenv EM_CMIP_MPA_DEFAULT_PORT 5558
```

4. Start MPA #2:

```
host# /etc/rc2.d/S98cmipmpa start
```

In this example, MPA #2 uses the non-default port of 5558 and is bound to:

```
PAddr(rfc1,Prs,CMIP,<0xIPlocalhostinHex>)
```

5. Configure Agent(1) and (2) using `em_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 MPAs are running on same system and MIS forwards requests to Agent1 and Agent2 using MPA1 and MPA2 respectively.

Configuring CMIP MPA Overload

This chapter describes how to detect overload conditions for Common Management Information Protocol (CMIP) Management Protocol Adapter (MPA), and what action(s) must be taken when overload conditions exist.

This chapter describes the following topics:

- Section 13.1 “Understanding CMIP MPA Overload” on page 13-1
- Section 13.2 “Configuration Parameters” on page 13-2
- Section 13.3 “Management Information Tree of Overload Control Objects” on page 13-5
- Section 13.4 “GDMO Classes” on page 13-6

13.1 Understanding CMIP MPA Overload

Occasionally the *Solstice Enterprise Manager* (Solstice EM) network management system may experience network overload conditions caused by event storming. Under such conditions, the process size grows very fast and the CPU usage would be very high eventually causing the system to hang or crash. It is very important to ensure that Solstice EM handles this condition and that no Solstice EM platform component crashes.

In Solstice EM releases 3.0 and above, the CMIP MPA implements the overload control mechanism to handle this situation. The CMIP is configured in the following way:

- If the threshold of a particular severity level for the CMIP agent is crossed, the `mpaOverloadAlarm` notification will be emitted.
- If the overload instruction is configured to `abortAssociations`, the CMIP MPA automatically sets the administrative state of all the CMIP agents, whose threshold is crossed, to locked.

- Upon receiving the `attributeValueChange` event of the administrative state of the CMIP agent, the CMIP MPA issues an `ABORT_ASSOCIATION` request to these agents.

The threshold levels are used to indicate the level of the overload. Following are the threshold levels used:

- Critical
- Major
- Minor
- Warning
- Cleared

13.2 Configuration Parameters

The overload control feature in the CMIP MPA requires configuration parameters to be stored in the `EM-config` file in `/var/opt/SUNWconn/em/conf` and GDMO objects (`overload.gdmo` and `overload.asn1`). These parameters in `EM-config` file are used for the initial value if the `mpaOverloadController` object does not exist in the MIS. The parameters in `EM-config` can be dynamically changed through PMI or `em_obed`.

Once the `mpaOverloadController` object is created, any modification to the `EM-config` file will have no effect unless you delete the existing overload object and bring up the `em_cmip` again.

The CMIP MPA will implement this overload control mechanism through the following parameters:

- Overload Control Parameter
- Overload Notification Parameter
- Overload Threshold Parameter
- Minimum Threshold Parameter
- Overload Instruction Parameter
- Poll Interval Parameter

Note – The CMIP agents must be configured in the MIS before enabling the overload control feature.

13.2.1 Overload Control Parameter

This parameter determines whether the overload control feature is enabled or needful disabled. The valid values are `ENABLED` and `DISABLED`. The default is `DISABLED`.

```
EM_MPA_OVERLOAD_CONTROL: DISABLED
```

13.2.2 Overload Notification Parameter

The following parameters determine whether or not the notification is emitted when the threshold is crossed. In addition, whether or not the notification is emitted also depends on the overload state. Notification will not be emitted if the previous and current overload states are same. This will minimize the traffic between the CMIP MPA and the EDS. The valid values are `ENABLED` and `DISABLED`. The default is `DISABLED`.

- `EM_MPA_CRITICAL_NOTIFICATION: DISABLED`
- `EM_MPA_MAJOR_NOTIFICATION: DISABLED`
- `EM_MPA_MINOR_NOTIFICATION: DISABLED`
- `EM_MPA_WARNING_NOTIFICATION: DISABLED`
- `EM_MPA_CLEAR_NOTIFICATION: DISABLED`

13.2.3 Overload Threshold Parameter

The following parameters indicate the threshold value of overload conditions in the units of events per second. The valid value is a positive integer. The default value for these parameters is 0. The threshold parameter will not be checked if its value is equal to 0.

```
PER_MPA
```

- `EM_MPA_CRITICAL_THRESHOLD_RATE_PER_MPA: 0`
- `EM_MPA_MAJOR_THRESHOLD_RATE_PER_MPA: 0`
- `EM_MPA_MINOR_THRESHOLD_RATE_PER_MPA: 0`
- `EM_MPA_WARNING_THRESHOLD_RATE_PER_MPA: 0`

```
PER_AGENT
```

- `EM_MPA_CRITICAL_THRESHOLD_RATE_PER_AGENT: 0`
- `EM_MPA_MAJOR_THRESHOLD_RATE_PER_AGENT: 0`
- `EM_MPA_MINOR_THRESHOLD_RATE_PER_AGENT: 0`
- `EM_MPA_WARNING_THRESHOLD_RATE_PER_AGENT: 0`

13.2.4 Minimum Threshold Parameter

The minimum threshold parameter is a low water mark set up for each threshold to avoid ping-pong between threshold levels. The default and the minimum value is 5.

13.2.5 Overload Instruction Parameter

This parameter indicates what kind of action is taken in case the level of a particular threshold is crossed. The supported actions are, do nothing or abort associations of the agent. The valid values are `DO_NOTHING` and `ABORT_ASSOCIATIONS`. The default value is `DO_NOTHING`.

When the CMIP MPA reaches a particular threshold level and the action is `ABORT_ASSOCIATIONS`, the CMIP MPA will abort all the open associations to the CMIP agents.

- `EM_MPA_CRITICAL_OVERLOAD_INSTRUCTION: DO_NOTHING`
- `EM_MPA_MAJOR_OVERLOAD_INSTRUCTION: DO_NOTHING`
- `EM_MPA_MINOR_OVERLOAD_INSTRUCTION: DO_NOTHING`
- `EM_MPA_WARNING_OVERLOAD_INSTRUCTION: DO_NOTHING`

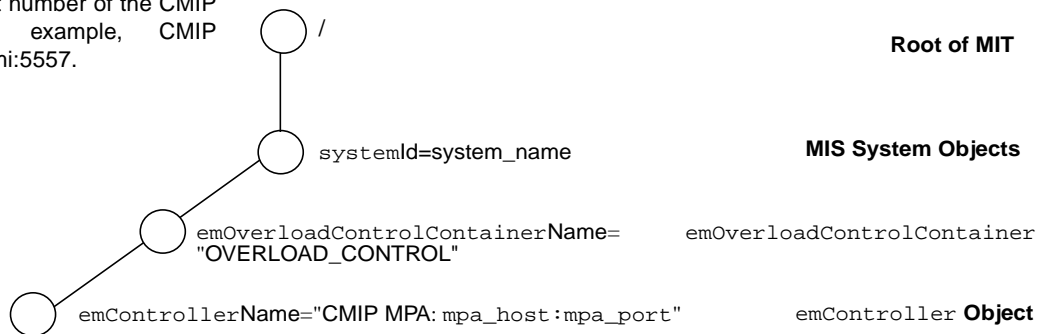
13.2.6 Poll Interval Parameter

The poll interval parameter `EM_MPA_OVERLOAD_POLL_INTERVAL: 30` indicates how frequently the CMIP MPA checks overload conditions (in seconds). The minimum value is 30 seconds. If the value is less than 30 seconds, it defaults to 30 seconds.

13.3 Management Information Tree of Overload Control Objects

The following figure illustrates the Management Information Tree (MIT) of Overload Control Objects.

`mpa_host` is the CMIP MPA hostname, whereas `mpa_port` is the port number of the CMIP MPA—for example, CMIP MPA:miami:5557.



Note: All attributes of `mpaOverloadController` named by `emControllerName` are maintained by the MIS.

FIGURE 13-1 Management Information Tree of Overload Control Objects

13.4 GDMO Classes

13.4.1 Mapping Between Attributes of the GDMO Classes and Configuration Parameters

TABLE 13-1 GDMO Mapping

Configuration Parameters	GDMO attributes
EM_MPA_OVERLOAD_CONTROL	administrativeState
EM_MPA_OVERLOAD_POLL_INTERVAL	pollInterval
EM_MPA_MINIMUM_THRESHOLD	minimumThreshold
EM_MPA_CRITICAL_NOTIFICATION, EM_MPA_MAJOR_NOTIFICATION, EM_MPA_MINOR_NOTIFICATION, EM_MPA_WARNING_NOTIFICATION	notificationEnabledStatus
EM_MPA_CRITICAL_THRESHOLD_RATE_PER_MPA, EM_MPA_MAJOR_THRESHOLD_RATE_PER_MPA, EM_MPA_MINOR_THRESHOLD_RATE_PER_MPA, EM_MPA_WARNING_THRESHOLD_RATE_PER_MPA	thresholdRatePerMPA
EM_MPA_CRITICAL_THRESHOLD_RATE_PER_AGENT, EM_MPA_MAJOR_THRESHOLD_RATE_PER_AGENT, EM_MPA_MINOR_THRESHOLD_RATE_PER_AGENT, EM_MPA_WARNING_THRESHOLD_RATE_PER_AGENT	thresholdRatePerAgent
EM_MPA_CRITICAL_OVERLOAD_INSTRUCTION, EM_MPA_MAJOR_OVERLOAD_INSTRUCTION, EM_MPA_MINOR_OVERLOAD_INSTRUCTION, EM_MPA_WARNING_OVERLOAD_INSTRUCTION	overloadInstruction

13.4.2 emOverloadControlContainer Class

This GDMO class serves as a container for objects used to implement the overload control feature. The attribute value of `overloadControlContainerName` is set to `OVERLOAD_CONTROL` when the object is created during Solstice EM installation.

13.4.3 emOverloadController Class

This GDMO class is used to represent the Solstice EM server process which has defined the overload control objects. The CMIP MPA will be able to listen for the events of the creation/deletion/attribute value change of this class.

13.4.4 mpaOverloadController Class

This GDMO class is derived from `emOverloadController` GDMO class, used to represent the MPA. This defines the threshold rate per agent attribute managed by the MIS. The class also generates notifications called `agentOverloadAlarm`, `mpaOverloadAlarm`, and `abortAssocNotification`. Each CMIP MPA will have its own instance of `mpaOverloadController` object class named by “CMIP MPA:mpa_host:mpa_port”. This class defines `mpaStateInfo` action used to query CMIP MPA to retrieve the overload state information for the CMIP MPA as well as for the CMIP agents. This action is designed for performance since the CMIP MPA keeps track of the overload state information in C++ class and the CMIP MPA does not maintain the `mpaOverloadController` objects.

mpaOverloadAlarm Notification

This alarm notification is sent by the instance of `mpaOverloadController` object class. It is used to report an alarm of overload condition when the threshold of the CMIP MPA is crossed. The alarm information consists of the alarm information of the CMIP MPA as well as of each CMIP agent. This notification is mapped to `mpaOverloadAlarmRecord` log record.

agentOverloadAlarm Notification

This alarm notification is sent by the instance of `mpaOverloadController` object class. It is used to report an alarm of overload condition when the threshold of the individual agent is crossed. The alarm information consists of the alarm information of each CMIP agent. This notification is mapped to `agentOverloadAlarmRecord` log record.

abortAssocNotification Notification

This notification is sent when the associations of the CMIP agent have been aborted due to the threshold of the CMIP MPA, or of the CMIP agent is crossed and whose overload instruction is `ABORT_ASSOCIATIONS`. The modification information includes the CMIP agent's MOI and text stating what action has been taken.

13.4.5 Overload Sample Programs

This section contains the sample programs for the overload control mechanism implemented in the CMIP MPA. See the following table.

TABLE 13-2 Sample Programs Description

<code>overload_get.cc</code>	This program will get all or a single attribute of <code>mpaOverloadController</code> object.
<code>overload_set.cc</code>	This program will set the attribute value of <code>mpaOverloadController</code> object.
<code>overload_action.cc</code>	This program will send action request (<code>mpaStateInfo</code>) to the CMIP MPA to retrieve the overload state of the CMIP MPA as well as the remote CMIP agent.
<code>overload_alarm.cc</code>	This program will listen for the <code>mpaOverloadAlarm</code> and <code>agentOverloadAlarmaction</code> event generated by the CMIP MPA when the threshold is crossed. If the action parameter of the threshold level is specified, it will set the administrative state of the CMIP agent to locked. The CMIP MPA will receive this attribute change event and trigger the RELEASE ASSOCIATIONS to be sent to the CMIP agent.
<code>get_agent_admin_state.cc</code>	This program will get the administrative state of the specified CMIP agent.
<code>set_agent_admin_state.cc</code>	This program will set the administrative state of the specified CMIP agent.

13.4.5.1 get_agent_admin_state

Purpose: Get the administrative state of the CMIP agent Id and print it.

Syntax: `get_agent_admin_state -agent <CMIP agent Id> -o <object class>\`
`[-host <host>] [-help]`

where *<CMIP agent Id>* is the CMIP agent Id.

<object class> is the object class of the CMIP agent.

<host> is the MIS host

CODE EXAMPLE 13-1 Sample Syntax for get_agent_admin_state

```
//      ./get_agent_admin_state -a 'thomaseng' -o cmipAgent
//      ./get_agent_admin_state -agent 'thomaseng' -o cmipAgent -host emperf
//
//

#include <limits.h>  // LINE_MAX
#include <netdb.h>
#include <sys/systeminfo.h>
#include <rw/cstring.h>
#include <hi.hh>
#include <installation.hh>  // GETENV

const char *agent_rdn = "agentTableType='CMIP'/agentId=id:";

void usage()
{
    cout << "\nUsage:" ;
    cout << " ./get_agent_admin_state -agent <cmip agent id> " ;
    cout << " -o <object class>" << endl;
    cout << " [-host <host>] [-help]" << endl;
    cout << " -agent <cmip agent id> : CMIP agent Id (e.g.'emperf') " << endl ;
    cout << " -o <object class> : object class of the cmip agent" << endl;
    cout << " -host <host> : host to connect to" << endl;
    cout << " -help : print this message and exit" << endl;
}
```

CODE EXAMPLE 13-2 Main Program for get_agent_admin_state

```
int
main(int argc, char **argv)
{
    Platform plat(duEM);

    RWCString dn;
    RWCString agent_name;
    RWCString class_name;
    RWCString attribute_name = "administrativeState";
    RWCString host;

    // Get the host name.

    char * env_host = GETENV("EM_SERVER");

    if (!env_host)
    {
        char system_host[MAXHOSTNAMELEN + 1];

        sysinfo(SI_HOSTNAME, system_host, MAXHOSTNAMELEN);

        host = system_host;
    }
    else
        host = env_host;

    // Parse the cmd line for options.

    Boolean o_specified = FALSE;
    Boolean a_specified = FALSE;

    argv++;
    argc--;

    // get command line inputs
    void get_command_line_options(int,char **,Boolean *,Boolean
*,RWCString&,RWCString&,RWCString&);
    get_command_line_options(argc,argv,&a_specified,&o_specified,host,agent_name,
class_name);

    // Check if all params have been properly provided
```

```

int

    if (!a_specified || !o_specified) {
        usage();
        exit(1);
    }

    // construct dn
    dn = agent_rdn +
        WCString("") + agent_name + WCString("");

    cout << "\n" << endl;
    cout << "===== " <<
endl;
    cout << "The object name is " << endl;
    cout << dn << endl;
    cout << "===== " <<
endl;

    // Connect to platform.
    cout << "Connecting to ... " << host << endl;
    if (!plat.connect((char *) (const char *) host,
        "em_sample"))
    {

        cout << "Failed to Connect to " << host << endl;
        cout << plat.get_error_string() << endl;

        exit(2);
    }
    cout << "Connected." << endl;

    // Declare cmipAgent/cmipAgentEntity image.
    Image im = Image((char *) (const char *) dn,
        (char *) (const char *) class_name);

    // Could not boot image
    if (!im.boot())
    {
        cout << "Failed to boot " << dn << endl;
        cout << im.get_error_string() << endl;
        exit(3);
    }

    // print the header

```

CODE EXAMPLE 13-2 Main Program for get_agent_admin_state (Continued)

```
int
    cout << "\nAttribute      Value";
    cout << "\n-----      ----" << endl;

    // Get attribute value and print it.

    cout << attribute_name;
    cout << " "
        << im.get_str((char *) (const char *) attribute_name).chp()
        << endl;
    exit(0);
}

/*****

Method: substring

General Description:
Returns true if all of sub_string matches the main_string from the
beginning of main_string.

e.g.  there is a match if
      main_string is "help"
      substring is  "he"

No match when:
      main_string is  "he"
      sub_string is   "help"
or
      main_string is "host"
      sub_string is  "he"

Arguments:

Return Value:

Algorithm:

Usage:

WARNING:  An empty sub_string ("") will return TRUE.

Exceptions:

*****/
```


CODE EXAMPLE 13-2 Main Program for get_agent_admin_state (Continued)

```
int
Boolean substring( const char * main_string, const char * sub_string)
{
    while (*sub_string != 0)
    {
        if (*sub_string++ != *main_string++)
            return FALSE;
    }

    return TRUE;
}

//
// Get command line inputs.
//
void
get_command_line_options(int argc, char **argv,
                        Boolean *a_specified, Boolean *o_specified,
                        RWCString &host,
                        RWCString &agent_name, RWCString &class_name)
{
    while (argc > 0 && argv[0][0] == '-')
    {
        switch (argv[0][1])
        {
            case 'a':
            {
                if (!substring("agent", &(argv[0][1])))
                {
                    usage();
                    exit(1);
                }
                argc--;
                if (!argc)
                {
                    usage();
                    exit(1);
                }
                argv++;

                agent_name = argv[0];

                *a_specified = TRUE;
            }
            break;
            case 'o':
```

CODE EXAMPLE 13-2 Main Program for get_agent_admin_state (Continued)

```
int
    if (!substring("object_class", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;

    class_name = argv[0];

    *o_specified = TRUE;

    break;

    case 'h':
        if (strlen(&(argv[0][1])) < 2)
        {
            // can not distinguish between -help and -host
            usage();
            exit(1);
        }
        if (substring("help", &(argv[0][1])))
        {
            usage();
            exit(0);
        }

        if (!substring("host", &(argv[0][1])))
        {
            usage();
            exit(1);
        }
        argc--;
        if (!argc)
        {
            usage();
            exit(1);
        }
        argv++;
        host = argv[0];
```

CODE EXAMPLE 13-2 Main Program for get_agent_admin_state (Continued)

```
int
    break;

    case '?':
        usage();

        exit(1);
    }
    argv++;
    argc--;
}
}
```

13.4.5.2 overload_action

Purpose: This program demonstrates how to invoke mpaOverloadController object action mpaStateInfo. GDMO class mpaOverloadController supports action mpaStateInfo which retrieves the overload state of the CMIP MPA and the remote agents.

Action: mpaStateInfo

Usage: overload_action -n <emControllerName> -l <action parameter> [-host <host>] [-help]

where <emControllerName> is the emControllerName in the form of "CMIP MPA:mpa_host:mpa_port" <action parameter> is the MpaStateInfoRequest asn1 syntax of mpaStateInfo action <host> is the host connect to (see overload.asn1).

CODE EXAMPLE 13-3 Sample for overload_action

```
// EXAmple:
//
// %overload_action -n 'CMIP MPA:thomaseng:5557' -l 'default : NULL'
// %overload_action -n 'CMIP MPA:thomaseng:5557' -l 'agentId : { { { systemId,
"thomaseng" } }, { { agentTableType, "CMIP" } }, { { agentId, "miami" } } }'
// %overload_action -n 'CMIP MPA:thomaseng:5557' -l 'agentId :
localDistinguishedName : { { { agentId, "miami" } } }'
//
//

#include <netdb.h>
#include <sys/systeminfo.h>

#include <hi.hh>
```

CODE EXAMPLE 13-3 Sample for overload_action (Continued)

```
// EXAmple:
#include <iostream.h>
#include <rw/cstring.h>
#include <unistd.h>

const char *overload_container_name =
"overloadControlContainerName='OVERLOAD_CONTROL'/emControllerName=";

void usage();
```

CODE EXAMPLE 13-4 Main Program for overload_action

```
int
main(int argc, char **argv)
{
    RWCString dn;
    RWCString action_name = "mpaStateInfo";
    RWCString action_para;
    RWCString mpa_name;

    // Get the host name
    char *host = getenv("EM_SERVER");
    if (!host) {
        host = new char[MAXHOSTNAMELEN+1];
        sysinfo(SI_HOSTNAME, host, MAXHOSTNAMELEN-1);
    }

    // Parse the cmd line for options.

    Boolean n_specified = FALSE;
    Boolean l_specified = FALSE;

    argv++;
    argc--;

    // get command line inputs
    void get_command_line_options(int,char**,Boolean*,Boolean*,char
*,RWCString&,RWCString&);
    get_command_line_options(argc, argv, &n_specified,&l_specified,
        host, mpa_name, action_para);

    // Check if all params have been properly provided

    if (!n_specified || !l_specified) {
```

CODE EXAMPLE 13-4 Main Program for overload_action (Continued)

```
int
    usage();
    exit(1);
}

cout << endl;
cout << "hostname = " << host << endl;
cout << "mpaname = " << mpa_name << endl;
cout << "action name = " << action_name << endl;
cout << "action parameter = " << action_para << endl;
cout << endl << endl;

// Set up connect to MIS
Platform plat = Platform(duEM);
if (plat.get_error_type() != PMI_SUCCESS) {
    cout << plat.get_error_string() << endl;
    exit(2);
}

cout << "Connecting to ... " << host << endl;

if (!plat.connect(host, "em_sample")) {
    cout << "Failed to connect to " << host << endl;
    cout << plat.get_error_string() << endl;
    exit(4);
}
cout << "Connected." << host << endl;

// Construct dn
dn = overload_container_name +
    RWCString ("") + mpa_name + RWCString("");

Image ov_image = Image((char *)(const char *)dn);
if (ov_image.get_error_type() != PMI_SUCCESS) {
    cout << ov_image.get_error_string() << endl;
    exit(2);
}

// Boot mpaOverloadController object.
if (!ov_image.boot()) {
    cout << ov_image.get_error_string() << endl;
    exit(2);
}

// Use action name to get input syntax.
Syntax syn_input = ov_image.get_param_syntax((char *)(const char
*)action_name);
```

CODE EXAMPLE 13-4 Main Program for overload_action (Continued)

```
int
if (syn_input.get_error_type() != PMI_SUCCESS) {
    cout << syn_input.get_error_string() << endl;
    exit(2);
}

// Use action name to get result syntax.
Syntax syn_result = ov_image.get_result_syntax((char *)(const char
*)action_name);
if (syn_result.get_error_type() != PMI_SUCCESS) {
    cout << syn_result.get_error_string() << endl;
    exit(2);
}

// Print the syntaxes.
cout << "Input Syntax is " << syn_input.get().chp();
cout << endl;
cout << "Result Syntax is " << syn_result.get().chp();
cout << endl;

// Invoke the action and get the result data.
DU action_data = ov_image.call((char *)(const char *)action_name, (char
*)(const char *)action_para);

// Use input syntax and action parameter to get input morf.
Morf morf_input(syn_input, (char *)(const char *)action_para);
if (morf_input.get_error_type() != PMI_SUCCESS) {
    cout << morf_input.get_error_string() << endl;
    exit(2);
}

cout << "Input Morf--> " << morf_input.get().chp() << endl;

// User input morf and action name to get the result.
Morf morf_result = ov_image.call_raw((char *)(const char *)action_name,
morf_input);
if (morf_result.get_error_type() != PMI_SUCCESS) {
    cout << morf_result.get_error_string() << endl;
    exit(2);
}

// Print the result.
cout << "Result --> " << morf_result.get().chp() << endl;

return 0;
}
```

CODE EXAMPLE 13-4 Main Program for overload_action (Continued)

```
int

void
usage()
{
    cout << "\nUsage: ";
    cout << "overload_action -n <emControllerName> -l <action parameter> [-host
<host>] [-help]" << endl;
    cout << "    -n <emControllerName> : emControllerName in the form of"<<endl ;
    cout << "                                'CMIP MPA:mpa_host:mpa_port'"<< endl;
    cout << "    -l <action parameter> : MpaStateInfoRequest asnl syntax of "
<< endl;
    cout << "                                mpaStateInfo action" << endl;
    cout << "    -host <host> : host to connect to " << endl;
    cout << "    -help : print this message and exit" << endl;
}
```

CODE EXAMPLE 13-5 Exceptions

```
Boolean substring(
    const char * main_string,
    const char * sub_string
)
{
    while (*sub_string != 0)
    {
        if (*sub_string++ != *main_string++)
            return FALSE;
    }

    return TRUE;
}

//
// Get command line inputs
//
void
get_command_line_options(int argc, char **argv,
                        Boolean *n_specified, Boolean *l_specified,
                        char *host,
                        RWCString &mpa_name, RWCString &action_para)
{
    while (argc > 0 && argv[0][0] == '-')
    {
        switch (argv[0][1])
        {
            case 'n':
```

CODE EXAMPLE 13-5 Exceptions *(Continued)*

```
Boolean substring(
{
    if (!substring("n", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;

    mpa_name = argv[0];

    *n_specified = TRUE;
}
break;
case 'l':

    if (!substring("l", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;

    action_para = argv[0];

    *l_specified = TRUE;
    break;
case 'h':
    if (strlen(&(argv[0][1])) < 2)
    {
        // can not distinguish between -help and -host
        usage();
        exit(1);
    }
    if (substring("help", &(argv[0][1])))
```


CODE EXAMPLE 13-5 Exceptions *(Continued)*

```
Boolean substring(
{
    usage();
    exit(0);
}

if (!substring("host", &(argv[0][1])))
{
    usage();
    exit(1);
}
argc--;
if (!argc)
{
    usage();
    exit(1);
}
argv++;
strcpy(host,argv[0]);

break;

case '?':
    usage();

    exit(1);
}
argv++;
argc--;
}
```

13.4.5.3 overload_alarm

Purpose: Listen for `mpaOverloadAlarm` and `agentOverloadAlarm` events which are generated from the CMIP MPA if the overload control mechanism is enabled and the threshold is crossed. If the overload instruction of particular threshold level is `releaseAssociations`, update the `administrativeState` of the CMIP agent to

locked. The CMIP MPA will listen for this attribute change event and sends the RELEASE_ASSOCIATION REQUEST to the agent. If the overload instruction of particular threshold level is doNothing, no action will be taken.

CODE EXAMPLE 13-6 Syntax for overload_alarm

```
overload_alarm [-cr_action <action>] [-mj_action <action>]
//                                     [-mn_action <action>] [-wn_action <action>]
//                                     [-host <MIS host>] [-help]
//
//      where xx_action - the threshold level and
//      action         - the overload instruction. The valid values
//                        are doNothing/releaseAssociations.
//      host           - MIS host
//      help           - print command line options and exit
//      (refer to overload.gdmo/overload.asn1)
//
// Notes: 1. This program never exits; it enters an infinite listening loop.
//         Use Control-C to terminate it.
//         2. If no argument is supplied, no overload action will be taken.
//
// Examples:
// %overload_alarm -cr_action releaseAssociations -wn_action doNothing
// %overload_alarm -mj_action releaseAssociations -mn_action doNothing
// %overload_alarm
//

#include <netdb.h>
#include <sys/systeminfo.h>

#include <hi.hh>
#include <message.hh>

const Oid mpaOverloadAlarmOid = Oid("1.3.6.1.4.1.42.2.2.11.10.1");
const Oid agentOverloadAlarmOid = Oid("1.3.6.1.4.1.42.2.2.11.10.2");

void raw_cb( Ptr, Ptr calldata);

enum AGENT_TYPE {mpa, cmip_agent};
enum SEVERITY {critical, major, minor, warning, cleared};
enum ADMIN_STATE {unlocked, locked};

const char *sev_str[] = {"Critical", "Major", "Minor", "Warning", "Cleared",
"Unknown"};
const DU mpaOverloadAlarmDU = "mpaOverloadAlarm";
const DU agentOverloadAlarmDU = "agentOverloadAlarm";
const DU AgentTableDU = "agentCmip";
```

CODE EXAMPLE 13-6 Syntax for overload_alarm (Continued)

```

overload_alarm [-cr_action <action>] [-mj_action <action>]
const DU ExtAgentTableDU = "agentCmpEntity";

char *cr_action="doNothing";
char *mj_action="doNothing";
char *mn_action="doNothing";
char *wn_action="doNothing";
Boolean cr_action_specified=FALSE, mj_action_specified=FALSE;
Boolean mn_action_specified=FALSE, wn_action_specified=FALSE;

void usage()
{
    cout << "\nUsage: " ;
    cout << "./overload_alarm [-cr_action <action>] [-mj_action <action>]
[mn_action <action>] [wn_action <action>] [-host <host>] [-help] " << endl;
    cout << "    -cr_action <action> : overload instruction for critical threshold
" << endl;
    cout << "                                (doNothing/releaseAssociations)" << endl;
    cout << "    -mj_action <action> : overload instruction for major threshold
" << endl;
    cout << "                                (doNothing/releaseAssociations)" << endl;
    cout << "    -mn_action <action> : overload instruction for minor threshold
" << endl;
    cout << "                                (doNothing/releaseAssociations)" << endl;
    cout << "    -wn_action <action> : overload instruction for warning threshold
" << endl;
    cout << "                                (doNothing/releaseAssociations)" << endl;
    cout << "    -help                : print this message and exit" << endl;
    cout << "    -host <host>         : host to connect to" << endl;
}

```

CODE EXAMPLE 13-7 Main Program for overload_alarm

```

}
int
main(int argc, char **argv)
{
    // Setup the connection to the MIS.

    Platform plat = Platform(duEM);
    if (plat.get_error_type()!=PMI_SUCCESS) {
        cout << plat.get_error_string() << endl;
        exit(1);
    }

    // Get the host name.
}

```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```

}

char *host = getenv("EM_SERVER");
if (!host) {
    host = new char[MAXHOSTNAMELEN+1];
    sysinfo(SI_HOSTNAME, host, MAXHOSTNAMELEN-1);
}

argv++;
argc--;

// get command line inputs
void get_command_line_options(int, char**, char *);
get_command_line_options(argc,argv,host);

/*
if (argc > 0 && (!cr_action_specified || !mj_action_specified ||
                !mn_action_specified || !wn_action_specified)) {
    usage();
    exit(1);
}
*/

cout << "Connecting to ... " << host << endl;
if (!plat.connect(host, "em_sample")) {
    cout << "Failed to connect to " << host << endl;
    cout << plat.get_error_string() << endl;
    exit(2);
}
cout << "Connected." << endl << endl;

// Only interested in this specified object class.
Array(DU)classes(1);
classes[0] = "mpaOverloadController";

Array(DU)events(2);
events[0] = mpaOverloadAlarmDU;
events[1] = agentOverloadAlarmDU;

if(!plat.replace_discriminator_classes(classes, events)) {
    cout << plat.get_error_string();
    exit(4);
}

if (!plat.when("RAW_EVENT", Callback(raw_cb, 0))) { // Say when to call
raw_cb().
    cout << plat.get_error_string() << endl;
    exit(5);
}

```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}  
}  
  
cout << "Waiting for events..... " << endl << endl;  
  
while(1) {                                // Enter the infinite listen loop.  
    dispatch_recursive(TRUE);  
}  
  
exit(0);  
}  
  
//  
// Check the overload instruction:  
// Return true if the action is releaseAssociations, otherwise, return false.  
//  
Boolean need_change_state(I32 sev)  
{  
    switch (sev-1) {  
        case critical:  
            if (cr_action_specified && !strcmp(cr_action, "releaseAssociations"))  
                return TRUE;  
            break;  
        case major:  
            if (mj_action_specified && !strcmp(mj_action, "releaseAssociations"))  
                return TRUE;  
            break;  
        case minor:  
            if (mn_action_specified && !strcmp(mn_action, "releaseAssociations"))  
                return TRUE;  
            break;  
        case warning:  
            if (wn_action_specified && !strcmp(wn_action, "releaseAssociations"))  
                return TRUE;  
            break;  
        default:  
            return FALSE;  
    }  
    return FALSE;  
}  
  
//  
// Change administrativeState of cmip agent.  
//  
void change_admin_state_of_agent(const Asn1Value oi, ADMIN_STATE state)  
{
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}

DU fdn = oi2fdn(oi);
Image agent_im = Image(fdn);
if (!agent_im.boot()) {
    cout << "Can't boot agent image" << endl;
    exit(3);
}
if (!agent_im.exists()) {
    cout << "Agent image does not exist" << endl;
    exit(3);
}
char *state_str;
if (state == unlocked)
    state_str = "unlocked";
else if (state == locked)
    state_str = "locked";
if (!agent_im.set_str("administrativeState", state_str)) {
    cout << "Failed to set administrative state of the agent ( "
        << fdn.chp() << " ) " << endl;
    exit(3);
}
if (!agent_im.store()) {
    cout << "Failed to store administrative state of the agent ( "
        << fdn.chp() << " ) " << endl;
    exit(3);
}
else {
    cout << "\n\n=====
<< endl;
    cout << "SET the administrative state to locked for agent: "<<endl;
    cout << "<" << fdn.chp() << "> " << endl;
    cout << "=====\n\n"
<< endl;
}

}

//
// Decode alarm information of each cmip agent.
//
void decode_agent_info(const Asn1Value &agent_info)
{
    Asn1Value ava, ava2, agent_moi;
    U32 num = agent_info.num_comps();
    for (int i=0; i < num; i++) {
        if (i == 0) {
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```

    }

    VTRYINV(agent_info.first_component(ava));
    VTRYINV(ava.first_component(agent_moi)); // moi of agent Id
    change_admin_state_of_agent(agent_moi, locked);
}
else if (i != 0) {
    VTRYINV(agent_info.next_component(ava, ava2));
    VTRYINV(ava2.first_component(agent_moi)); // moi of agent Id
    change_admin_state_of_agent(agent_moi, locked);
}
}
}

//
// Print out the useful alarm information such as what severity of
// alarm received, where does it come from, and the action to be taken.
//
void
print_alarm_info(SEVERITY sev, AGENT_TYPE type, Asn1Value agent_oi)
{
    // convert oi to fdn
    DU fdn = oi2fdn(agent_oi);

    // check unknown severity
    if (sev < critical || sev > cleared)
        sev = SEVERITY(5);

    cout << "\n\n===== " <<
endl;
    if (type == mpa) {
        cout << "Received <" << sev_str[sev] << "> alarm from CMIP MPA " << endl;
    }
    else { // type == cmip_agent
        cout << "Received <" << sev_str[sev] << "> alarm from CMIP Agent " << endl;
    }
    cout << "          <" << fdn.chp() << ">" << endl;

    switch (sev) {
    case critical:
        cout << "Action taken: <" << cr_action << ">" << endl;
        break;
    case major:
        cout << "Action taken: <" << mj_action << ">" << endl;
        break;
    case minor:
        cout << "Action taken: <" << mn_action << ">" << endl;
        break;
    }
}

```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}
    case warning:
        cout << "Action taken: <" << wn_action << ">" << endl;
        break;
    default:
        cout << "Action taken: <" << sev_str[sev] << ">" << endl;
        break;
    }
    cout << "=====\n\n" <<
endl;
}

//
// Decode mpaOverloadAlarm and agentOverloadAlarm.
/* Here's the asnl syntax for these alarms.

--
-- mpa overload alarm information including all agents overload alarm
-- information. The moi of the mpa is in the event message.

MpaOverloadAlarmInfo ::= SEQUENCE {
    threshold                Threshold,
    probableCause             ProbableCause,
    perceivedSeverity         EMPerceivedSeverity,
    additionalText            AdditionalText,
    agentsOverloadAlarmInfo  AgentsOverloadAlarmInfo
}
AgentsOverloadAlarmInfo ::= SET OF SEQUENCE {
    agentMOI                 AgentMOI,
    threshold                Threshold,
    probableCause             ProbableCause,
    perceivedSeverity         EMPerceivedSeverity,
    additionalText            AdditionalText
}

AgentMOI ::= ObjectInstance

--
-- agent overload alarm information; The agent moi is in the event message
--
AgentOverloadAlarmInfo ::= SEQUENCE {
    threshold                Threshold,
    probableCause             ProbableCause,
    perceivedSeverity         EMPerceivedSeverity,
    additionalText            AdditionalText
}
```


CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}
Threshold ::= INTEGER

*/

void decode_message(CurrentEvent ce)
{
    ObjReqMess *req = (ObjReqMess *)ce.get_message();
    EventReq *evt_req = (EventReq *)req;
    Asn1Value ava;
    Oid evt_oid;
    if (evt_req->event_type.decode_oid(evt_oid) != OK) {
        cout << "Can't decode event oid" << endl;
    }
    else {
        I32 sev;

        // Decode threshold
        VTRYINV(evt_req->event_info.first_component(ava));

        // Decode probableCause
        VTRYINV(evt_req->event_info.next_component(ava,ava));

        // Decode perceivedSeverity
        VTRYINV(evt_req->event_info.next_component(ava,ava));

        // mpaOverloadAlarm
        if (evt_oid == mpaOverloadAlarmOid) {

            Asn1Value agent_info;
            Boolean found = FALSE;

            VTRYINV(ava.decode_int(sev));
            if (need_change_state(sev)) {
                print_alarm_info(SEVERITY(sev-1), mpa, req->oi);
                while (!found) {
                    VTRYINV(evt_req->event_info.next_component(ava,ava));
                    if (ava.tag() == TAG_SET)
                        found = TRUE;
                }
                if (found) {
                    agent_info = ava;
                    decode_agent_info(agent_info);
                }
            }
        }
        // agentOverloadAlarm
    }
}
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}

    else if (evt_oid == agentOverloadAlarmOid) {
        VTRYINV(ava.decode_int(sev));
        if (need_change_state(sev)) {
            print_alarm_info(SEVERITY(sev-1), cmip_agent, req->oi);
            change_admin_state_of_agent(req->oi, locked);
        }
    }
}

//
// Define a function to do something with an event notification.
//
void
raw_cb(Ptr, Ptr calldata)
{
    //
    // Print interesting things about the event.
    //
    CurrentEvent ce(calldata);
    DU tmp;

    cout << "***** RAW_EVENT received *****" << endl;

    tmp=ce.get_event();
    if (ce.get_error_type() != PMI_SUCCESS) {
        cout << ce.get_error_string() << endl;
        return;
    } else {
        cout << "EVENT = " << tmp.chp() << endl << endl;
    }

    // Decode event message if action specified
    if (cr_action_specified || mj_action_specified ||
        mn_action_specified || wn_action_specified)
        decode_message(ce);

    return;
}

/*****
Method:substring

General Description:
Returns true if all of sub_string matches the main_string from the
beginning of main_string.
*****/
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}

e.g.  there is a match if
      main_string  is "help"
      substring is  "he"

No match when:
      main_string is  "he"
      sub_string is  "help"
or
      main_string is "host"
      sub_string is  "he"

Arguments:

Return Value:

Algorithm:

Usage:

WARNING:  An empty sub_string ("") will return TRUE.

Exceptions:

*****/

Boolean substring(
    const char * main_string,
    const char * sub_string
)
{
    while (*sub_string != 0)
    {
        if (*sub_string++ != *main_string++)
            return FALSE;
    }

    return TRUE;
}

//
// Get command line inputs.
//
void
get_command_line_options(int argc, char **argv, char *host)
{
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}
while (argc > 0 && argv[0][0] == '-')
{
    switch (argv[0][1])
    {
        case 'c':
        {
            if (!substring("cr_action", &(argv[0][1])))
            {
                usage();
                exit(1);
            }
            argc--;
            if (!argc)
            {
                usage();
                exit(1);
            }
            argv++;

            cr_action = argv[0];

            cr_action_specified = TRUE;
        }
        break;
        case 'w':

            if (!substring("wn_action", &(argv[0][1])))
            {
                usage();
                exit(1);
            }
            argc--;
            if (!argc)
            {
                usage();
                exit(1);
            }
            argv++;

            wn_action = argv[0];

            wn_action_specified = TRUE;

            break;

        case 'm':
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}

    if (strlen(&(argv[0][1])) < 2)
    {
        // can not distinguish between -mj_action and -mn_action
        usage();
        exit(1);
    }
    if (substring("mj_action", &(argv[0][1])))
    {
        mj_action_specified = TRUE;
    }

    else if (!substring("mn_action", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    else
        mn_action_specified = TRUE;

    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;
    if (mj_action_specified)
        mj_action = argv[0];
    else if (mn_action_specified)
        mn_action = argv[0];

    break;
case 'h':
    if (strlen(&(argv[0][1])) < 2)
    {
        // can not distinguish between -help and -host
        usage();
        exit(1);
    }
    if (substring("help", &(argv[0][1])))
    {
        usage();
        exit(0);
    }

    if (!substring("host", &(argv[0][1])))
```

CODE EXAMPLE 13-7 Main Program for overload_alarm (Continued)

```
}

    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;
    strcpy(host,argv[0]);

    break;

case '?':
    usage();
    exit(1);
default :
    usage();
    exit(1);
}
argv++;
argc--;
}
}
```

13.4.5.4 overload_get

Purpose: Get and print attributes of the mpaOverloadController object.

CODE EXAMPLE 13-8 Syntax for overload_get

```
overload_get -n <emControllerName> [ -attribute <attribute>] \
//          [-host <hostname>] [-help]
//
//      where <n> is emControllerName of the CMIP MPA.
//      <attribute> is the attribute of the object
//      <host> is the MIS host
//      <help> is to print the command line options
//
// Note:
//      User MUST specify emControllerName in the form of
//      "CMIP MPA:mpa_host:mpa_port".
//      If no attribute is specified, get and print all attributes of
```

CODE EXAMPLE 13-8 Syntax for overload_get (Continued)

```

overload_get -n <emControllerName> [ -attribute <attribute>] \
//      mpaOverloadController object of specified emControllerName.
//
// Example 1: Get and print all attributes of the mpaOverloadController
//      object named.
//
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'emControllerName'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'minimumThreshold'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a
'notificationEnabledStatus'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'overloadInstruction'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'overloadPollInterval'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'thresholdRatePerAgent'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'thresholdRatePerMPA'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'administrativeState'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'nameBinding'
//      ./overload_get -n 'CMIP MPA:thomaseng:5557' -a 'objectClass'
//
// Example 2: Get and print all attributes.
//
//      ./overload_get -n 'CMIP MPA:thomaseng:5557'
//

#include <limits.h> // LINE_MAX
#include <netdb.h>
#include <sys/systeminfo.h>
#include <rw/cstring.h>
#include <hi.hh>
#include <installation.hh> // GETENV

const char *overload_container_name =
"overloadControlContainerName='OVERLOAD_CONTROL'/emControllerName=";

void usage()
{
    cout << " ./overload_get -n <emControllerName in 'CMIP
MPA:mpa_host:mpa_port'> ";
    cout << "[ -attribute <attribute> ] [-host <host>] [-help]" << endl;
    cout << "      -n <emControllerName> : emControllerName in the form of " <<
endl;
    cout << "                                'CMIP MPA:mpa_host:mpa_port' " << endl;
    cout << "      -attribute <attribute> : attribute name" << endl;
    cout << "      -host <host> : host to connect to" << endl;
    cout << "      -help : print this message and exit" << endl;
}

```

CODE EXAMPLE 13-9 Main Program for overload_get

```
int
main(int argc, char **argv)
{

    Platform plat(duEM);

    RWCString dn;
    RWCString mpa_name;
    RWCString class_name = "mpaOverloadController";
    RWCString attribute_name;
    RWCString host;

    // Get the host name.

    char * env_host = GETENV("EM_SERVER");

    if (!env_host)
    {

        char system_host[MAXHOSTNAMELEN + 1];

        sysinfo(SI_HOSTNAME, system_host, MAXHOSTNAMELEN);

        host = system_host;
    }
    else
        host = env_host;

    // Parse the cmd line for options.

    Boolean n_specified = FALSE;
    Boolean a_specified = FALSE;

    argv++;
    argc--;

    // Get command line inputs
    void get_command_line_options(int argc, char **argv, Boolean *, Boolean *,
                                RWCString &, RWCString &, RWCString &);
    get_command_line_options(argc, argv, &n_specified, &a_specified, host,
                            mpa_name, attribute_name);

    // Check if all params have been properly provided

    if (!n_specified) {
        usage();
    }
```


CODE EXAMPLE 13-9 Main Program for overload_get (Continued)

```
int
    exit(1);
}

// construct dn
dn = overload_container_name +
    RWCString ("" ) + mpa_name + RWCString("");

    cout << "\n" << endl;
cout << "=====" << endl;
    cout << "The object name is " << endl;
    cout << dn << endl;
cout << "=====" << endl;

// Connect to platform.
cout << "Connecting to ... " << host << endl;
if (!plat.connect((char *) (const char *) host,
                  "em_sample"))
{

    cout << "Failed to Connect to " << host << endl;
    cout << plat.get_error_string() << endl;

    exit(2);
}
cout << "Connected." << endl;

// Declare mpaOverloadController image.
Image im = Image((char *) (const char *) dn,
                 (char *) (const char *) class_name);

// Could not boot image
if (!im.boot())
{
    cout << "Failed to boot " << dn << endl;
    cout << im.get_error_string() << endl;
    exit(3);
}

// Perform a get on each attribute to get its value
// Note we have stripped off the document name to make
// the attribute value pairs more readable
// the chp() method of the DataUnit is necessary to null
// terminate the DataUnit.

// print the header
```

CODE EXAMPLE 13-9 Main Program for overload_get (Continued)

```
int

    cout << "\nAttribute      Value";
    cout << "\n-----      ----" << endl;

    // If an attribute is specified, get it, print it.

    if (strlen(attribute_name))
    {
        cout << attribute_name;
        cout << " "
            << im.get_str((char *) (const char *) attribute_name).chp()
            << endl;
    }
    else
    {

        // Create array attr_names of DataUnits; get attribute names, copy to
        array.

        Array(DU) attr_names = im.get_attr_names();

        for (int i = 0; i < attr_names.size; i++)
        {
            DU& name = attr_names[i];                // Get name as DataUnit.

            char *short_name = strrchr(attr_names[i].chp(), ':'); // Name as str

            // Print attribute
            cout << ++short_name << " ";

            // Print value.
            cout << im.get_str(name, USE_EXPLICIT_CHOICE | OMIT_NEWLINES).chp();
            cout << endl;
        }
    }
    exit(0);
}
```

```
/*****
```

Method:substring

General Description:

Returns true if all of sub_string matches the main_string from the beginning of main_string.

CODE EXAMPLE 13-9 Main Program for overload_get (Continued)

```
int

e.g.  there is a match if
      main_string  is "help"
      substring is  "he"

No match when:
      main_string is  "he"
      sub_string is  "help"
or
      main_string is "host"
      sub_string is  "he"

Arguments:

Return Value:

Algorithm:

Usage:

WARNING:  An empty sub_string ("") will return TRUE.

Exceptions:

*****/

Boolean substring( const char * main_string, const char * sub_string)
{
    while (*sub_string != 0)
    {
        if (*sub_string++ != *main_string++)
            return FALSE;
    }

    return TRUE;
}

//
// Get command line inputs
//
void
get_command_line_options(int argc, char **argv,
                        Boolean *n_specified, Boolean *a_specified,
                        RWCString &host,
                        RWCString &mpa_name, RWCString &attribute_name)
{
```

CODE EXAMPLE 13-9 Main Program for overload_get (Continued)

```
int
while (argc > 0 && argv[0][0] == '-')
{
    switch (argv[0][1])
    {
        case 'n':
        {
            if (!substring("n", &(argv[0][1])))
            {
                usage();
                exit(1);
            }
            argc--;
            if (!argc)
            {
                usage();
                exit(1);
            }
            argv++;

            mpa_name = argv[0];

            *n_specified = TRUE;
        }
        break;
        case 'a':

            if (!substring("attribute", &(argv[0][1])))
            {
                usage();
                exit(1);
            }
            argc--;
            if (!argc)
            {
                usage();
                exit(1);
            }
            argv++;

            attribute_name = argv[0];

            *a_specified = TRUE;

            break;

        case 'h':
```

CODE EXAMPLE 13-9 Main Program for overload_get (Continued)

```
int
    if (strlen(&(argv[0][1])) < 2)
    {
        // can not distinguish between -help and -host
        usage();
        exit(1);
    }
    if (substring("help", &(argv[0][1])))
    {
        usage();
        exit(0);
    }

    if (!substring("host", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;
    host = argv[0];

    break;

    case '?':
        usage();

        exit(1);
    }
    argv++;
    argc--;
}
```

13.4.5.5 overload_set

Purpose: Set and print attributes of the mpaOverloadController object.

CODE EXAMPLE 13-10 Syntax for overload_set

```
overload_set -n <emControllerName> -attribute <attribute>\
//          -value <attributeValue [-host <hostname>] [-help]
//
//      where <n> is emControllerName of the CMIP MPA.
//          <attribute> specifies the attribute of the object
//          <attributeValue> specifies the attribute value of the object
//          <host> specifies the MIS host
//          <help> indicate to print the command line options
//
// Note:
//      User MUST specify emControllerName in the form of
//      "CMIP MPA:mpa_host:mpa_port".
//      User MUST specify the attribute value to change to.
//
// Example 1:
//
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a 'minimumThreshold'
//                      -v 20
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a
// 'notificationEnabledStatus'
//      -v '{ { critical, disabled }, { major, disabled }, { minor, disabled
// }, { warning, disabled }, { cleared, disabled } }'
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a 'overloadInstruction'
//      -v '{ { critical, doNothing }, { major, doNothing }, { minor, doNothing
// }, { warning, doNothing } }'
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a 'overloadPollInterval'
//                      -v 60
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a 'thresholdRatePerAgent'
//      -v '{ { critical, 100 }, { major, 80 }, { minor, 60 }, { warning, 40 } }'
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a 'thresholdRatePerMPA'
//      -v '{ { critical, 100 }, { major, 80 }, { minor, 60 }, { warning, 40 } }'
//      ./overload_set -n 'CMIP MPA:thomaseng:5557' -a 'administrativeState'
//                      -v locked
//
//
#include <limits.h> // LINE_MAX
#include <netdb.h>
#include <sys/systeminfo.h>
#include <rw/cstring.h>
#include <hi.hh>
#include <installation.hh> // GETENV
```

CODE EXAMPLE 13-10 Syntax for overload_set (Continued)

```
overload_set -n <emControllerName> -attribute <attribute>\
const char *overload_container_name =
"overloadControlContainerName='OVERLOAD_CONTROL'/emControllerName=";

void usage()
{
    cout << "./overload_set -n <emControllerName in 'CMIP
MPA:mpa_host:mpa_port'> ";
    cout << " -attribute <attribute> -value <attributeValue> [-host <host>] [-
help]" << endl;
    cout << "      -n <emControllerName> : emControllerName in 'CMIP
MPA:mpa_host:mpa_port'" << endl;
    cout << "      -attribute <attribute> : attribute name" << endl;
    cout << "      -value <attributeValue>: attribute value" << endl;
    cout << "      -host <host>           : host to connect to" << endl;
    cout << "      -help                   : print this message and exit" << endl;
}
```

CODE EXAMPLE 13-11 Main Program for overload_set

```
int
main(int argc, char **argv)
{
    Platform plat(duEM);

    RWCString dn;
    RWCString mpa_name;
    RWCString class_name = "mpaOverloadController";
    RWCString attribute_name;
    RWCString attribute_value;
    RWCString host;

    // Get the host name.

    char * env_host = GETENV("EM_SERVER");

    if (!env_host)
    {
        char system_host[MAXHOSTNAMELEN + 1];

        sysinfo(SI_HOSTNAME, system_host, MAXHOSTNAMELEN);

        host = system_host;
    }
}
```

CODE EXAMPLE 13-11 Main Program for overload_set (Continued)

```
int
    else
        host = env_host;

    // Parse the cmd line for options.

    Boolean n_specified = FALSE;
    Boolean a_specified = FALSE;
    Boolean v_specified = FALSE;

    argv++;
    argc--;

    // Get command line inputs
    void get_command_line_options(int,char **,Boolean*, Boolean *, Boolean *,
                                RWCString &, RWCString &, RWCString &, RWCString &);
    get_command_line_options(argc,argv,&n_specified,&a_specified,&v_specified,
                            host,mpa_name,attribute_name,attribute_value);

    // Check if all params have been properly provided

    if (!n_specified || !a_specified || !v_specified) {
        usage();
        exit(3);
    }

    // Construct dn
    dn = overload_container_name +
        RWCString ("") + mpa_name + RWCString("");

    cout << "\n" << endl;
    cout << "=====" <<
endl;
    cout << "The object name is " << endl;
    cout << dn << endl;
    cout << "=====" <<
endl;

    // Connect to platform.
    cout << "Connecting to ... " << host << endl;
    if (!plat.connect((char *) (const char *) host,
                    "em_sample"))
    {

        cout << "Failed to Connect to " << host << endl;
```


CODE EXAMPLE 13-11 Main Program for overload_set (Continued)

```
int
    cout << plat.get_error_string() << endl;

    exit(4);
}
cout << "Connected. " << endl;

// Declare mpaOverloadController image.
Image im = Image((char *) (const char *) dn,
                 (char *) (const char *) class_name);

// Could not boot image
if (!im.boot())
{
    cout << "Failed to boot " << dn << endl;
    cout << im.get_error_string() << endl;
    exit(5);
}

// Now set the attribute.
// Set the attribute value to the object image at application side.
// The attribute value does not actually get changed in the MIS
// until the store function is successfully completed.

cout << "Set attribute " << attribute_name;
cout << " to " << attribute_value << endl;

DU prev_val = im.get_str((char *) (const char *)attribute_name);
if (!prev_val) {
    cout << "Failed to get the attribute value ";
    cout << im.get_error_string() << endl;
    exit(5);
}
cout << "Before the set operation the attribute ";
cout << attribute_name << " value is ";
cout << prev_val.chp() << endl;

// Set attribute value for the object image.

if(!im.set_str((char *) (const char *)attribute_name, (char *) (const char
*)attribute_value)) {
    cout << "Failed to set. ";
    cout << im.get_error_string() << endl;
    exit(6);
}

cout << "Store the attribute value to the MIS..." << endl;
```

CODE EXAMPLE 13-11 Main Program for overload_set (Continued)

```
int

// Store the object image back to the MIS.

if(!im.store()) {
    cout << "Failed to store. ";
    cout << im.get_error_string() << endl;
    exit(7);
}

// Get and print the new attribute value.

cout << "After the set operation the attribute ";
cout << attribute_name << " value is ";
cout << (im.get_str(attribute_name.data())).chp() << endl;

exit(0);
}

/*****

Method:substring

General Description:
Returns true if all of sub_string matches the main_string from the
beginning of main_string.

e.g.  there is a match if
      main_string  is "help"
      substring is  "he"

No match when:
      main_string is  "he"
      sub_string is  "help"
or
      main_string is "host"
      sub_string is  "he"

Arguments:

Return Value:

Algorithm:

Usage:
```

CODE EXAMPLE 13-11 Main Program for overload_set (Continued)

```
int
WARNING:  An empty sub_string (") will return TRUE.

Exceptions:

*****/

Boolean substring( const char * main_string, const char * sub_string)
{
    while (*sub_string != 0)
    {
        if (*sub_string++ != *main_string++)
            return FALSE;
    }

    return TRUE;
}

//
// Get command line input
//
void
get_command_line_options(int argc, char **argv,
                        Boolean *n_specified, Boolean *a_specified,
                        Boolean *v_specified, RWCString &host,
                        RWCString &mpa_name, RWCString &attribute_name,
                        RWCString &attribute_value)
{
    while (argc > 0 && argv[0][0] == '-')
    {
        switch (argv[0][1])
        {
            case 'n':
            {
                if (!substring("n", &(argv[0][1])))
                {
                    usage();
                    exit(1);
                }
                argc --;
                if (!argc)
                {
                    usage();
                    exit(1);
                }
                argv ++;
            }
        }
    }
}
```

CODE EXAMPLE 13-11 Main Program for overload_set (Continued)

```
int
mpa_name = argv[0];

*n_specified = TRUE;
}
break;
case 'a':

    if (!substring("attribute", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc --;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv ++;

    attribute_name = argv[0];

    *a_specified = TRUE;

    break;

case 'v':

    if (!substring("value", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc --;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv ++;

    attribute_value = argv[0];

    *v_specified = TRUE;

    break;
```

CODE EXAMPLE 13-11 Main Program for overload_set (Continued)

```
int
    case 'h':
        if (strlen(&(amp;argv[0][1])) < 2)
        {
            // can not distinguish between -help and -host
            usage();
            exit(1);
        }
        if (substring("help", &(argv[0][1])))
        {
            usage();
            exit(0);
        }

        if (!substring("host", &(argv[0][1])))
        {
            usage();
            exit(1);
        }
        argc --;
        if (!argc)
        {
            usage();
            exit(1);
        }
        argv ++;
        host = argv[0];

        break;

    case '?':
        usage();

        exit(1);
}
argv++;
argc--;
}
```

13.4.5.6 set_agent_admin_state

Purpose: Set the administrative state of the CMIP agent id and print it.

CODE EXAMPLE 13-12 Syntax for set_agent_admin_state

```
set_agent_admin_state -agent <cmip agent id> -o <object class>
//      -value <value of admin. state> [-host <host>] [-help]
//
//      where <agentId> is cmip agent id.
//      <object class> specifies the object class of the cmip agent,
//      either cmipAgent or cmipAgentEntity
//      <attributeValue> specifies the administrative state
//      attribute value of the agent
//      <host> specifies the MIS host
//      <help> indicate to print the command line options
//
//      User MUST specify the attribute value to change to.
//
// Example 1:
//
//      ./set_agent_admin_state -agent 'emperf' -v locked
//      ./set_agent_admin_state -agent 'emperf' -v unlocked
//

#include <limits.h> // LINE_MAX
#include <netdb.h>
#include <sys/systeminfo.h>
#include <rw/cstring.h>
#include <hi.hh>
#include <installation.hh> // GETENV

const char *agent_rdn = "agentTableType='CMIP'/agentId=id:";

void usage()
{
    cout << "\nUsage:" ;
    cout << " ./set_agent_admin_state -agent <cmip agent id> " ;
    cout << "-o <object class>" ;
    cout << "-value <attributeValue> [-host <host>] [-help]" << endl;
    cout << "      -agent <cmip agent id> : CMIP agent Id (e.g.'emperf') " << endl;
    cout << "      -o <object class> : object class of CMIP agent Id " << endl;
    cout << "      -value <attributeValue> : attribute value of administrative
state " << endl;
    cout << "      -host <host> : host to connect to" << endl;
    cout << "      -help : print this message and exit" << endl;
}
```

CODE EXAMPLE 13-13 Main Program for set_agent_admin_state

```
int
main(int argc, char **argv)
{

    Platform plat(duEM);

    RWCString dn;
    RWCString agent_name;
    RWCString class_name;
    RWCString attribute_name = "administrativeState";
    RWCString attribute_value;
    RWCString host;

    // Get the host name.

    char * env_host = GETENV("EM_SERVER");

    if (!env_host)
    {

        char system_host[MAXHOSTNAMELEN + 1];

        sysinfo(SI_HOSTNAME, system_host, MAXHOSTNAMELEN);

        host = system_host;
    }
    else
        host = env_host;

    // Parse the cmd line for options.

    Boolean a_specified = FALSE;
    Boolean o_specified = FALSE;
    Boolean v_specified = FALSE;

    argv++;
    argc--;

    // get command line inputs
    void get_command_line_options(int,char **,Boolean*,Boolean*,Boolean *,
                                RWCString &, RWCString&,RWCString&, RWCString&);
    get_command_line_options(argc,argv,&a_specified,&o_specified,&v_specified,
                            host,class_name,agent_name,attribute_value);

    // Check if all params have been properly provided
```

CODE EXAMPLE 13-13 Main Program for set_agent_admin_state (Continued)

```
int
if (!a_specified || !o_specified || !v_specified) {
    usage();
    exit(1);
}

// Construct dn
dn = agent_rdn +
    RWCString ("") + agent_name + RWCString("");

cout << "\n" << endl;
cout << "===== " <<
endl;
cout << "The object name is " << endl;
cout << dn << endl;
cout << "===== " <<
endl;

// Connect to platform.
cout << "Connecting to ... " << host << endl;
if (!plat.connect((char *) (const char *) host,
    "em_sample"))
{
    cout << "Failed to Connect to " << host << endl;
    cout << plat.get_error_string() << endl;

    exit(4);
}
cout << "Connected. " << endl;

// Declare cmip agent image.
Image im = Image((char *) (const char *) dn,
    (char *) (const char *) class_name);

// Could not boot image
if (!im.boot())
{
    cout << "Failed to boot " << dn << endl;
    cout << im.get_error_string() << endl;
    exit(5);
}

// Now set the attribute.
// Set the attribute value to the object image at application side.
// The attribute value does not actually get changed in the MIS
```


CODE EXAMPLE 13-13 Main Program for set_agent_admin_state (Continued)

```
int
// until the store function is successfully completed.

cout << "Set attribute " << attribute_name;
cout << " to " << attribute_value << endl;

DU prev_val = im.get_str((char *) (const char *)attribute_name);
if (!prev_val) {
    cout << "Failed to get the attribute value ";
    cout << im.get_error_string() << endl;
    exit(5);
}
cout << "Before the set operation the attribute ";
cout << attribute_name << " value is ";
cout << prev_val.chp() << endl;

// Set attribute value for the object image.

if(!im.set_str((char *)(const char *)attribute_name, (char *)(const char
*)attribute_value)) {
    cout << "Failed to set. ";
    cout << im.get_error_string() << endl;
    exit(6);
}

cout << "Store the attribute value to the MIS..." << endl;

// Store the object image back to the MIS.

if(!im.store()) {
    cout << "Failed to store. ";
    cout << im.get_error_string() << endl;
    exit(7);
}

// Get and print the new attribute value.

cout << "After the set operation the attribute ";
cout << attribute_name << " value is ";
cout << (im.get_str(attribute_name.data())).chp() << endl;

exit(0);

}
```

/*****

CODE EXAMPLE 13-13 Main Program for set_agent_admin_state (Continued)

```
int
Method:substring

General Description:
Returns true if all of sub_string matches the main_string from the
beginning of main_string.

e.g.  there is a match if
      main_string  is "help"
      substring is  "he"

No match when:
      main_string is  "he"
      sub_string is  "help"
or
      main_string is "host"
      sub_string is  "he"

Arguments:

Return Value:

Algorithm:

Usage:

WARNING:  An empty sub_string ("") will return TRUE.

Exceptions:

*****/

Boolean substring( const char * main_string, const char * sub_string)
{
    while (*sub_string != 0)
    {
        if (*sub_string++ != *main_string++)
            return FALSE;
    }

    return TRUE;
}

void
get_command_line_options(int argc, char **argv,
                        Boolean *a_specified, Boolean *o_specified,
                        Boolean *v_specified, RWCString &host,
```

CODE EXAMPLE 13-13 Main Program for set_agent_admin_state (Continued)

```
int
    RWCString &class_name, RWCString &agent_name,
    RWCString &attribute_value)
{
    while (argc > 0 && argv[0][0] == '-')
    {
        switch (argv[0][1])
        {
            case 'a':
            {
                if (!substring("agent", &(argv[0][1])))
                {
                    usage();
                    exit(1);
                }
                argc--;
                if (!argc)
                {
                    usage();
                    exit(1);
                }
                argv++;

                agent_name = argv[0];

                *a_specified = TRUE;
            }
            break;
            case 'o':

                if (!substring("object_class", &(argv[0][1])))
                {
                    usage();
                    exit(1);
                }
                argc--;
                if (!argc)
                {
                    usage();
                    exit(1);
                }
                argv++;

                class_name = argv[0];

                *o_specified = TRUE;
```

CODE EXAMPLE 13-13 Main Program for set_agent_admin_state (Continued)

```
int
    break;

case 'v':

    if (!substring("value", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
    argv++;

    attribute_value = argv[0];

    *v_specified = TRUE;

    break;
case 'h':
    if (strlen(&(argv[0][1])) < 2)
    {
        // can not distinguish between -help and -host
        usage();
        exit(1);
    }
    if (substring("help", &(argv[0][1])))
    {
        usage();
        exit(0);
    }

    if (!substring("host", &(argv[0][1])))
    {
        usage();
        exit(1);
    }
    argc--;
    if (!argc)
    {
        usage();
        exit(1);
    }
}
```

CODE EXAMPLE 13-13 Main Program for set_agent_admin_state *(Continued)*

```
int
    argv++;
    host = argv[0];

    break;

    case '?':
        usage();

        exit(1);
    }
    argv++;
    argc--;
}
}
```


PART **IV** Nerve Center

Nerve Center Overview

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

This chapter describes the following topics:

- Section 14.1 “Nerve Center Components” on page 14-1
- Section 14.2 “Nerve Center Documentation” on page 14-2
- Section 14.3 “Nerve Center Operation” on page 14-3

14.1 Nerve Center Components

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 Design Advanced Requests to build sets of instructions called “conditions.” Conditions are the building blocks of request templates.
- A set of applications that allows you to create, save, and debug request templates, and launch and monitor requests. This includes:
 - The Design Advanced Requests, 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 lists running requests and allows you to stop or examine them.

- Facilities in the Design Advanced Requests (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. Some of the Nerve Center functions can be done using Simple Requests (see Chapter 15). The Nerve Center Interface Library is described in the “Nerve Center Interface Library” chapter in the *API Syntax*.

14.2 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 15, “Requesting Data in Solstice EM,” provides step-by-step guidance on building and debugging request templates using the Design Advanced Requests, RCL, `em_debug`, and Simple Request.
- Chapter 17, “Building Templates for SunNet Manager Event Requests,” describes building and using request templates with SunNet Manager RPC agents.
- Chapter 18, “Building Advanced Requests,” describes the features and usage of the Design Advanced Requests application.
- Chapter 20, “Request Condition Language,” describes the components of the Request Condition Language used to build Nerve Center request templates.
- Chapter 22, “RCL Functions,” describes the built-in functions that can be used in building RCL conditions.
- The Requests tool is described in Chapter 4 of *Managing Your Network*.
- The Nerve Center interface library is described in the *API Syntax*.
- The request templates shipped with Solstice EM are described in Chapter 4 of *Managing Your Network*.

14.3 Nerve Center Operation

14.3.1 How a Request Gets Information

A request can get information in the following ways:

- 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.
- It can *subscribe* for event notifications. The Nerve Center sets the values of variables related to the notification before it “awakens” the request.
- 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 Chapter 20.

14.3.1.1 Where and When a Request’s Notifications Arise

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

- 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:
 - All notifications of a particular type, or
 - 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.

- 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. Actions are operations performed after the transition occurs from one state to another. Every poll of an attribute results in a CMIS GET request internally. RCL provides a system variable call poll. This variable must be set to the right object FDN before the polling occurs. This variable value is used by the FDR for the CMIS GET request. Then several attributes can be specified in a condition for a state or a set of conditions for a state. All of the attributes must refer to the same object to get the valid poll results for all of the attributes.

A state must be awakened by retrieving information from the MIS. Otherwise, a request sleeps in that state indefinitely. 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.

14.3.1.2 When Information From Managed Objects can Arrive

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

- 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.

- 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()`

14.3.2 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 Design Advanced Requests tool automatically scans the definitions of all the conditions mentioned in the template for references to variables and attributes. A template represents the definition, and a request is a running instance of a template.

14.3.2.1 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 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 14.3.5 “Specifying the Objects to be Polled” on page 14-8 for more information.

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()`, as shown in the following code example.

CODE EXAMPLE 14-1 Attributes

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

14.3.2.2 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 20 for a list of system variables and a list of the possible values of the `$messType` variable.

14.3.2.3 User Variables

Any condition can create a variable by using the name of the variable to the left of an equal (=) 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.

14.3.2.4 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.

14.3.3 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 when

- 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.

14.3.4 Action at a Transition

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

14.3.4.1 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 the following table.

TABLE 14-1 Action Menu Items

Action	Description
<none>	No action taken.
UNIXCMD	The name of command with any required parameters. For example, for <code>netstat -rn</code> , you enter <code>netstat</code> in the Command field and <code>-rn</code> in the Arguments field.
MAIL	An electronic mail address and message. For example, <code>verma@halcyon</code> in the Address field and <code>CPU usage exceeded 90%</code> 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 Design Advanced Requests (which saves it into the MIS).

You can specify any combination of the supported actions to occur in a transition.

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.

14.3.4.2 Logging an Event

A log object (described in Chapter 5) 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 20) provides alarm logging functions (`alarm()`, `alarmOi()`, `alarmStr()`, and `sendEvent()`) to send notifications that can be logged to the log object `alarmLog` or a user defined Log Object. 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 generate only `nerveCenterAlarms`. The `sendEvent()` function can be used to send 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. By default, Nerve Center Alarms are automatically logged into the Alarm Log.

14.3.4.3 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.

14.3.5 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. In most cases you will not want to write a new template for each target object. Typically, you define a template that can be used to manage objects of the same type — for example, a certain type of router manageable via SNMP. 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-mibIII` system group, such as `sysDescr`.

When a user launches a request template against a device selected in the Network Views, Nerve Center places all of the managed objects configured for the device into the system variable `$pollFdnSet`. “Managed objects” are internal representations in the MIS of the agent capabilities supported by the device. If Network Discovery was configured to search for RPC-based SunNet Manager agents when populating the MIS, devices that have both SNMP and RPC agents are configured in the MIS to indicate this. Fully distinguished names (FDNs) pointing to these managed objects in the MIS are thus loaded into the `$pollFdnSet` variable when a user launches 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. In designing a Nerve Center template, a typical task that should be performed in the template’s initialization is 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 shipped with Solstice EM, 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.

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: `$pollfdn = appendRdn($dn,"/internetSystemId=NULL") ;`

For example, the router `bigguy` is an SNMP agent system. A request launched against the `bigguy` has its `$pollfdn` set to the `cmipsnmpProxyAgent` / `systemId=name:"gato loco"/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 append to the `ObjectInstance` a further specification of the object it wishes to poll.

In a Fully Distinguished Name (FDN) of the form `/< 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.

To set the `$pollfdn` to point to the `snmp-mibII system` group, the template must concatenate the Relative Distinguished Name (RDN) for the object of interest, which in this case is `system`. The `SetInternetSystem` sample condition, for example, resets the value of `$pollfdn` to point to the `snmp-mibII system` group object as shown in the following code example.

```
$tmp = "/InternetSystemId=NULL";
$pollfdn = appendRdn($pollfdn,$tmp);
true;
```

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 router `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"/InternetSystemId=NULL
```

14.3.6 Alarm Logging and the Alarm Service

The RCL alarm-logging functions (`alarm()`, `alarmStr()`, and `alarmOi()`) allow 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 Alarms application.

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 Network Views. Fault status is an attribute of topology nodes, which are represented by icons

in the Network Views. 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 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. Requests can clear previous alarms they have posted against a device by posting an alarm with a severity of cleared. For example:

```
$info = StrToAsn("EM-NC-ASN1.NerveCenterAlarmInfo", "{1,5,\"Device  
is up\",3,1}");  
alarmOi($save_pollfdn,$info);
```

Note that, in the above example, the `probableCause` value (the first value in the set of ASN.1 values making up the alarm) has been set to 1 in the clear 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 of the alarm. For this reason, only the `alarmOi()` function can be used to log alarms that clear previous alarms.

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 can 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);
```


Requesting Data in Solstice EM

Solstice Enterprise Manager (Solstice EM) enables you to request data from agents to monitor the status of network components and to take action in response. This proactive management of network conditions enables you to quickly find and remedy faults and enhance the performance of your network.

This chapter describes the following topics:

- Section 15.1 “Polling for Data in Solstice EM” on page 15-1
- Section 15.2 “Subscribing for Events” on page 15-3
- Section 15.3 “Using Solstice EM Tools for Polling” on page 15-4
- Section 15.4 “Working with Basic Requests” on page 15-5
- Section 15.5 “Working with Advanced Requests” on page 15-12
- Section 15.6 “Building Blocks: States, Transitions, and Conditions” on page 15-15
- Section 15.7 “Designing Request Templates” on page 15-26
- Section 15.8 “Requests Based on Polling” on page 15-28
- Section 15.9 “Polling RPC Agents” on page 15-35
- Section 15.10 “Requests Based on Event Subscription” on page 15-41
- Section 15.11 “Subscribing for Enterprise-Specific SNMP Traps” on page 15-42
- Section 15.12 “Requests that Combine Subscription and Polling” on page 15-47
- Section 15.13 “Building Request Definitions” on page 15-51

15.1 Polling for Data in Solstice EM

The architecture of Solstice EM enables you to keep informed of changes in status of network components and overall network conditions. Through multiple Solstice EM tools, you can design and customize request templates, scripts that enable a variety of different actions to be completed, each of which causes a specific event to occur. Request templates are used to poll or subscribe to an agent for data about a network component.

Polling is a means to obtain data from an agent configured for a network component. Attribute data, such as the percentage of disk capacity used, can be useful for fine-tuning the performance of your network or for locating system faults.

Solstice EM supports three ways to poll for data:

- Direct polling
- Indirect polling
- Event Request polling

15.1.1 Direct Polling

Direct polling supports Simple Network Management Protocol (SNMP) agents. The term, direct polling, refers to the ability to directly request information from an SNMP agent configured for a network component. The request is initiated from an application and is translated into an SNMP request by the SNMP Management Protocol Adapter (MPA). Direct polling is advantageous for direct communication between the SNMP MPA and the MIS. However, as the request must travel through the SNMP MPA and the MIS, the MIS can support only a limited amount of direct polls. For more information about Solstice EM architecture, including the MIS and MPAs, see the *Management Information Server (MIS) Guide*.

15.1.2 Indirect Polling

Indirect polling enables an SNMP agent configured for a network component to obtain attribute data from a SunNet Manager (SNM) RPC agent or proxy agent via the MIS. 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 17.)

15.1.3 Event Request Polling

Event request polling refers to setting up a Nerve Center request template to poll an agent for states and conditions of a network component. The agent configured for the network component sends the requested data to the Nerve Center of the MIS. The Nerve Center then issues instructions to the agent to complete an action depending on the data returned.

Event request polling is highly scalable because the polling is completed by remote agents. An unlimited number of remote agents may be distributed around the network.

15.2 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 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.

15.2.1 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 subscribe for incoming `linkDown` traps and then use polling to count the elapsed time before the arrival of a `linkUp` trap for the downed router interface. If the elapsed time exceeds a certain threshold, the request might then emit a Nerve Center alarm. A sample template that illustrates this scenario is described in Section 15.12, “Requests that Combine Subscription and Polling.”

15.3 Using Solstice EM Tools for Polling

Solstice EM provides several tools for creating requests. The following table describes each tool.

TABLE 15-1 Solstice EM Request Tools

Tool	Description	For More Information . . .
Basic Requests	Enables you to set up individual or groups of requests to be sent to an SNMP agent. This tool is accessible from the Network Views Tool menu or by typing <code>em_simplerequests</code> at the command line.	For introductory information about Basic Requests, refer to Chapter 4 in <i>Managing Your Network</i> . For more complete information, refer Section 15.4 “Working with Basic Requests” on page 15-5 of this book.
Simple Requests	Same as Basic Requests	Same as Basic Requests.
Advanced Requests	Enables you to create, modify, and start Nerve Center request templates	For introductory information about Advanced Requests, refer to Chapter 4 in <i>Managing Your Network</i> . For more complete information, see Section 15.7 “Designing Request Templates” on page 15-26 of this book.
Design Advanced Requests	Enables you to create requests to be launched against SNMP or SNM RPC agents, including specifications of states, conditions, transitions, and the poll rate.	For complete information, see Section 15.7 “Designing Request Templates” on page 15-26.
Request Controllers	Enables you to automate Nerve Center requests to issue many requests at a time. Involves correlating a Nerve Center request template to a type of object. Each time that type of object is created, the <code>autoManagement</code> daemon is launched against the object to obtain its status. Accessible from the Administration tool in Network Tools.	Refer to Chapter 7 in <i>Managing Your Network</i> .

15.4 Working with Basic Requests

15.4.1 Viewing Basic Request Information

You can view information about basic agent-oriented SNMP, RPC, and CMIP requests by clicking Tools → Basic Requests from the Network Views main window.

▼ To View Basic Agent Request Information

1. In the Network Views main window, select an object associated with the agent for which you want to display request information, and then click Tools → Basic Requests to open the Basic Requests tool, as illustrated in the following figure.

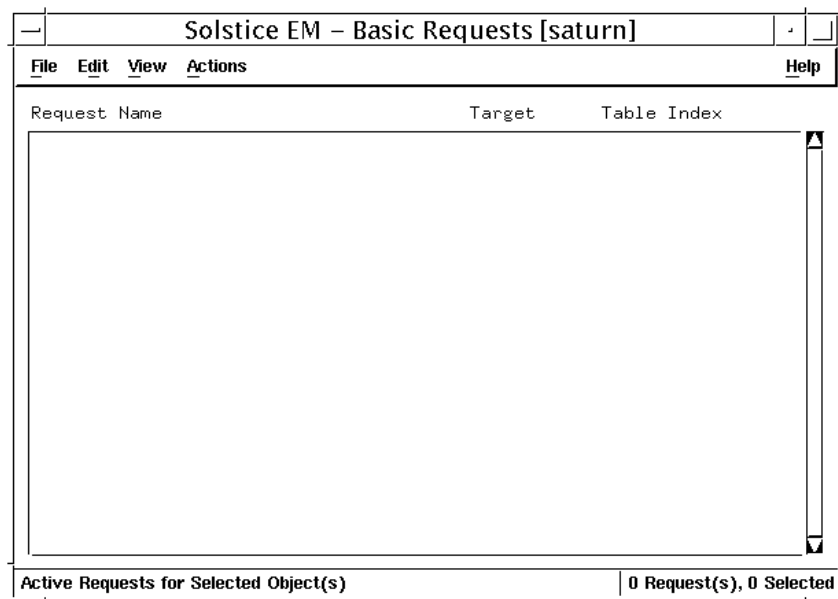
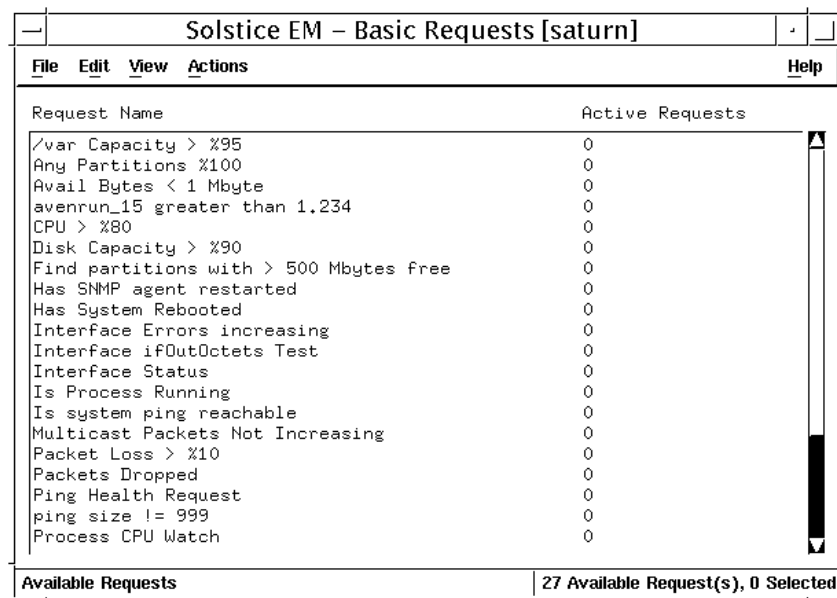


FIGURE 15-1 Viewing Request Information in the Basic Requests Main Window

2. Click **View → Requests → Available** or **View → Request Groups** to view available or active requests, or request groups, as desired.

A list of active requests, available requests (see FIGURE 15-2), or request groups (see FIGURE 15-3) is displayed, according to your choice.



The screenshot shows a window titled "Solstice EM – Basic Requests [saturn]". It has a menu bar with "File", "Edit", "View", "Actions", and "Help". Below the menu bar is a table with two columns: "Request Name" and "Active Requests". The table lists 20 requests, all with a value of 0 in the "Active Requests" column. A vertical scrollbar is on the right side of the table. At the bottom of the window, there is a status bar that reads "Available Requests" and "27 Available Request(s), 0 Selected".

Request Name	Active Requests
✓var Capacity > %95	0
Any Partitions %100	0
Avail Bytes < 1 Mbyte	0
avenrun_15 greater than 1.234	0
CPU > %80	0
Disk Capacity > %90	0
Find partitions with > 500 Mbytes free	0
Has SNMP agent restarted	0
Has System Rebooted	0
Interface Errors increasing	0
Interface ifOutOctets Test	0
Interface Status	0
Is Process Running	0
Is system ping reachable	0
Multicast Packets Not Increasing	0
Packet Loss > %10	0
Packets Dropped	0
Ping Health Request	0
ping size != 999	0
Process CPU Watch	0

Available Requests | 27 Available Request(s), 0 Selected

FIGURE 15-2 Viewing Requests in the Basic Requests Available Window

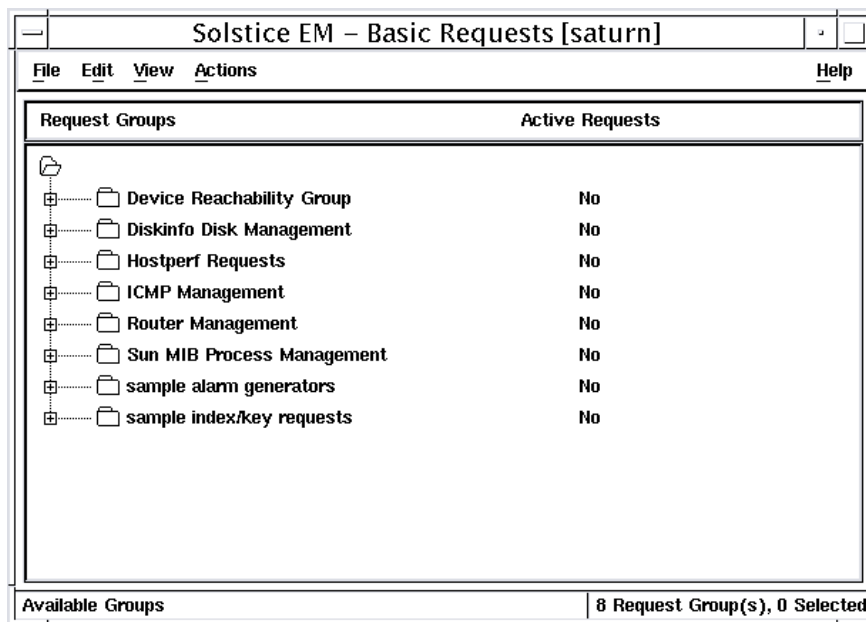


FIGURE 15-3 Viewing Request Groups in the Basic Requests Groups Window

3. Select the request or request group for which you want more information, and then click **Actions → Properties** to display detailed information.

In the following figure, see request for Avail Bytes < 1Mbytes as well as see FIGURE 15-5 and FIGURE 15-6.



FIGURE 15-4 Basic Requests Properties Conditions Window



FIGURE 15-5 Basic Requests Properties General Window

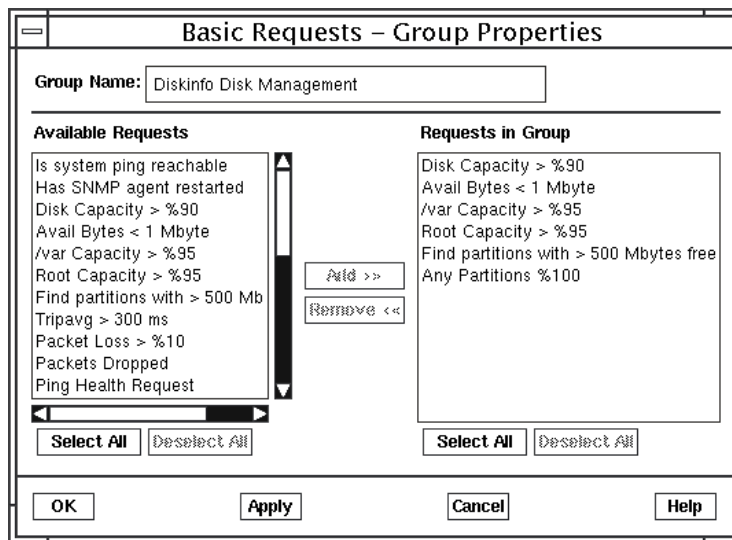


FIGURE 15-6 Basic Requests Group Properties Window

15.4.2 Creating, Modifying, and Initiating Basic Requests

You can use the Basic Requests tool to start, stop, create, and modify basic requests and request groups based on agent conditions. Complete instructions for using the Basic Requests tool are provided in the *Managing Your Network*.

▼ To Initiate a Basic Request

1. In the Network Views main window, select an object associated with the agent for which you want to initiate a request, and then click **Tools → Basic Requests** to open the Basic Requests tool.
2. Click **View → Requests → Active Requests**, or **View → Request Groups** to view active requests or request groups, as desired.
3. Select the request you want to initiate, and then click **Actions → Start**.

▼ To Halt a Basic Request

1. In the Network Views main window, select an object associated with the agent for which you want to halt a request, and then click Tools → Basic Requests to open the Basic Requests tool.
2. Click View → Requests → Active Requests, or View → Request Groups to view active requests or request groups, as desired.
3. Select the request you want to halt, and then click Actions → Stop.

▼ To Create a Basic Request or Request Group

1. In the Network Views main window, select an object associated with the agent for which you want to create a request, and then click Tools → Basic Requests to open the Basic Requests tool.
2. Click Actions → Create → Request or Actions → Create → Request Group, as desired, to open the Create Request or Create Group dialog.
3. Specify Request or Request Group options as needed.
Refer to *Managing Your Network* for complete information about the Create Request and Create Request Group options.

▼ To Modify a Basic Request or Request Group

1. In the Network Views main window, select an object associated with the agent for which you want to modify a request, and then click Tools → Basic Requests to open the Basic Requests tool.
2. Click View → Requests or View → Request Groups to view available or active requests, or request groups, as desired.
A list of active requests, available requests, or request groups is displayed, according to your choice.
3. Select the request or request group you want to modify, and then click Actions → Properties to display detailed information.
4. Specify Request or Request Group options as needed.

15.5 Working with Advanced Requests

▼ To View Advanced Request Information

1. In the Network Views main window, click **Tools → Advanced Requests** to display the **Requests** dialog.

All available and active requests are displayed in two lists at the top and bottom of the dialog, as shown in the following figure.

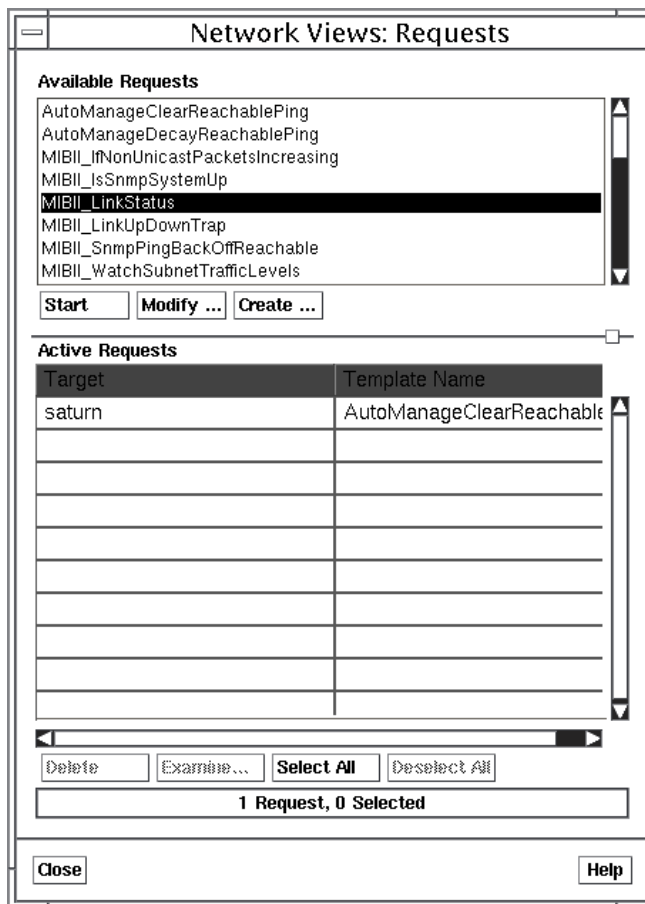


FIGURE 15-7 Viewing Available Requests in the Advanced Request Dialog

2. Click **Start** to display the selected **Available Requests** in the **Active Requests** list.
3. In the **Active Requests** list, select the request that you want to view detailed information for.
4. Click **Examine** to open the **Request Examine** dialog, as illustrated in the following figure.

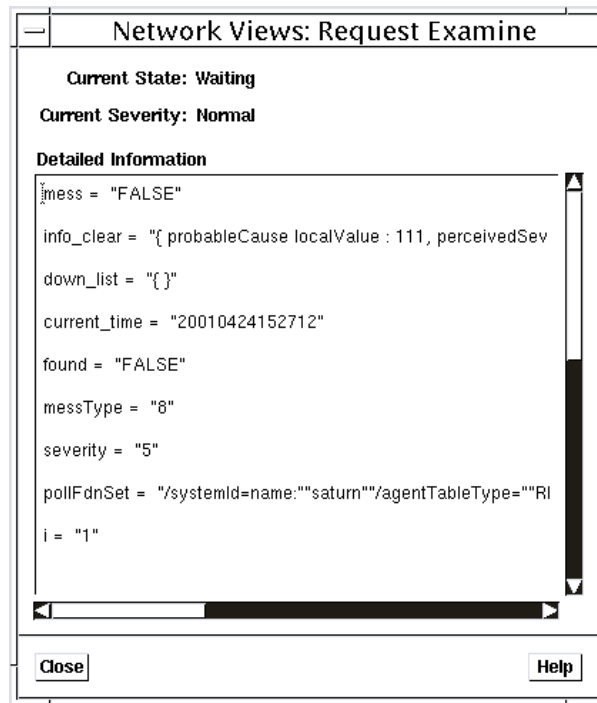


FIGURE 15-8 Advanced Request Examine Window

15.5.1 Creating, Modifying, and Initiating Advanced Requests

You can use the Network Views Requests dialog to start and stop requests based on existing request templates. You can also use the Requests dialog to open the Request Designer tool, from which you can perform a full range of request management tasks.

▼ To Initiate an Advanced Request

1. In the Network Views main window, click **Tools → Advanced Requests** to display the Requests dialog.
2. Select the request you want to start from the Available Requests list.
3. Select the target object(s) in the current view, and then click **Start**.
Alternatively, you can use the middle mouse button to drag a request from the Available Requests list and drop it on an object in the Network Views window.
4. Repeat Steps 2 and 3 for each request you want to initiate, or click **Close** to exit the Requests dialog.

▼ To Halt an Advanced Request

1. In the Network Views main window, click **Tools → Advanced Requests** to display the Requests dialog.
2. Select the request you want to halt from the Active Requests list.
3. Click **Delete**.
The request template itself is not deleted from the MIS; only the particular request object instance based on the request template is deleted.
4. Repeat Steps 2 and 3 for each request you want to halt, or click **Close** to exit the Requests dialog.

▼ To Create or Modify an Advanced Request

1. In the Network Views main window, click **Tools → Advanced Requests** to display the Requests dialog.
2. In the Requests dialog, do either of the following:
 - If you want to modify a request, select the request from the Available Requests lists, and then click **Modify**.
 - If you want to create a request, click **Create**.In both cases, the Request Designer (`em_reqedit`) is opened.
3. In the Request Designer, modify or create requests or request templates as desired.

15.6 Building Blocks: 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.

Note – The “severities” that attach to template states in the Design Advanced Requests do not control the fault status indication (icon color) of devices in the Network Views; severities of states only affect the color attached to states in the Design Advanced Requests’s graphical display. Fault status color of devices in the Network Views 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-emitting functions. This is discussed below in “Controlling Fault Status Color” on page 23.”

In designing a template you will need to specify 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 Design Advanced Requests. 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 “awake.” The current state is awake only if one of the following occur:

- The request has received an incoming event notification.
- The condition attempts to access 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.

15.6.1 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, emitting an alarm or sending an email message).

A state diagram shows how a request template works. The following figure shows a simple example that illustrates request-related concepts.

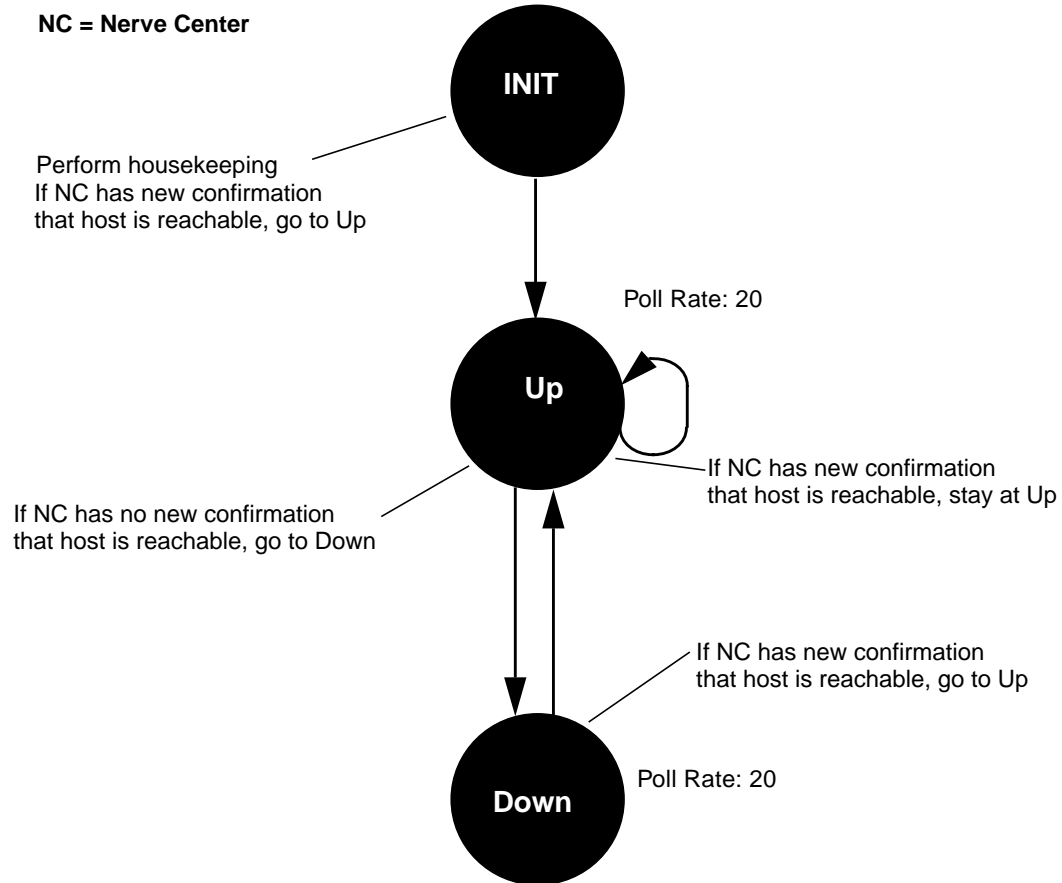


FIGURE 15-9 Request Example with Poll Rates

The following figure is a state diagram for an example request template that adds the “Missed” and “InitComplete” states to the template in FIGURE 15-9.

Note that a request can cause a change in the fault status colors for the device icon in the Network Views only if an alarm-emitting function (such as `alarmOn()`) is used in a condition.

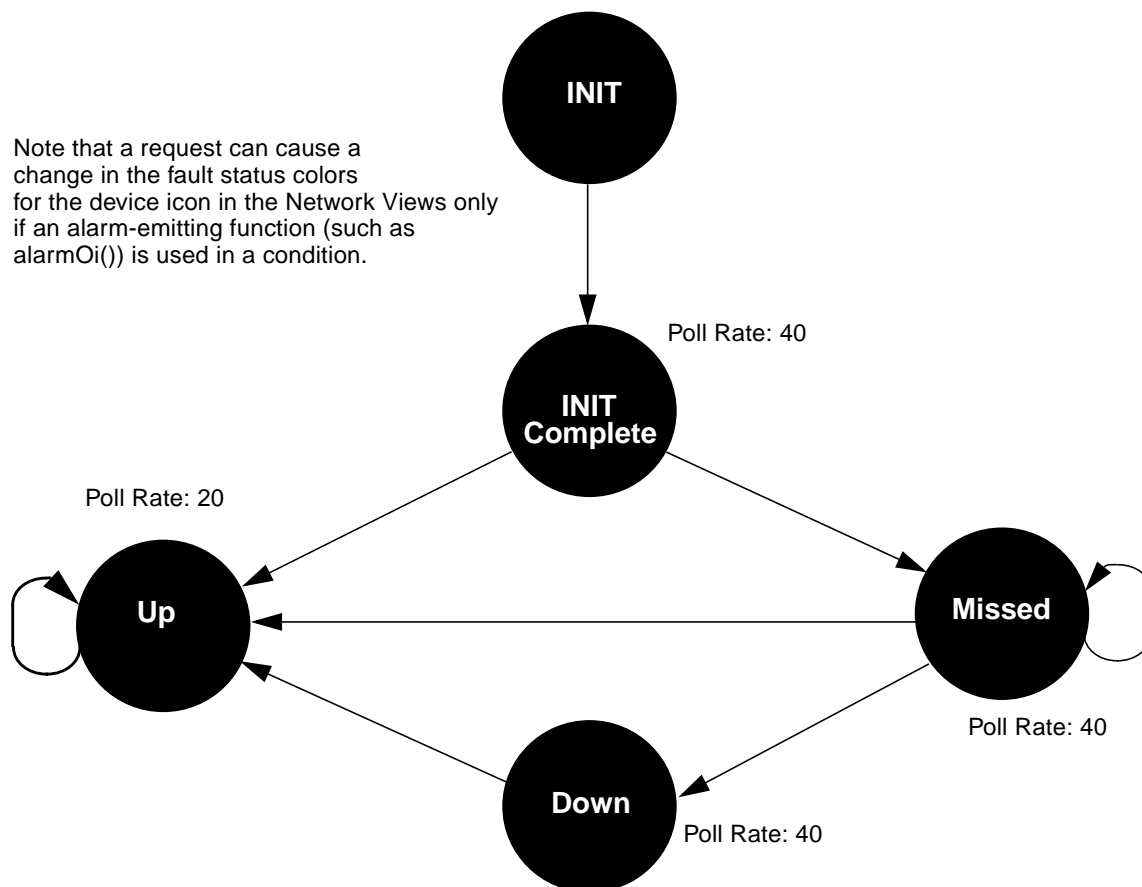


FIGURE 15-10 Request Example with Poll Rates and Severities

The following figure shows the same template as shown in FIGURE 15-10 with the conditions associated with each transition.

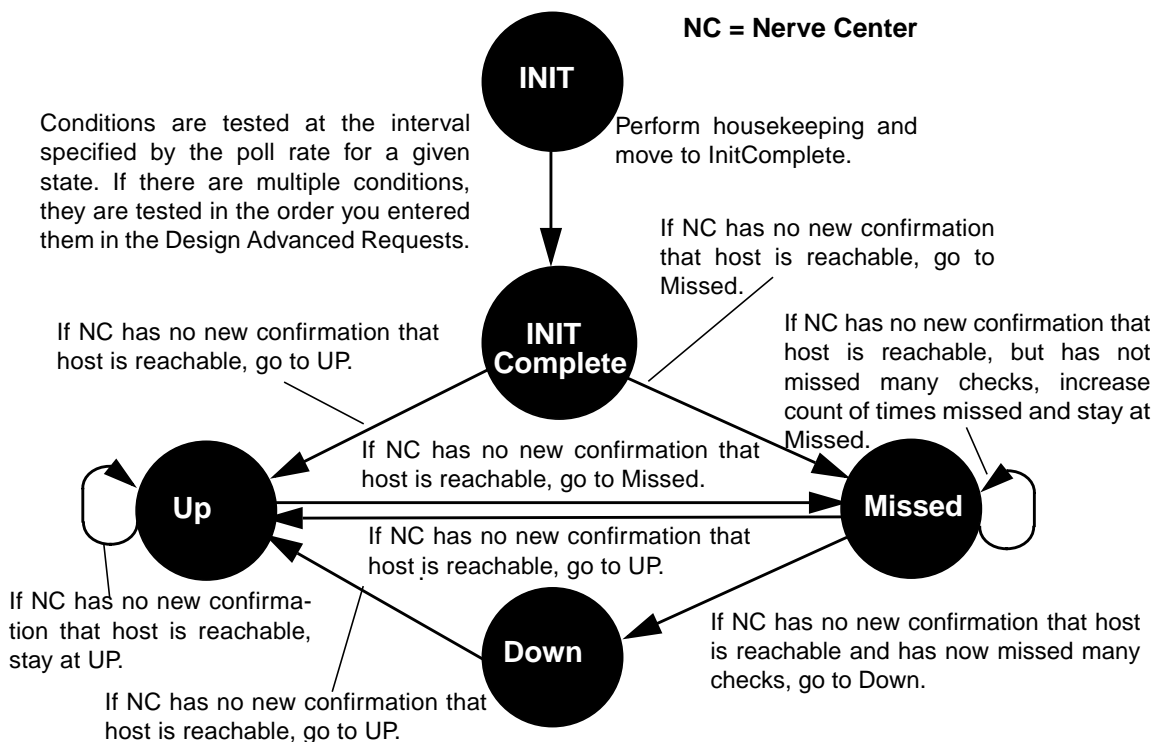


FIGURE 15-11 Request Example with Conditions

In FIGURE 15-11, the conditions are described in English. In an actual request template, you define conditions in the Request Condition Language.

15.6.2 Sample Request Template

FIGURE 15-12 illustrates the workings of a sample request template `IsSnmppSystemUp`, which is shipped with Solstice EM. This template does what its name suggests: It 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 15-11.

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 `AlarmCriticalOid` 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.

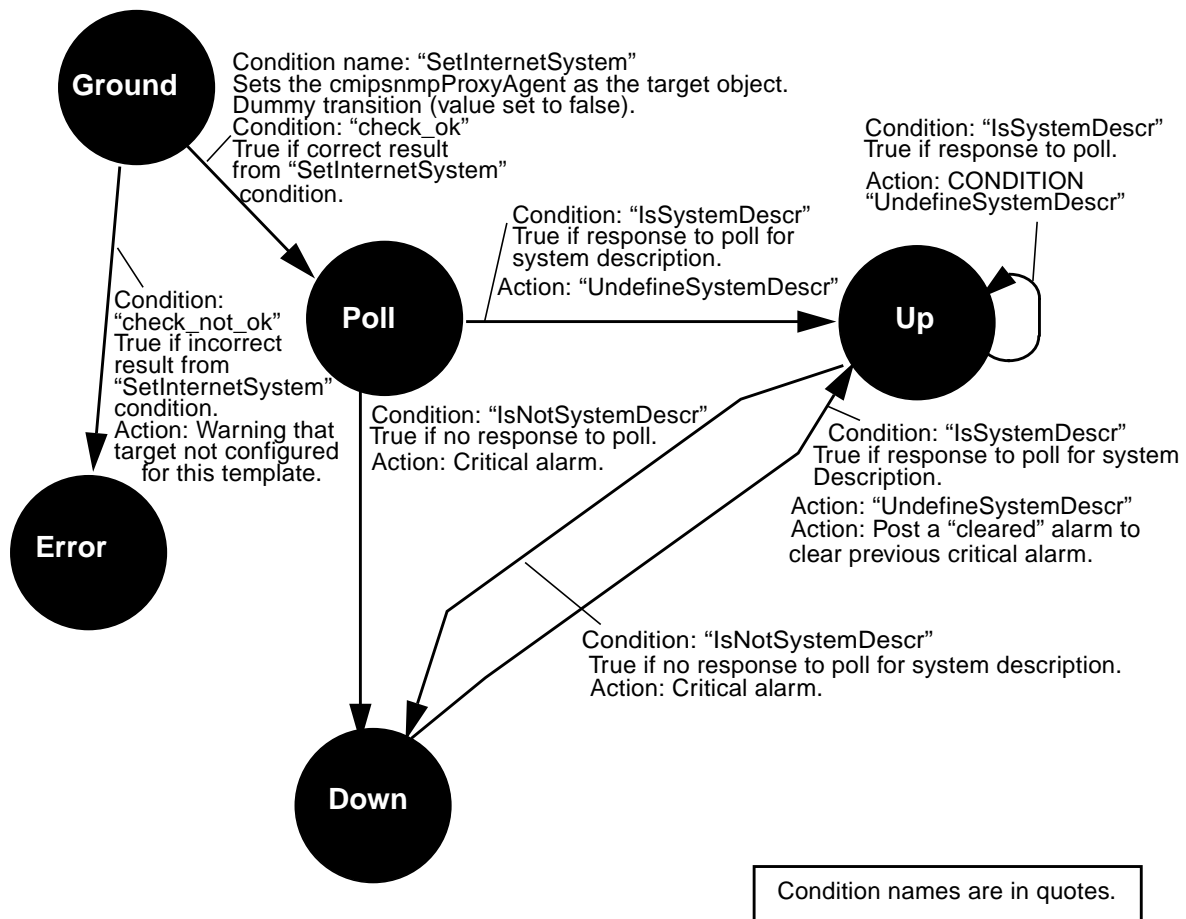


FIGURE 15-12 IsSnmppSystemUp Sample Request Template

In the Design Advanced Requests' graphical display, the "IsSnmppSystemUp" request template is displayed much as it appears in FIGURE 15-12, 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 IsSnmppSystemUp template (FIGURE 15-12) has five states: Ground, Error, Poll, Up, and Down.

15.6.2.1 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 Network Views, Nerve Center places all of the managed objects configured for the device into the system variable `$pollFdnSet`. “Managed objects” are internal representations in the MIS of the agent capabilities supported by the device. If you configured Network Discovery to search for RPC-based SunNet Manager agents when you populated your MIS, devices that have both SNMP and RPC agents are configured in the MIS to indicate this. Fully distinguished names (FDNs) pointing to these managed objects in the MIS are thus loaded into the `$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 there 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,"/internetSystemId=NULL");
        $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

15.6.2.2 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 Design Advanced Requests displays an error message and the condition is not saved to the MIS. (You can use the SNMP Data application to examine the attributes and groups supported by GDMO documents that the MIS knows about.)

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.

15.6.3 Controlling Fault Status Color

Your network topology is represented in the Network Views 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 of 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 Network Views 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 represents 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 fully 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 Network Views.

When a request is launched against a selected device in the Network Views, the topology node's list of managed objects (`topoNodeMOSet`) is loaded into the RCL system variable `$pollFdnSet`. Thus, if you want to use the RCL alarm-logging functions to change the fault status color for the device in the Network Views, you need to ensure that alarms are logged against one of the objects in `$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 approach 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. And the `internetSystem` group object is not a member of `$pollFdnSet`. Therefore, 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 condition:

```
$info = StrToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{3,minor,\"Device is up  
after being down\",3,1}");  
alarmOi($save_pollfdn,$info);
```

In this example the ASN.1 alarm type (a Nerve Center alarm), and the ASN.1 values of the alarm attributes, are derived from their text equivalents using the `strToAsn()` function. The ASN.1 values are then passed to `alarmOi()`. The curly braces (“{,”}”) indicate the set of attributes that are to make up the Nerve Center alarm that is posted by `alarmOi()`. The first value (3 in this case) is the `probableCause`, the second value (minor) is the `perceivedSeverity` of the alarm, the fourth value is `additionalText` (which is an optional attribute). The last two values (`mosiState` and `mosiSeverity`) are not significant.

15.6.3.1 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 15-14, 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 (the first value in the set of ASN.1 values making up the alarm) has been set to 1 in the clear 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 of the alarm. 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;
```

15.6.3.2 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 following chapters in this guide:

- Chapter 4, "Using the Alarm Service "
- Chapter 14, "Nerve Center Overview "
- Chapter 18, "Building Advanced Requests "
- Chapter 20, "Request Condition Language "
- Chapter 22, "RCL Functions "

15.7 Designing Request Templates

Before embarking on building a request template from scratch, use the Design Advanced Requests to examine the sample request templates supplied with the product. You may be able to use a template as is, modify a template, or 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.

▼ To Create a Request Template

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 Design Advanced Requests, as described below.**

All the following steps involve the use of the Design Advanced Requests.

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 Design Advanced Requests' Template → File → Export Current option. Use the Template → import option when you want to reload that template into the Design Advanced Requests.

With a template created, you can invoke the Advanced Requests window from the Network Views'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 Design Advanced Requests, although you can use the application's graphical display as your drawing board. We recommend composing your conditions in the Design Advanced Requests, 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 Design Advanced Requests gives you a choice of a text-based or a graphical method of creating templates.

15.8 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 Network Views, we can log an alarm with a `perceivedSeverity` of minor when a transition to the `EverDown` state occurs. However, this will only cause the Network Views 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 Network Views 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 Network Views 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 as a latch, to ensure that the request does not flood the alarm log with duplicate alarms.

As in the `IsSnmppSystemUp` 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 “`IsSnmppSystemEverDown`”; a possible state diagram for our template is pictured in FIGURE 15-13.

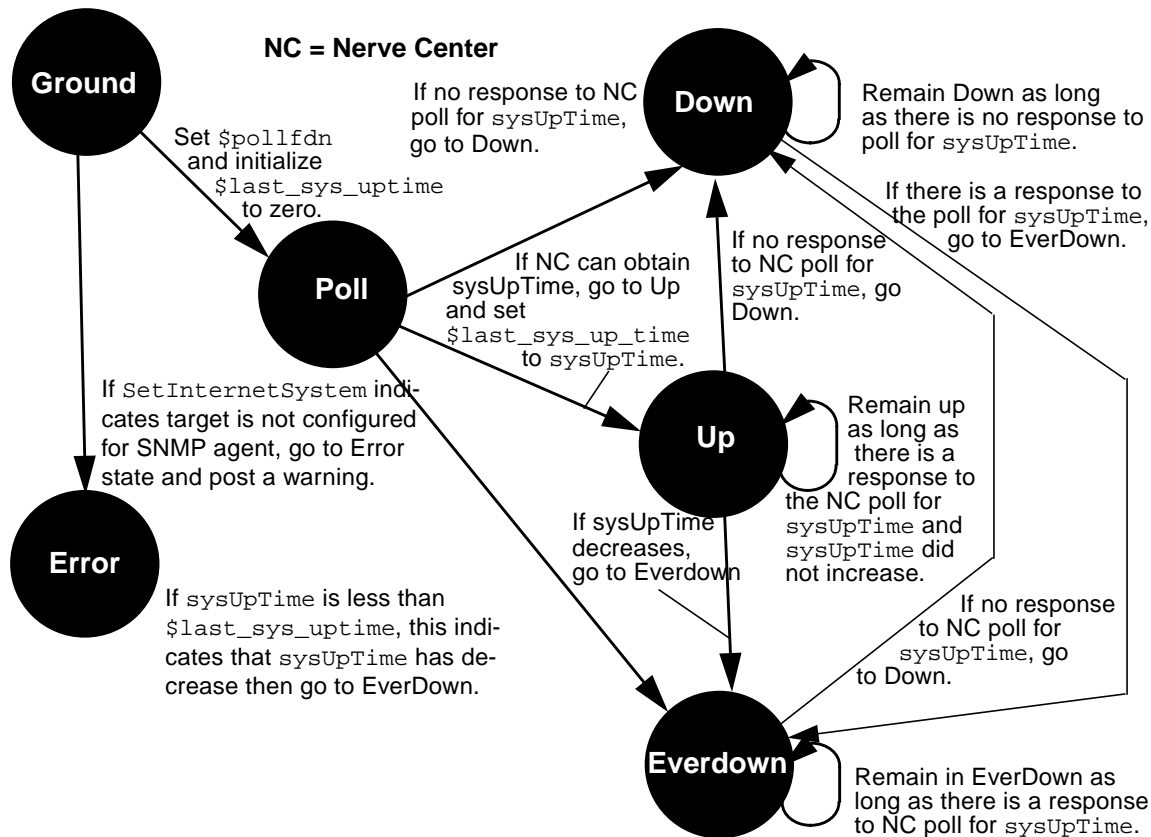


FIGURE 15-13 State Diagram of `IsSnmptSystemEverDown` Template

15.8.1 Adding States

If you want to add a state, for example `EverDown`, to the template use the following procedure.

▼ To Add a State

1. **Click States.**
2. **Type in the name (“EverDown”) in the State field.**

3. Describe what the state represents in the Description field.

For example, “Agent system is up but has been down.”

4. Select a poll rate from the Poll Rate menu.

Note – 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).

5. 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 Network Views 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 15.6.3, “Controlling Fault Status Color .”

6. Click Add to add the new state to the template.

Repeat this procedure for the other states to be added.

15.8.2 Adding Conditions

To add a condition, invoke the Conditions window by clicking Conditions in the Design Advanced Requests 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. Refer to the following figure.



FIGURE 15-14 Entering Condition Code in the Design Advanced Requests

This RCL statement returns a value of true when (and only when) the current value of the `sysUpTime` attribute is less than the value currently stored in the `$last_sys_up_time` variable.

▼ To Add the New Condition to the MIS

1. Enter a name (“`sysUpTimeNotDecrease`”) for the condition in the **Name** field.
2. Describe what the condition does in the **Description** field.
3. Click **Add** to store the new condition in the MIS.

Repeat this procedure for the other conditions to be added.

15.8.3 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, do the following steps to create the first transition from the Ground to Poll state.

▼ To Create the Transition

1. Invoke the Transitions window in the Design Advanced Requests 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 Add to add this transition.

▼ To Add Additional Actions at a Transition

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 creating the transition, the settings will be already 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 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 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 two transitions out of the Poll state shown in the following figure.

State	Transition	Condition	Action
Ground	-> Poll	(SetInternetSystem)	
	-> Error	(check_not_ok)	
		CONDITION : AlarmWarningConfigureError	
	-> Poll	(check_ok)	CONDITION : initSysUpTime
Error			
Poll	-> Down	(!IsNotSysUpTime)	CONDITION : AlarmCriticalOI_sysUpTime
	-> Up	(!IsSysUpTime)	CONDITION : GetSysUpTime CONDITION : undefine_sysUpTime
Up	-> Down	(!IsNotSysUpTime)	CONDITION : AlarmCriticalOI_sysUpTime
	-> Up	(sysUpTimeNotDecrease)	CONDITION : GetSysUpTime CONDITION : undefine_sysUpTime
	-> EverDown	(sysUpTimeDecrease)	CONDITION : undefine_sysUpTime CONDITION : AlarmMinor_sysUpTime
Down	-> EverDown	(!IsSysUpTime)	CONDITION : AlarmClearedOI_sysUpTime CONDITION : AlarmMinor_sysUpTime CONDITION : undefine_sysUpTime
EverDown	-> Down	(!IsNotSysUpTime)	CONDITION : AlarmCriticalOI_sysUpTime
	-> EverDown	(!IsSysUpTime)	CONDITION : undefine_sysUpTime

FIGURE 15-15 IsSnmpSystemEverDown Template

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 then transitions to the Up state. Polling in the Up state checks to determine if `sysUpTime` has decreased. If it has not decreased, the request loops in 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 Order Transitions in the Transitions window.

The condition `undefine_sysUpTime` 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 15-15 shows the completed `IsSnmpSystemEverDown` template as it appears in the Design Advanced Requests textual display. Select the Template → Save As option to save the completed template.

15.9 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 the following figure.

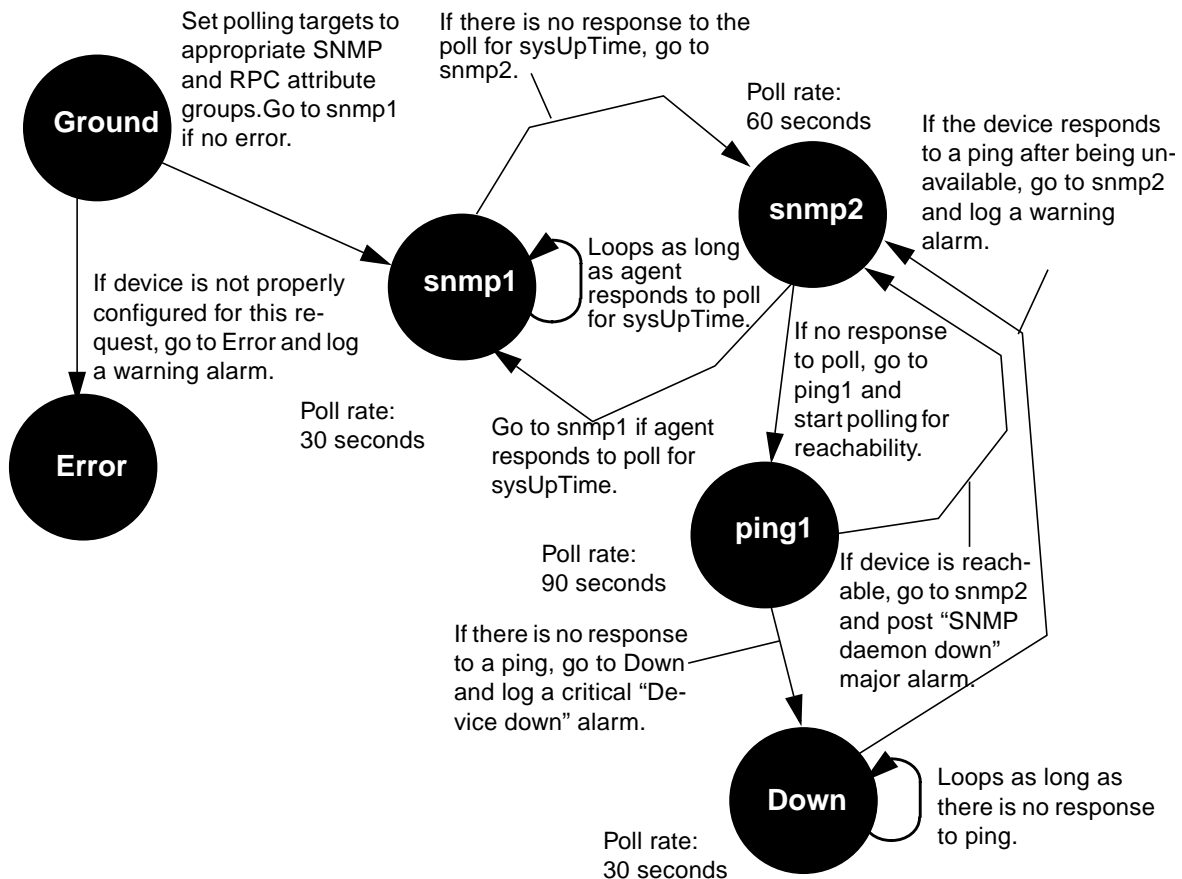


FIGURE 15-16 State Diagram of SnmpPingBackoffReachable Request

15.9.1 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,"distinguishedName");
    $dnstr = AsnToStr($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;
```

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 Network Views.

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 15.6.2.1, “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.

15.9.2 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.

15.10 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 appendix on CMIS filters in *Developing C++ Applications*.)

When designing requests based on event subscription, you will want to correlate this with your use of the following Solstice EM event-handling features.

15.10.1 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 Network Views, 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 `enterpriseSpecificTraps`, you might not want `enterpriseSpecificTraps` to be automatically logged to the `AlarmLog`. The Event Logs is used to define which events are logged to specified logs. The Event Logs is discussed in Chapter 5.

15.10.2 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 11.) 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.

15.11 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. By default, the Solstice EM SNMP trap daemon converts all incoming enterprise-specific SNMP traps into `enterpriseSpecificTraps` with severity set to indeterminate.

You could design an event-subscription request that uses the Nerve Center alarm-logging function to generate more meaningful alarms. For example, suppose that you are interested in the status of certain devices on your network that emit traps with the enterprise identifier of 1.3.6.1.4.1.46. These enterprise-specific traps have specific trap values that are to be interpreted as indicated in the following table.

TABLE 15-2 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	Minor alarm
5	Realtime Clock Failure	Ignore
6	Network Connection Failure	Warning alarm

Your request could subscribe for `enterpriseSpecificTrap` notifications 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 Network Views 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 Network Views icon color.

By default, `enterpriseSpecificTraps` are logged to the Alarm Log and thus affect icon status color in the Network Views. However, in this example we are assuming that you want to control fault status indication for enterprise-specific traps with a request template that logs `nerveCenterAlarms` with a severity that is appropriate to the problem indicated by the trap.

Thus, we want to remove `enterpriseSpecificTraps` from the AlarmLog to eliminate duplication of alarms. To do this, you can use the Event Logs to alter the log discriminator for the AlarmLog to filter out `enterpriseSpecificTraps`. (An example is described in Chapter 3. Also, you may refer to Chapter 5.)

15.11.1 Initiating the Event Subscription

In this example, the trap daemon is mapping incoming enterprise-specific traps to `enterpriseSpecificTrap` event notifications, we will want our request template to listen for incoming `enterpriseSpecificTraps`.

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:

```
$esindx=Subscribe("enterpriseSpecificTrap");
$abccorp_id = "enterprise = 1.3.6.1.4.1.46";
false;
```

The variable `$esindx` 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:

```
$esindx < 0;
```

If a request does transition to the Dead state, you may also want to use the `Exit()` function to cause the request to delete itself. We could define a condition `SelfDestruct` that is called as an action when the transition to the Dead state occurs:

```
Exit();
```

The variable `$abccorp_id` is set to contain the enterprise-identifier of the devices whose enterprise-specific traps this request is to listen for. This will be checked against the `additionalText` field of incoming `enterpriseSpecificTrap` notifications since the `additionalText` field contains the enterprise identifier of the device that generated the trap.

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 ($esindx < 0);
```

After subscribing for `enterpriseSpecificTraps` and checking for subscription error, the request transitions from Ground to a state where it listens for incoming `enterpriseSpecificTraps`, if no error has occurred. Accordingly, you might want to create a state in the template called “Waiting,” to represent this situation.

15.11.2 Listening for Incoming Events

In the Waiting state the request checks for the arrival of incoming `enterpriseSpecificTraps` by testing the following condition:

```
$messType == 0;
```

`$messType` is a system variable that indicates the type of message that “woke” the current state of the request. A value of 0 indicates an incoming event that corresponds to a type that the request has subscribed for. If this condition evaluates to true, an `enterpriseSpecificTrap` has arrived and we will want this to cause a transition to another state, that we might call the “Problem” state, where the request examines the `enterpriseSpecificTrap` 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 15-17.

The `examineTrap` condition illustrated in the following figure is an example of how the request could interpret enterprise-specific traps in the way suggested by the example in TABLE 15-2. The `extract()` function is used to pull out the `additionalText` field of the event. This is then checked against the enterprise identifier in `$abccorp_id`. If `anystr()` detects a match, `extract()` is then used to retrieve the specific trap number, which is in the `probableCause` field.

Alarms are posted with the appropriate severity and the cause of the trap is passed in the `additionalText` field of the `nerveCenterAlarm`.

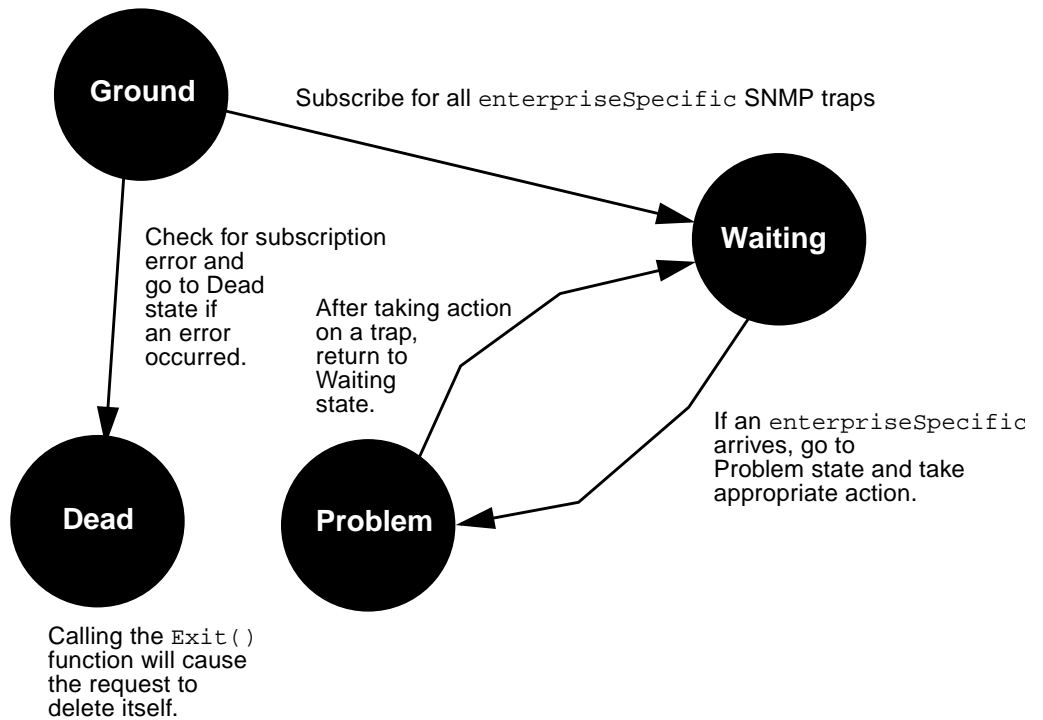


FIGURE 15-17 State Diagram for `IsEnterpriseSpecificTrap` Template

```

$abc_device = FALSE;
$textinfo = extract(&$eventInfo,"additionalText");
$abc_device = anyStr($textinfo,$abccorp_id);
IF ($abc_device == TRUE)
{
    $snum = extract(&$eventInfo,"probableCause");
    IF ($snum == 1)
    {
        $info = strToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{1,critical,\"CPU
Failure\", 3, 1}");
        alarmOi($eventOi,$info);
    }
    ELSE
    {
        IF ($snum == 2)
        {
            $info = strToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{1,critical,\"Power
Supply Failure\", 3, 1}");
            alarmOi($eventOi,$info);
        }
        ELSE
        {
            IF ($snum == 3)
            {
                $info = strToAsn("EM-NC-ASN1.NerveCenterAlarmInfo","{1,critical,\"Fan
Failure\", 3, 1}");
                alarmOi($eventOi,$info);
            }
            ELSE
            {
                IF ($snum == 4)
                {
                    $info = strToAsn("EM-NC-
ASN1.NerveCenterAlarmInfo","{3,minor,\"Overheating\", 3, 1}");
                    alarmOi($eventOi,$info);
                }
                ELSE
                {
                    IF ($snum == 6)
                    {
                        $info = strToAsn("EM-NC-
ASN1.NerveCenterAlarmInfo","{4,warning,\"Network Connection Failure\", 3,
1}");
                        alarmOi($eventOi,$info);
                    }
                }
            }
        }
    }
}

```

When using the `alarmOi()` function to log Nerve Center alarms, `$eventOi` is a system variable that indicates the managed object that is the source of the event. `$info` contains five values that are used to build the Nerve Center alarm. The first

value is the probableCause of the alarm. This is used by the Alarm Service to match clear alarms with the previous alarm that is to be cleared. The second value is the severity of the alarm. The text string, such as “CPU Failure,” becomes the additionalText field of the alarm, which can be viewed in the Alarms. (The last two values are not significant.)

Our completed IsEnterpriseSpecificTrap template is shown in the following figure.

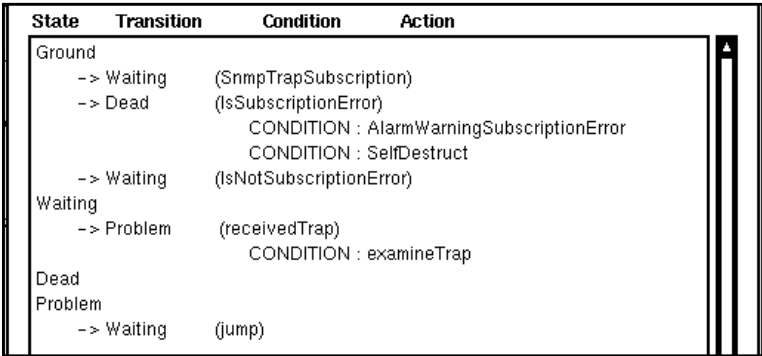


FIGURE 15-18 SNMP Trap Subscription Template

15.12 Requests that Combine Subscription and Polling

Event subscription and polling can be combined in a single template. An example of a template that does this is the SnmpLinkUpDownTrap sample template. This template subscribes for incoming linkDownTrap notifications. When a linkDownTrap is received, the request transitions to the LinkDown state which initiates polling in order to count the elapsed time until a linkUp trap is received for the downed interface. If a linkUpTrap notification is not received within the polling interval (20 seconds), a critical Nerve Center alarm is posted against the target SNMP device. If a linkUpTrap notification is received within the polling interval, no alarm is logged. A state diagram for the SnmpLinkUpDownTrap template is shown in the following figure.

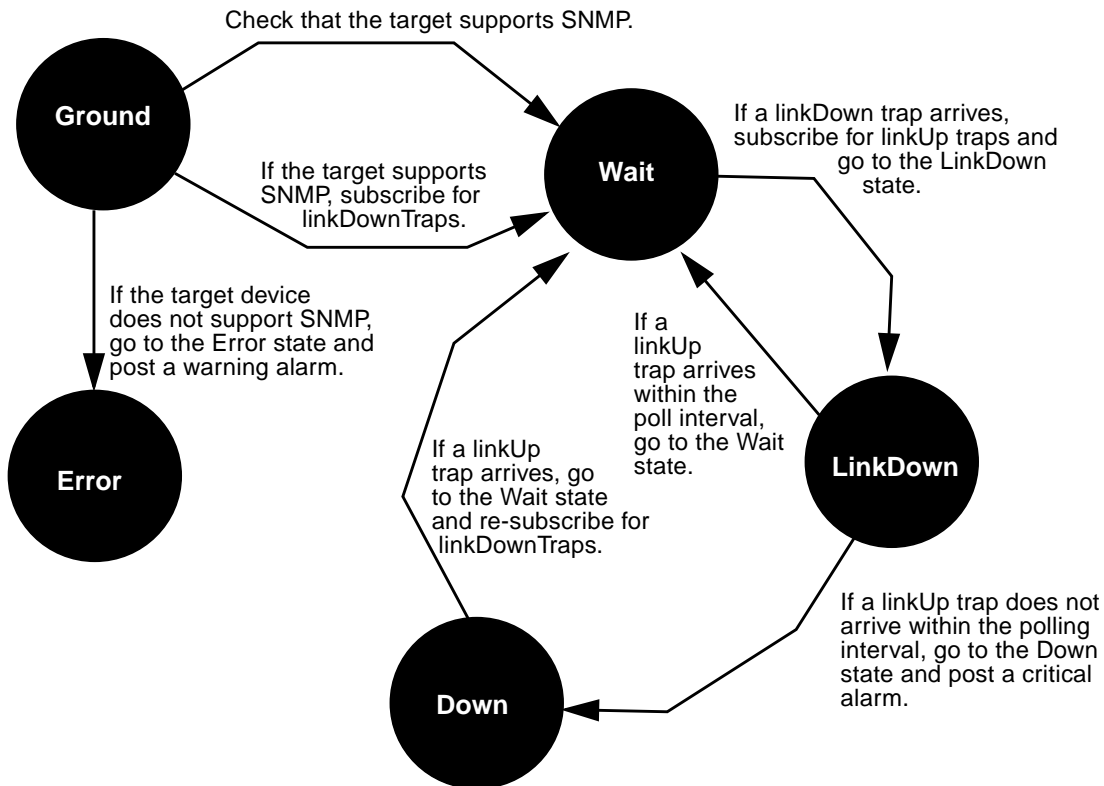


FIGURE 15-19 `SnmpLinkUpDownTrap` Template State Diagram

Since a request is being used to control logging of alarms for linkDown traps in this example, you would not want linkDownTrap notifications to automatically lead to a change in Network Views status color. However, by default, the linkDownTrap notifications are logged to the Alarm Log. Since, by default, the Alarm Log is monitored by the Alarm Service, which controls fault status indication in the Network Views, these trap notifications thus affect the fault status color of icons in the Network Views. For this template, then, you would probably want to use the Event Logs to add linkDownTrap and linkUpTrap notifications to the list of events excluded from the Alarm Log. You might create a separate log, such as LinkLog, to maintain an historical record of incoming linkDownTrap and linkUpTrap notifications. For information on accomplishing this task, see Chapter 5.

15.12.1 Checking for a Correct Target

The first transition out of the Ground state uses the `get_snmp_agent_dn` condition to ensure that this request has been launched at a device that is configured as supporting SNMP. The request will log a warning alarm if this condition is not satisfied.

To test for SNMP support, the `get_snmp_agent_dn` condition examines the contents of the RCL system variable `$pollFdnSet`.

When a request is launched against a target device selected in the Network Views, distinguished names (DNs) for the set of managed objects configured for that device, representing the manageable attributes of the device, are loaded into `$pollFdnSet`. The first member of `$pollFdnSet` is loaded into `$pollfdn`. The `get_snmp_agent_dn` condition stores the value of `$pollfdn` into `$save_pollfdn`. This provides a pointer to a managed object for the target device which can be used as the target for alarms logged against this device. Alarms must be logged against one of the managed objects in the target's `$pollFdnSet` for the Alarm Service to know which topology node is the target of the alarm.

The `get_snmp_agent_dn` condition also searches the DN's in `$pollFdnSet` for a match on "cmipsnmp." If there is a match, the target has been configured with a `cmipsnmpProxyAgent`, representing the SNMP agent on the device.

```
$save_pollfdn = $pollfdn;
$num = numElements(&$pollFdnSet);
$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 = $dn;
        $count = $num +1;
    }
    $count = $count +1;
}
false;
```

If a match was found, `$res` is TRUE, otherwise it is FALSE.

This condition ends with the statement “false;” to ensure that the transition from the Ground state to the Wait state does not yet occur. The request then falls through to the next transition. If the device was configured for SNMP support, the `check_ok` condition evaluates to true:

```
$res == TRUE;
```

The request then subscribes for `linkDownTrap` notifications and transitions to the Wait state; otherwise, it transitions to the Error state and posts a warning alarm to indicate the request was launched against a device that is not configured to support it. The subscription for `linkDownTrap` notifications occurs in the `subscribe_to_linkdown` condition:

```
$handle =  
subscribeOi("linkDownTrap", "cmipsnmpProxyAgent", $pollfdn);
```

`subscribe_to_linkdown` Condition

The `subscribeOi()` function here limits the subscription to events generated by the device associated with `$pollfdn`. The type of event subscribed for is indicated by “`linkDownTrap`”. The condition `is_event` will be true if a subscribed event arrives:

```
$messtype == 0;
```

This condition thus defines the transition to the LinkDown state. Since we want to now test for the arrival of follow-on `linkUp` traps, we terminate the subscription for `linkDown` traps by passing `$handle` to the `unsubscribe` function:

```
unsubscribe($handle);
```

In the transition to the Wait state the request also now subscribes for `linkUp` traps:

```
$handle =  
subscribeOi("linkUpTrap", "cmipsnmpProxyAgent", $pollfdn);
```

The Wait state has a polling interval of 20 seconds. Polling is activated by the test for an attribute value (the value of the `map` attribute) in the `is_poll_timeout` condition:

```
$x = map;  
$messType != 0;
```

This condition is used to define the transition from the Wait state to the Down state. If the Wait state is woken up by the arrival of a `linkUpTrap` notification, `$messType != 0` evaluates to false and this transition does not occur. (`$messType` is 0 only if the message that woke up the state is subscribed-for event notification.) If the polling interval elapses and no `linkUpTrap` event has arrived, then `$messType != 0` evaluates to true and the transition to the Down state does take place.

If `is_poll_timeout` evaluates to false, the request then evaluates the next transition out of the Wait state, which is defined by the `is_event` condition. If a `linkUpTrap` notification has arrived, this condition evaluates to true and the request transitions back to the Wait state. In the transition back to the Wait state, the request drops the subscription for `linkUpTraps` (by executing the `unsubscribe` condition) and re-subscribes for `linkDownTrap` notifications (by executing the `subscribe_to_linkdown` condition).

On the other hand, if the request transitions to the Down state, indicates that the polling interval expired before the arrival of a `linkUp` trap. In this case, a critical alarm is posted against `$save_pollfdn` (a managed object corresponding to the target device) and the request continues to wait for the arrival of a `linkUp` trap.

15.13 Building Request Definitions

The meta data browser presents data in a three level hierarchy:

- document
- class
- attributes in each class

You can create multiple thresholds for multiple attributes provided the attributes are within the same document and class. When any one of the thresholds is crossed a notification is sent.

You can name the request definition, enter a description of the request definition based on a list of the following threshold operators:

- equal to threshold
- not equal to threshold

- less than threshold
- less than or equal to threshold
- greater than threshold
- greater than or equal to threshold
- value changed
- value increased by threshold
- value decreased by threshold
- value increased by more than threshold
- value increased by less than threshold

In order for you to organize the requests, each Request Definition can belong to a group that you define by the following methods:

- Create request groups and add request definitions to the group (a Request Definition defaults to the root group if none is specified)
- Modify groups
- Delete groups
- Provide a default name for the request
- Create a new request from an existing request (similar to a “save as” option)
- Delete request definitions
- Modify request definitions

Debugging Request Templates

There are several facilities available to you in debugging Nerve Center request templates:

- The Design Advanced Requests tool does RCL syntax checking when you attempt to save a newly created, or modified, condition. However, the Design Advanced Requests tool will not catch possible runtime errors.
- The Basic Requests tool (accessible from the Network Views menu) lists the requests currently executing in the Nerve Center. If you select a request and click Examine, the Request Examine window is invoked. This window displays the values of variables in the request as it is executing. (Refer to Chapter 4 in *Managing Your Network* for information on the Basic Requests tool.)

The *Solstice Enterprise Manager* (Solstice EM) `em_debug` utility also provides facilities that are useful in debugging templates.

This chapter describes the following topics:

- Section 16.1 “Nerve Center Debugging Agents” on page 16-1
- Section 16.2 “Activating RCL Print Statements” on page 16-2
- Section 16.3 “Turning Off Debug Agents” on page 16-3

16.1 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.
- `nc_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 Design Advanced Requests textual display.

You should invoke the debugging agents before launching the request in the Network Views.

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.

16.2 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 evaluated to `true`, and the request transitions out of the state, the subsequent transitions in that state are not evaluated.

16.3 Turning Off Debug Agents

You can turn off a particular debugging agent by entering the following command:

```
% 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 *"
```


Building Templates for SunNet Manager Event Requests

Solstice Enterprise Manager (Solstice EM) 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.

This chapter describes the following topics:

- Section 17.1 “RPC Agents” on page 17-1
- Section 17.2 “Nerve Center’s SNM Event Request Capability” on page 17-3
- Section 17.3 “SNM Alarms” on page 17-4
- Section 17.4 “Building SNM Event Request Templates” on page 17-6

17.1 RPC Agents

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 Solstice 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 Solstice EM MIS by Solstice EM's SNM Event Forwarder (em_snmfwd).

As illustrated in the following figure, RPC proxy agents use Remote Procedure Call (RPC) protocol (over IP) to communicate with the Solstice EM MIS. However, an RPC proxy agent may use a different management protocol in gathering information from other agents. In the example in the following figure, SNM's Simple Network Management Protocol (SNMP) proxy agent (na.snm) is used to manage devices that support the SNMP protocol.

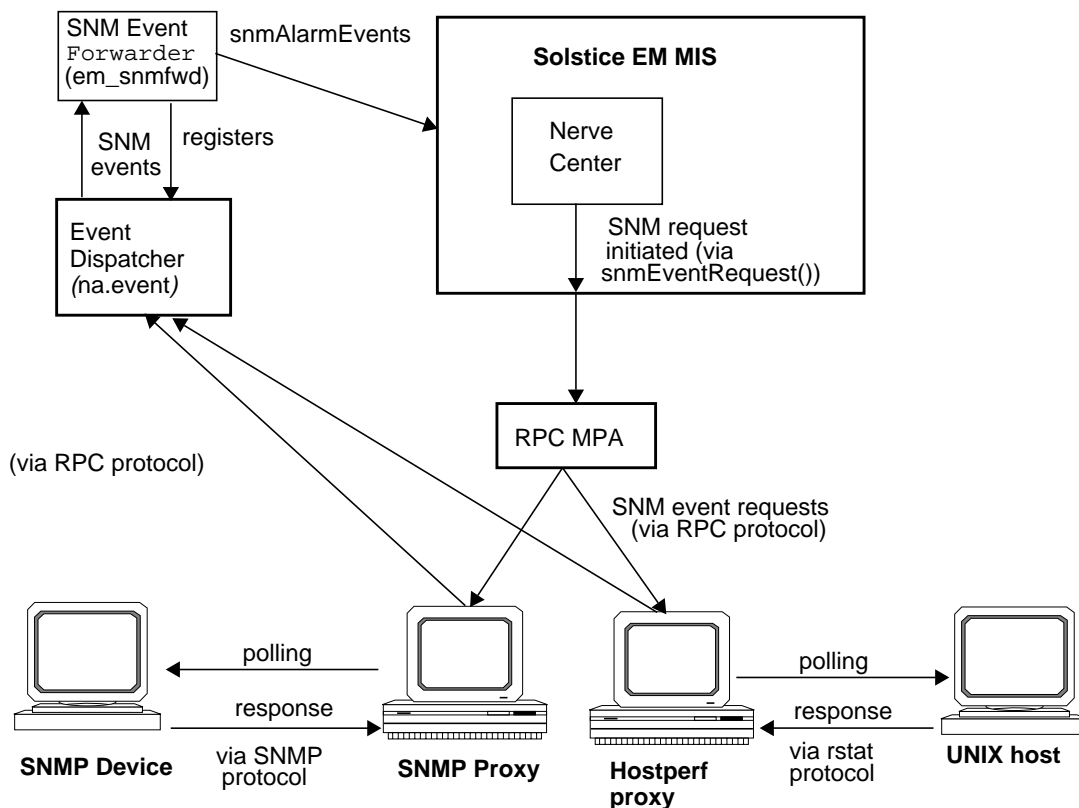


FIGURE 17-1 Using SNM Event Requests with Solstice EM

For information on the installation of RPC proxy agents, refer to Chapter 6 in *Installation Guide*.

Note – SNM events that are received by SunNet Manager Consoles managing segments of your network can also be forwarded to the Solstice EM MIS using Cooperative Consoles. This type of distributed management scenario is described in Chapter 7.

For general information on using SunNet Manager RPC agents with Solstice EM, see Chapter 6.

17.2 Nerve Center's SNM Event Request Capability

The Nerve Center module in the MIS contains the request-handling capabilities of Solstice EM. Nerve Center requests are based on request templates, which are built using the Design Advanced Requests application. A key building block in request templates are request *conditions* — sets of instructions defined using the Solstice EM Request Condition Language (RCL). RCL provides two built-in functions, `snmEventRequest()` and `snmKillRequest()`, for starting and stopping SNM event requests. For general guidance in building request templates, read Chapter 15. For information on the RCL functions, see Chapter 22.

SNM event requests can be launched from the Solstice EM management station using the request-handling capabilities of the Solstice EM Nerve Center. Request templates built using the Solstice 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 is 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 SNMP event if it occurs for the specified attribute.
- The SNMP priority of an alarm generated if the threshold obtains—the possible priorities for SNMP events are High, Medium, or Low. These correspond to `perceivedSeverity` values of Solstice EM alarms as indicated in TABLE 17-1.
- Polling interval (in seconds)—the delay between polls of the target object by the SNMP agent or proxy.
- The number of times the device is 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 SNMP event if the `capacity` attribute value is greater than 90% on the target file system.

Once the Nerve Center has initiated the SNMP request, polling of the managed resource at the specified intervals is handled by the SNMP proxy rather than the Solstice EM Nerve Center, thus minimizing network traffic and the polling work required of the Nerve Center.

When a SNMP agent or proxy 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 SNMP agents and proxy agents can be found in the *Site/SunNet/Domain Manager Reference Manual*.

17.3 SNMP Alarms

When a critical threshold defined in an SNMP request is detected by the SNMP agent, a response—called an *event* in SNMP terminology—is sent via RPC protocol to the SNMP Event Dispatcher (`na.event`). The SNMP Event Forwarder daemon (`em_snmfwd`) registers with the SNMP Event Dispatcher to receive incoming SNMP events. SNMP 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 a `snmAlarmEvent`, which will be sent to the Solstice EM MIS. The Event Forwarder maps SNM event severities to the `perceivedSeverity` values used by the Alarm Service in the manner indicated in the following table.

TABLE 17-1 Mapping of SNM Event Severities

SNM Event Severity	<code>perceivedSeverity</code> 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 the table.
- `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 is 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 in the *Management Information Server (MIS) Guide*.

As `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 Network Views. By default, only alarms logged to the AlarmLog affect fault status color in the Network Views. The Alarm Service is a module in the Log Server 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, see Chapter 4.

However, a request that listens for incoming `snmAlarmEvents` can use the RCL alarm-logging functions to post appropriate `nerveCenterAlarms` to the Alarms Log. 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.

17.4 Building SNM Event Request Templates

An example of a Nerve Center request template that initiates a 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, a 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 17-1, 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. See the following figure.

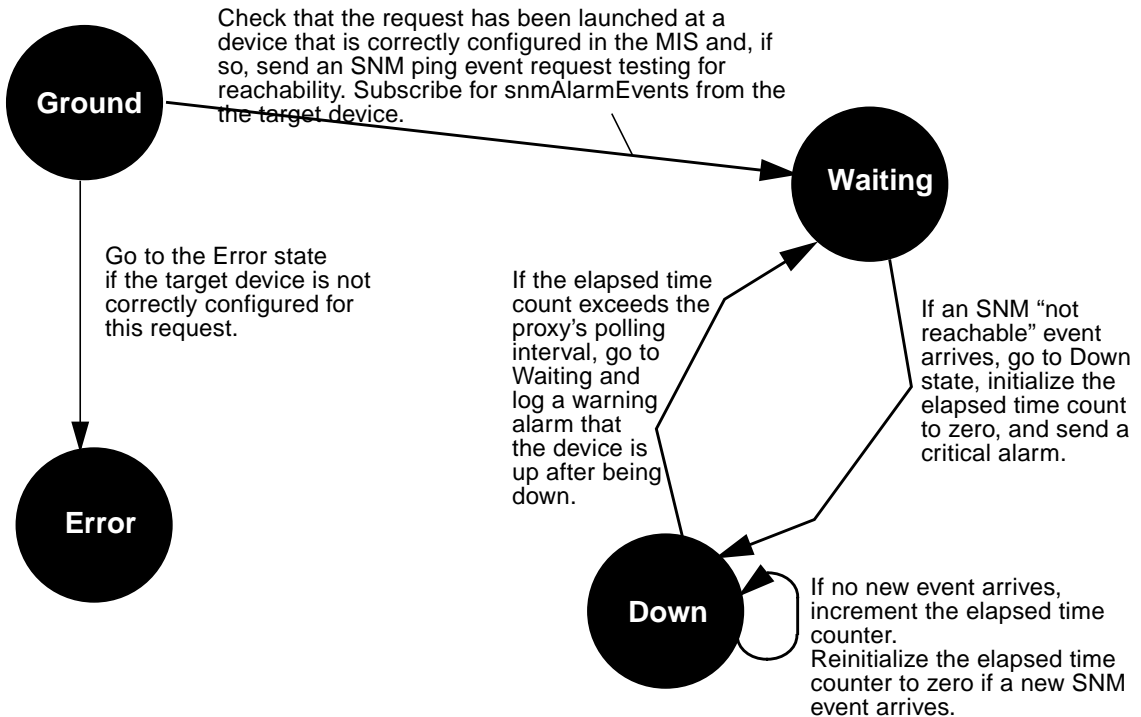


FIGURE 17-2 State Machine Diagram for DeviceReachablePing Template

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;
```

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 `$pollFdnSet` variable when the request is launched against a target device in the Network Views. The condition then uses a WHILE loop to examine the distinguished name (FDN) pointing to each such object to determine if the device is manageable via RPC. If the device is manageable by RPC, the RPC proxy table for the device (which “contains” under it the various RPC agent attribute groups supported by that device) will be represented in the `$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 which will, then, 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.

17.4.1 Subscribing for SNM Events

The subscription for `snmAlarmEvents` occurs in the following condition:

```
subscribeOi("snmAlarmEvent", "{}", $dn);  
false;
```

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.

17.4.2 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;
```

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.

17.4.3 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 Network Views. The string passed to the `alarmStr()` function appears in the `additionalText` field for that alarm in the Alarms tool.

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;
```

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;
```

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 Alarms tool. If you wanted to implement 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 to 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.

Building Advanced Requests

Requests are the series of activities through which the *Solstice Enterprise Manager* (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*. Design Advanced Requests is a tool that allows you to create request templates.

This chapter describes the following topics:

- Section 18.1 “Components of Request Templates” on page 18-1
- Section 18.2 “Using the Design Advanced Requests Tool to Build Nerve Center Templates” on page 18-4
- Section 18.3 “Conditions” on page 18-9
- Section 18.4 “States” on page 18-11
- Section 18.5 “Transitions” on page 18-12
- Section 18.6 “Actions” on page 18-16
- Section 18.7 “Poll Rates” on page 18-20
- Section 18.8 “Modifying the Mapping of Colors to Severities” on page 18-23
- Section 18.9 “Graphical State Diagram Display” on page 18-24

18.1 Components of Request Templates

States, conditions, and poll rates 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 a request’s 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 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 20.

The alarm logging activity of Nerve Center affects the Alarm Manager and Log Manager, which displays a summary of alarms in one or more logs, and the Log Entries, which displays log records for events you have chosen to log. Both the Alarms and Log Entries are described in Chapter 11 of *Managing Your Network*.

Nerve Center alarm logging also has an impact upon the color of icons in the Network Views. The MIS Alarm Service module monitors the alarm logs and updates the color of icons in the Network Views 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 Chapter 3. Nerve Center controls this mapping and you can change it through the Request Designer Edit ➔ Severities option, described in Chapter 3.

18.1.1 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. The following figure shows a very simple, yet valid, example that illustrates request-related concepts.

Note – The “severities” that attach to template states in the Design Advanced Requests tool do not control the fault status indication (icon color) of devices in the Network Views. The severities of request states only affect the color attached to states in the Request Designer graphical display. Fault status color of devices in the Network Views 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 in Chapter 22.

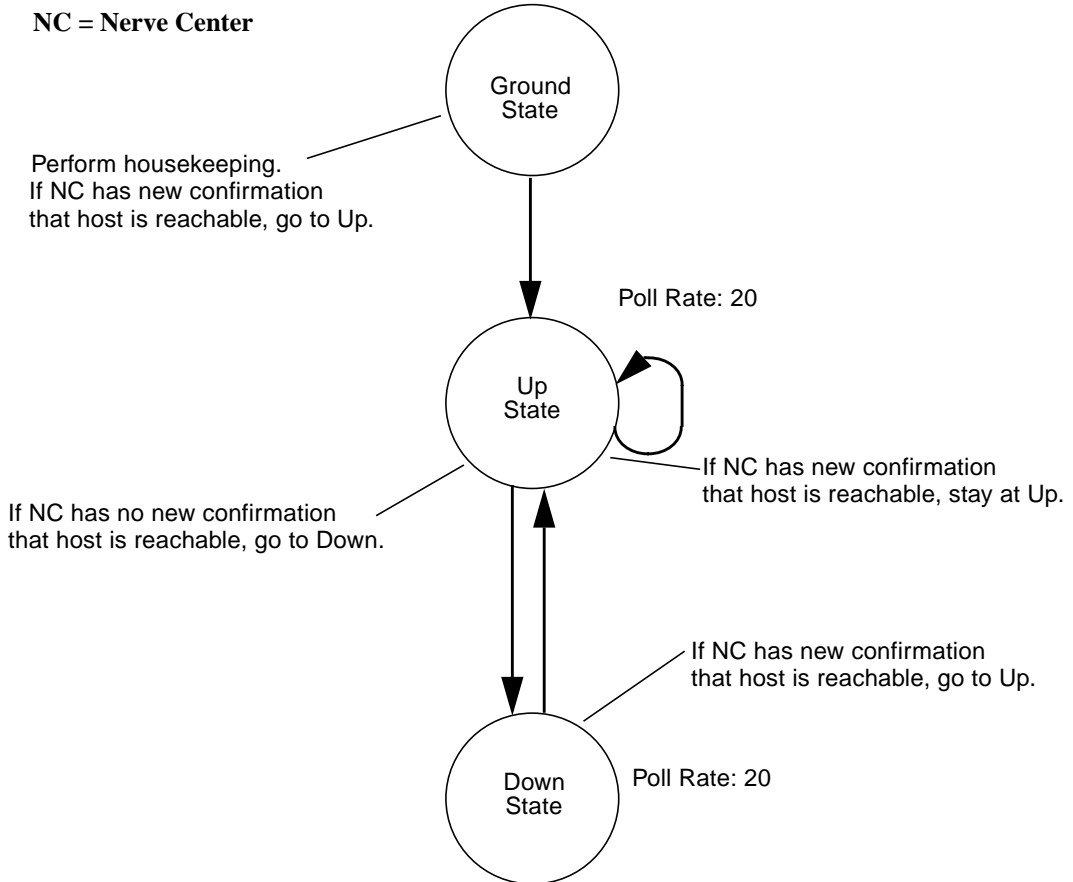


FIGURE 18-1 Request Example with Poll Rates and Severities

18.2 Using the Design Advanced Requests Tool to Build Nerve Center Templates

Before embarking on building a request template from scratch, use the Design Advanced Requests tool to examine the sample request templates supplied with Solstice EM. 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.

▼ To Create a Request Template

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 Design Advanced Requests tool, as described below in Section 18.2.1, “Starting Request Designer.”**

The following steps all involve the use of the Request Designer.

3. **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.

4. **Create the states you need.**

States are specific to each template, that is, you have to create new states for each template.

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. A condition is used to define when the transition is to take place.

6. **Add actions, if any, that you want to occur when a transition takes place.**

7. **Name the request template and enter a brief description of it.**

8. **Save the template.**

Note – You can save an incomplete template, to continue work on it at a later date, through the Request Designer 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 Advanced Requests window from the Network Views 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 tool.

18.2.1 Starting Request Designer

To use the Request Designer tool, a Solstice EM MIS must be running and the Request Designer must be able to communicate with it.

You can start the Request Designer tool by selecting Administration → Request Designer. Also, the tool is brought up if you click Create or Modify in the Advanced Requests tool. (The Advanced Requests tool is described in Chapter 4 of *Managing Your Network*.)

You can also invoke the Request Designer tool, and have it connected 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 logged on as a non-root user, you receive a Login window if password authentication has been activated for Solstice 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 Security tool.

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 tool comes up in its text mode. See Section 18.9, “Graphical State Diagram Display” for instructions on the using the graphical display.

18.2.2 Creating a New Nerve Center Template

The Request Designer tool provides two ways for creating a new Nerve Center template:

- You can create one from scratch, or
- You can use an existing template as a starting point, make modifications, and then save it under a new name.

Selecting File → New sets up the main Request Designer tool canvas for building a new template from scratch. The canvas is blank except for the presence of the Ground state. Do not delete the Ground state as every request must start from the Ground state.

Until you save the template, “NoName” is displayed as the template name in the footer.

Adding states, writing conditions, and defining transitions between states are the main work in the building of a Nerve Center template. These tasks are described below under Section 18.4, “States,” Section 18.5, “Transitions,” and Section 18.3, “Conditions.”

After adding states, and transitions between states, you can save your work by selecting the File → Save option. Saving the completed template loads it into the MIS. The name and description you enter when saving the template are displayed in the list of templates available to users in the Advanced Requests tool.

If you have not completed the template but want to save it to continue work on it later, you can use the File → Export Current option to save the unfinished template to an ASCII text file. To load this template back into the Request Designer workspace later, use the File → Import option. When prompted for a filename, specify the filename of the ASCII file to which the template was previously exported.

18.2.3 Modifying an Existing Nerve Center Template

Use the File → Open option to select the template you want to modify.

Note – You cannot save changes to a template under the same name if there are any running requests in the MIS based on that template. Invoke the Advanced Requests tool from the Network Views menu to determine if there are any requests running based on the template you wish to modify.

After making changes in the template, those changes are saved to the original template in the MIS if you select the File → Save option.

Existing templates are a convenient starting point for the creation of new templates. In this case you do not have to worry about the existence of requests based on the original template running in the MIS. You can save the template under a new name even if there are running requests based on the template you opened to start your template creation. Use the File ➔ Save As option, after you have completed your modifications, to create a new template in the MIS.

18.2.4 Deleting Nerve Center Templates

Use the File ➔ Delete option to delete a Nerve Center template from the MIS.

Note – You cannot delete a template if there are any requests running in the MIS that are based on that template. Invoke the Advanced Requests tool from the Network Views menu to determine if there are any running requests based on the template you wish to delete.

18.2.5 Exporting Nerve Center Templates to an ASCII File

Nerve Center templates, conditions, and poll rates can be exported to an ASCII file. If you export a request template to an ASCII file and print it out, you may find this helpful in analyzing the overall structure of the template. Also, if you want to copy a template from one MIS to another, one way to do this is to export the template to ASCII file from one MIS and then import that file into the other MIS.

There are three ways to export Nerve Center templates, conditions, and poll rates:

- You can export Nerve Center templates (and their components) using the Request Designer' File ➔ Export option, the imported components are loaded into the MIS.
- You can export Nerve Center templates (and their components) using the `em_ncexport` command-line utility. The use of this utility is described in Chapter 19.
- You can export Nerve Center templates (and their components) using the Request Designer' File ➔ Export Current option, the template is loaded into Design Advanced Requests but not into the MIS.

If you select the File ➔ Export option, the Export Customized window is displayed (FIGURE 18-2). By clicking the appropriate button at the top of the window, you receive scrolling lists of Templates, Conditions, or Poll Rates which you can select for

inclusion in the export to a specified ASCII file. In the following figure the SetInternetSystem condition has been selected for export to a file named myconditions.

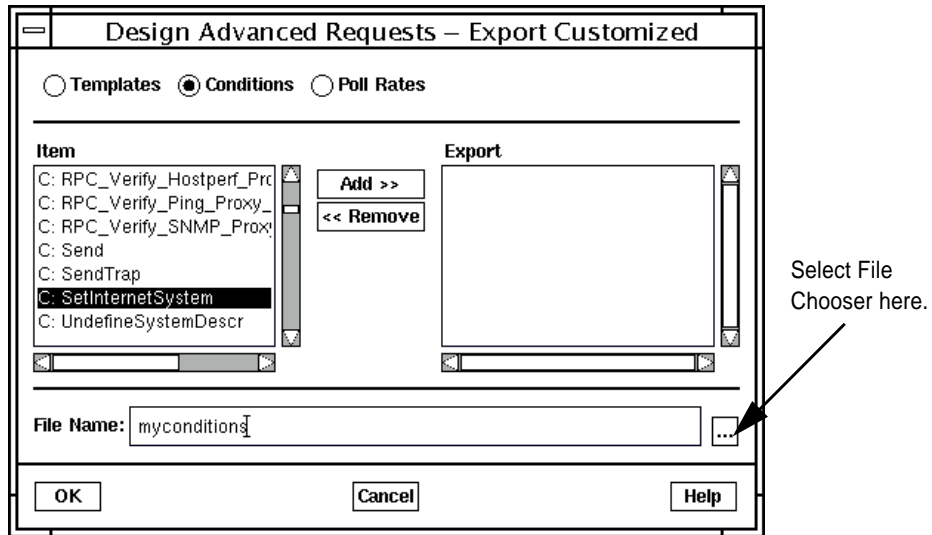


FIGURE 18-2 Example of Export to ASCII File

18.2.6 Importing Nerve Center Templates from an ASCII File

Use the File Import option to import Nerve Center templates and their components from an ASCII file to which they were previously saved. 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 Design Advanced Requests but not into the MIS.

18.3 Conditions

A *condition* is a script that contains one or more statements written in Request Condition Language (RCL). Conditions are used for two different functions in Nerve Center templates:

- To define when state-to-state transitions take place—this is described below in Section 18.5, “Transitions.”
- As actions to execute when a request transition takes place—this is described in Section 18.6, “Actions.”

Each RCL statement ends with a semicolon. For example, the following RCL statement logs a nerveCenterAlarm with a severity of critical:

```
alarmStr(1,"Device is down");
```

RCL conditions may also contain control structures, using expressions such as IF, IF ELSE, WHILE, or FOREACH, with statements contained within them. For example, the following control structure counts the number of events from a specified device:

```
IF ($eventOi = $pingFdn)
{$ping_response_count = ping_response_count+1;}
```

Conditions cannot be modified or deleted once they are included in Nerve Center templates that have been stored in the MIS. A condition can be modified or deleted only if no existing request template has a reference to it. If a condition that is referenced by any other request templates must be modified or deleted, it must first be unreferenced or excluded from all other request templates.

Information on RCL can be found in the following locations:

- Components of the RCL language are described in Chapter 20.
- RCL system variables that can be used in building conditions are described in Chapter 21.
- RCL also includes built-in functions that can be used to build RCL conditions. The RCL functions are described in Chapter 22.
- Conditions for several sample request templates are described in Chapter 15.
- Conditions used in request templates that initiate SunNetManager event requests are discussed in Chapter 17.

The Request Designer Conditions window provides a text canvas in which RCL conditions can be composed. You can invoke the Conditions window (shown in FIGURE 18-3) from the main Request Designer window by clicking Conditions, or by selecting the Edit → Conditions menu option.

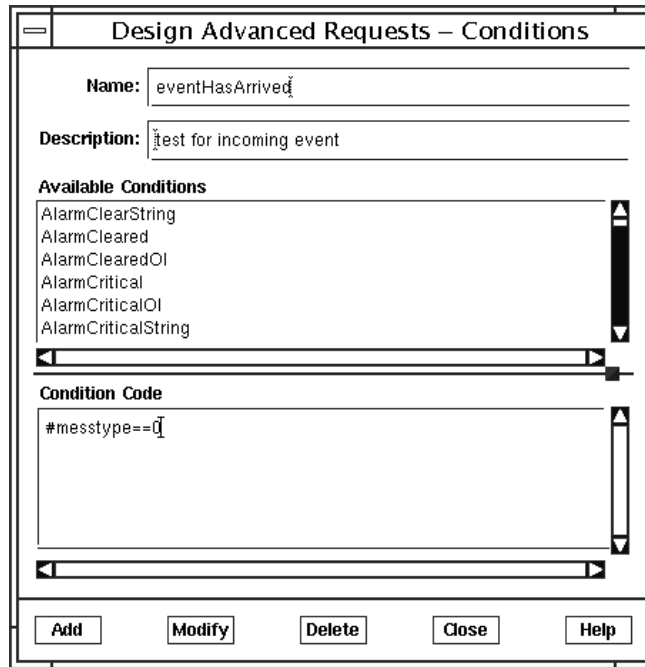


FIGURE 18-3 Viewing RCL Conditions in the Conditions Window

You can use the Conditions window to do the following:

- Add a new condition to the MIS

You create a new condition by saving your condition under a name that is not already in use for templates stored in the MIS.

- Modify an existing condition

Double-click on a condition name in the scrolling list in the Conditions window to open an existing condition.

- Delete an existing condition from the MIS

Note – Conditions cannot be modified or deleted once they are included in Nerve Center templates. If you want to modify a condition that is used in an existing Nerve Center template, you will need to remove the action or transition in which that condition occurs before you can save your modifications to the MIS.

18.4 States

States are used to represent the request's current knowledge of the state of a device, such as a Waiting state if the request is waiting for incoming events from the device, or a Down state to represent the situation where information has been received indicating that the device is unavailable. A request template is a finite state machine, consisting of multiple states and transitions between states. Every request begins in the Ground state; but Ground is the only state required for every request template.

18.4.1 Adding States to a Nerve Center Template

▼ To Add a State to a Request Template

1. **From the Request Designer main window invoke the States window by clicking States or selecting the Edit → State menu option.**
2. **Type in the name and description for the new state and select a poll rate.**
You can attach a different poll rate to each state, if desired.
3. **Select a severity.**
This does not affect logging of alarms. This “severity” only selects the color used to represent the state in the Request Designer graphical display.
4. **Clicking Add inserts this state into your template.**

18.4.2 Modifying States in a Nerve Center Template

▼ To Change an Existing State in a Request Template

1. From the Request Designer main window invoke the States window by clicking States or selecting the Edit → State menu option.
2. Select the state that you wish to change by typing its name in the name field or selecting the state in the tabular display.
3. Make your proposed changes to the fields other than the name field. And then click Modify to make the changes take effect.

18.5 Transitions

A *transition* occurs when a request moves from one state to another. A condition is used to define when a transition is to take place. The transition occurs if and only if the condition evaluates to true. A transition may cause a request to loop back from a state to that same state, or move to a different state. You may define multiple transitions out of a given state. There can be more than one transition defined from state A to state B. See the following figure.

State	Transition	Condition	Action
Ground	-> Waiting	(SnmpTrapSubscription)	
	-> Dead	(IsSubscriptionError)	
		CONDITION : AlarmWarningSubscriptionError	
		CONDITION : SelfDestruct	
	-> Waiting	(IsNotSubscriptionError)	
Waiting	-> Problem	(receivedTrap)	
		CONDITION : examineTrap	
Dead			
Problem	-> Waiting	(jump)	

FIGURE 18-4 Order of Transitions in a Template

Transitions out of a given state are evaluated by Nerve Center in the order they occur in the Request Designer tabular display. In the example in FIGURE 18-4, the transition from Ground to Waiting defined by the `SnmpTrapSubscription` condition is evaluated before the Ground to Dead transition defined by `IsSubscriptionError`.

Transitions out of a state are only evaluated if the state is “awake.” An “awake” state can occur if:

- The request has previously subscribed for specified events and an event of the specified sort has arrived.
- Conditions defining transitions out of that state poll for the current value of attributes known to the MIS.

18.5.1 Creating New State-to-State Transitions in a Template

▼ To Add a Transition to the Template

1. Invoke the Transitions window by clicking Transitions or selecting the Edit → Transition menu option.
2. Define the state the transition is *from*, and the state the transition is *to*.
3. Select a condition to test to determine if the transition is to occur.
4. You may also specify one or more actions to be executed if the transition occurs. (This is optional.)

18.5.2 Deleting Transitions from a Template

▼ To Delete a Transition From a Template

1. Invoke the Transitions window by clicking Transitions or selecting the Edit → Transitions menu option.
2. Define the state the transition is *from*, and the state the transition is *to*.
3. Select the condition that defines when the transition is to occur.

4. Select **<none>** for the Action field.
5. Click **Delete** to delete the transition.

18.5.3 Reordering Transitions

▼ To Change the Order in Which the Transitions Out of a Given State are Evaluated

1. Invoke the Transitions window by clicking **Transition** or by selecting the **Edit → Transitions** menu option.
2. Select the state whose transitions you wish to reorder on the **From** field.
3. Click **Order Transitions** to invoke the **Order Transitions** window.
4. You can change the order of the transitions out of that state by selecting a transition in the list and then clicking **Move Up** or **Move Down**, as shown in the following figures.

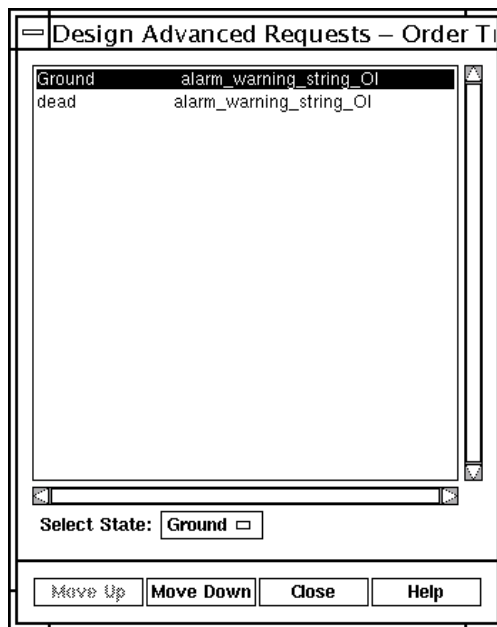


FIGURE 18-5 Reordering State Transition - Move Up

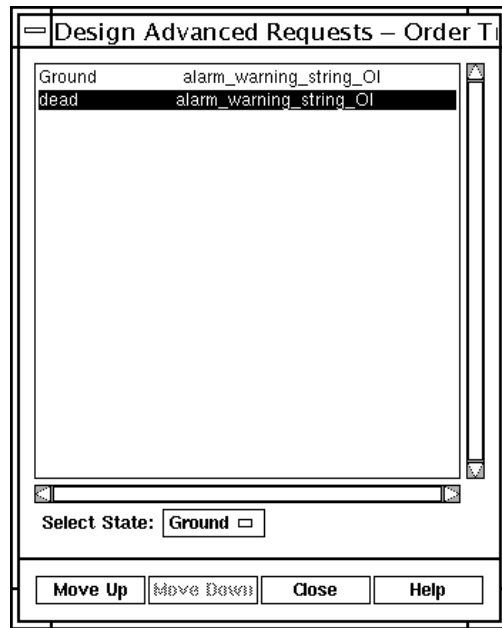


FIGURE 18-6 Reordering State Transition - Move Down

18.6 Actions

Actions are executed when a request moves from one state to another state (or loops back to the same state) in a state transition. The Action menu in the Transitions window is used to select which sort of action you want the request to execute when the transition occurs. The Action menu options are described in the following table.

TABLE 18-1 Action Menu Items

Action	Description
<none>	No action taken.
UNIXCMD	The name of command with any required parameters. For example, for <code>netstat -rn</code> , you enter <code>netstat</code> in the Command field and <code>-rn</code> in the Arguments field.
MAIL	An electronic mail address and message. For example, <code>verma@halcyon</code> in the Address field and <code>CPU usage exceeded 90%</code> 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).

RCL system variables or variables that you define can be used to define data that is to be passed in an email message (see the following figure). RCL variables can also be used as parameters in a UNIX command.

Action: MAIL ☐

Name:

Address: mgr@Eng

Message: \$hostname is down

FIGURE 18-7 Use of RCL Variables in Mail Action

18.6.1 Adding Actions at a Transition

The Transitions window allows you to add or delete actions at a transition.

▼ To Add an Action at a Transition

1. If the Transitions window is not already displayed, invoke it by clicking Transitions or by selecting the Edit → Transitions menu option.
2. Select the transition to which the action is to be added by setting the From, To, and Condition fields to match the target transition.
3. Select the appropriate action type from the Action menu.

- a. If you selected MAIL, type in the email address in the Address field and the message in the Message field.

User-defined or system RCL variables may be used in the message field.

- b. If you selected UNIXCMD, fill in the Command field and enter the required arguments for the command in the Arguments field.

User-defined or system RCL variables may be used in the Arguments field.

- c. If you selected CONDITION, select the name of the condition to be executed from the scrolling list of conditions that are available in the MIS.

This is inserted into the Name field after you have selected it. In the following figure, the `alarm_warning_string_OI` condition has been selected for adding as an action to the transition from the Ground to Dead state defined by the `result_equal_false` condition.

Design Advanced Requests – Transitions

Order Transitions

To	Condition	Action	Arg0	Arg1

From: Ground

To: dead

Condition: alarm_warning_string_OI

Action: CONDITION

Name: result_equal_false

Command:

Arguments:

Add

Delete

Modify

Close

Help

FIGURE 18-8 Adding a Condition as an Action at a Transition

4. Clicking Add inserts this action into the transition.

You can continue to add additional actions at the same transition by repeating this procedure. Actions are added to the transition in the order they are created.

18.6.2 Deleting Actions at a Transition

If there is more than one action at a transition, you can only delete the last action in the transition in a single step. If there are multiple actions at a transition and you want to delete an action other than the last one, you must first delete all of the actions that are listed after the target action in that transition.

▼ To Delete the Action That is the Last Action in the List of Actions at a Transition

1. If the Transitions window is not already displayed, invoke it by clicking Transitions or by selecting the Edit → Transitions menu option.
2. Select the transition from which the action is to be deleted by setting the From, To, and Condition fields to match the target transition.
3. Select the appropriate action type from the Action menu.
4. If the action to be deleted is a condition, select the target condition from the scrolling list.
5. Click Delete.

Note – If you select <none> as the action type, clicking Delete removes the entire transition.

18.6.3 Reordering the Actions at a Transition

Reordering actions in a transition is done by deleting actions from the bottom of the list of actions, and then adding them back in the desired order. If action B follows action A in a transition but you want action B to be first and action A to follow, you will need to delete both action B and action A, and then add them both in the proper order.

18.7 Poll Rates

The *poll rate* is the length of delay before the first poll and interval between polls thereafter. Poll rates attach to states. Different states can thus have different poll rates. If the transitions out of a state require polling for attribute values, Nerve Center schedules required polls when a request first enters that state. Poll rates supplied with the product are listed in the following table.

TABLE 18-2 Poll Rates

Name	Interval (secs.)
Poll	20
poll30	60
Fast	60
Moderate	300
Medium	900
Slow	3600
VerySlow	21600
default_rate	300
1secs	1
5secs	5
10secs	10
20secs	20
30secs	30
40secs	40
1min	60
2min	120
3min	180
4min	240
5min	300
10min	600
15min	900
20min	1200

TABLE 18-2 Poll Rates *(Continued)*

Name	Interval (secs.)
30min	1800
40min	2400
1hour	3600
2hour	7200
6hour	36000
12hour	72000
VeryFast	60
DefaultRate	300

18.7.1 Creating New Poll Rates

▼ To Create a New Poll Rate

1. Select the **Edit → Poll Rates** menu option to invoke the **Poll Rates** window.
2. Enter a name for your new poll rate and the polling interval (in seconds).
In the following example, a poll rate of 90 seconds is created and given the name “90secs.”
3. Click **Add** to load the new poll rate into the MIS.

Name	Value(seconds)
5min	300
5secs	5
6hour	36000
90secs	90
DefaultRate	300
Fast	60
Medium	900
Moderate	300
Poll	20

Name:

Rate:

FIGURE 18-9 Creating a New Poll Rate

18.7.2 Modifying a Poll Rate

Note – Poll rates cannot be modified if they are in use in a template stored in the MIS.

▼ To Modify a Poll Rate

1. Select the Edit → Poll Rates menu option to invoke the Poll Rates window.
2. Select the poll rate you wish to modify from the tabular display of poll rates.
3. Enter the new rate (in seconds) in the Rate field.
4. Click Modify.

18.8 Modifying the Mapping of Colors to Severities

A *severity* describes the degree of importance you attach to a network resource entering a state. 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 (as shown in the following figure) controls the use of color in the Network Views and Alarms to represent the severity of alarms logged against managed resources.

The screenshot shows a dialog box titled "Design Advanced Requests – Sever". Inside, there is a table with three columns: Name, Value, and Color. The table contains the following data:

Name	Value	Color
Critical	1	red
Major	2	orange
Minor	3	cyan
Warning	4	yellow
Normal	5	lightgrey
Indeterminate	0	blue

Below the table, there is a form to edit a severity. It includes labels for "Name: Indeterminate", "Value: 0", and "Color: blue". The "Color" field is a text input box containing the word "blue". At the bottom of the dialog, there are three buttons: "Modify", "Close", and "Help".

FIGURE 18-10 Nerve Centre’s Mapping of Colours to Severities

For information on changing the color associated with a severity, see Section 3.3.1, “Changing the Color Associated with a Severity.”

18.9 Graphical State Diagram Display

In the Request Designer main window, select View → Graphical (or View → Both) to receive the State Diagram window shown in the following figure. In this figure, you start with the single state, “Ground,” which is the required starting point for all request templates.

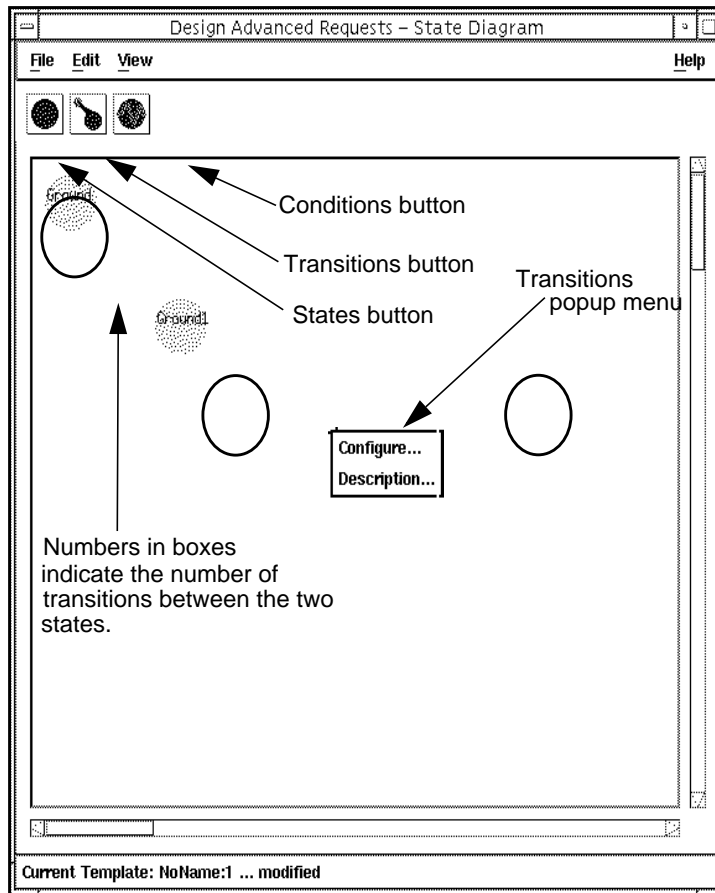


FIGURE 18-11 Graphical State Diagram Display

The menus at the top of this window are identical to those in the text-based display.

18.9.1 Creating a Template Through the State Diagram Display

The graphical display icons are pointed out in the above figure. 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

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. Refer to Section 18.4, “States.”

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 explanation of how to use that window in Section 18.5, “Transitions.” 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 graphical display.

The graphical display has a message area at the bottom of the window that is analogous to the text-display message area.

The numbers in boxes indicate the number of transitions that have been created between a pair of states.

18.9.2 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.

Nerve Center Utilities

Nerve Center 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 chapter describes the following topic:

- Section 19.1 “`em_ncimport` and `em_ncexport`” on page 19-1

19.1 `em_ncimport` and `em_ncexport`

The `em_ncimport` and `em_ncexport` facilities are 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]
[-template <template_name>] [-condition <condition_name>]
[-pollrate <poll_rate_name>] [-severity <severity_name>]
[-minimize] [-help]
```

```
em_ncimport [-host <hostname>] [-file <filename>] [-template]
[-help]
[-pollrate] [-severity] [-condition] [-v]
```

19.1.1 Options

- `-help`—Displays help text.

- `-host <hostname>`—`<hostname>` 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.
- `-template [<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 `-template` option imports all templates in the selected file.

- `-condition [<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 `-condition` option imports all conditions in the selected file.

- `-pollrate [<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 `-pollrate` option imports all poll rates from the selected file.

- `-severity [<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 `-severity` option imports all severities from the specified file.

- `-verbose[-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 supported only for `em_ncexport`.

Note – For `em_ncexport`, at least one of the following options must be specified: `-template`, `-condition`, `-pollrate`, `-severity`. `em_ncimport` can be used without any options being specified. If no options are specified, `em_ncimport` imports all components in the specified file.

19.1.2 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
```

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

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 Design Advanced Requests.

This chapter describes the following topics:

- Section 20.1 “Conditions” on page 20-1
- Section 20.2 “Types of Operands” on page 20-2
- Section 20.3 “Constants” on page 20-3
- Section 20.4 “Variables in a Condition” on page 20-3
- Section 20.5 “Data Types” on page 20-5
- Section 20.6 “System Variables” on page 20-5
- Section 20.7 “Attributes” on page 20-6
- Section 20.8 “Operators” on page 20-8
- Section 20.9 “Control Structures” on page 20-11
- Section 20.10 “Timestamp Arithmetic” on page 20-16
- Section 20.11 “Error Checking” on page 20-16

20.1 Conditions

A *condition* is a sequence of one or more statements written in the *Solstice Enterprise Manager* (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.

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 22.

20.2 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;
```

20.3 Constants

The following table shows the list of constants represented in the RCL syntax.

TABLE 20-1 RCL Syntax Restraints

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)

20.4 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 20-2. For examples illustrating the use of RCL system variables, see Chapter 21.
- **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 22.2.9, “Defined”) to test whether a variable has been defined.

20.4.1 Variable Names

The name of a variable begins with \$ (dollar sign) followed by one or more alphanumeric characters or _ (underscore).

Note – Case is significant in variable names.

20.4.2 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.

20.5 Data Types

Because the Design Advanced Requests 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 20-1 with which most operators deal. An example of such an operator is `TrapSpecificType`, which takes as input an operand of type `InternetActionInfo`.

20.6 System Variables

The names and types of the available system variables are shown in the following table.

TABLE 20-2 System Variables Available to a Condition

Name	Type	Description
<code>\$eventOC</code>	<code>OID</code>	Object class of last/current event
<code>\$eventOI</code>	<code>ObjectInstance</code>	Object instance of event
<code>\$eventInfo</code>	<code>eventInfo</code>	<code>eventInfo</code> of <code>eventType</code>
<code>\$eventTime</code>	<code>GeneralizedTime</code>	Actual time the event was generated

TABLE 20-2 System Variables Available to a Condition *(Continued)*

Name	Type	Description
\$eventType	OID	OID of the event type of the last event
\$messType	INTEGER	Type of current message (refer to TABLE 21-3 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 ObjectInstance	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 Network Views.
\$pollTime	GeneralizedTime	Delay until the first poll is sent and the time between successive polls, in seconds
\$severity	INTEGER	Severity level of current state

Chapter 21 contains examples illustrating the use of system variables.

20.7 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.

20.7.1 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.

Note – It is recommended that the second format be used for attribute names to avoid any potential name collisions that occur when the same label name is defined in multiple documents.

For example, the attribute `sysContact` specified in the document `IIMCRFC1213-MIB`, is written:

```
"IIMCRFC1213-MIB":sysContact
```

20.8 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. See the following table.

TABLE 20-3 RCL Operator Symbols

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
Equality	>=	greater than or equal
	==	equal
Logical	!=	not equal
	AND	and
	OR	or
Address	NOT	not
	&	“Address of” is used in the argument of <code>defined</code> or <code>undefine</code> 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.

20.8.1 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;
```

20.8.2 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
```

20.8.3 Precedence and Associativity

The precedence and associativity of operators are summarized in the following table.

TABLE 20-4 Precedence of Operators

Operator	Name	Associativity
High		
=	Assign	right
OR	Or	left
AND	And	left

TABLE 20-4 Precedence of Operators *(Continued)*

Operator	Name	Associativity
	Bitwise numeric OR	left
^	Bitwise numeric XOR	
&	Bitwise numeric AND	
==	Equal	left
!=	Not equal	
<	Less than	left
<=	Less than or equal	
>	Greater than	
>=	Greater than or equal	
+	Plus	left
-	Minus	
*	Multiply	left
/	Divide	
%	Modulus	
&	Address	right
~	Bitwise NOT (numeric)	right
-	Negative (numeric)	
NOT	Negation (logical)	
Low		

Parentheses force precedence in the usual way.

20.9 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.

20.9.1 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;}
```

20.9.2 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;
}
```

20.9.3 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\"{ }");
```

20.9.4 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.

20.9.5 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);
            }
        }
    }
}
```

20.10 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.

20.11 Error Checking

The Design Advanced Requests Tool checks and catches lexical and syntactic errors at the time you try to save the condition's definition. If it finds an error, the Design Advanced Requests displays an error dialog, and the offending condition is not saved. If there are any references to attributes in conditions, the Design Advanced Requests 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, Design Advanced Requests 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 Design Advanced Requests 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.

Note – 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, see Chapter 16.

Using RCL System Variables

This chapter describes the Request Condition Language (RCL) system variables that you can use in building Nerve Center request templates. The individual system variables along with examples illustrating their use are discussed in alphabetical order.

This chapter describes the following topics:

- Section 21.1.1 “\$eventInfo” on page 21-2
- Section 21.1.2 “\$eventOI” on page 21-4
- Section 21.1.3 “\$eventTime” on page 21-4
- Section 21.1.4 “\$eventType” on page 21-4
- Section 21.1.5 “\$messType” on page 21-6
- Section 21.1.6 “\$pollfdn” on page 21-7
- Section 21.1.7 “\$pollFdnSet” on page 21-8

21.1 System Variables

The names and types of the available system variables are summarized in the following table.

TABLE 21-1 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.

TABLE 21-1 System Variables Available to a Condition *(Continued)*

Name	Type	Description
\$messType	INTEGER	Type of current message (refer to TABLE 21-3 for \$messType values).
\$pollOC	ObjectClass	Object class last polled or being currently polled.
\$pollfdn	ObjectInstance	Object instance being polled.
\$pollFdnSet	SET OF ObjectInstance	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 Network Views.
\$pollTime	GeneralizedTime	Delay until the first poll is sent and the time between successive polls, in seconds.
\$severity	INTEGER	Severity level of current state.

21.1.1 \$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 8 of *Management Information Server Guide*.

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 `strToAsn()` is text that refers to the event type—`NerveCenterAlarmInfo`. This type is defined in the ASN.1 document `/opt/SUNWconn/em/etc/asn1/nc.asn1`. 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 fifth attribute, `additionalText`. The permissible values and standard interpretation of `perceivedSeverity` values is provided in the following table.

TABLE 21-2 `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.

In the request template, `mosiSeverity` represents the severity of the state (this could be the state in which the event originated). The number of the state is `mosiStateID`, the same as the above referenced state (as ordered in the request designer).

21.1.2 \$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);
```

21.1.3 \$eventTime

\$eventTime provides the time when the event notification was generated. In the EventSample request template, a request is listening for startup of *Solstice Enterprise Manager* (Solstice EM) tools. After subscribing for `objectCreation` events, the time of connection of the application to the MIS is calculated from the \$eventTime value of the `objectCreation` event:

```
$connect_time = $eventTime;
```

When the application instance is terminated, the EventSample request receives an `objectDelection` event. The timestamp saved in `$connect_time` is then used to calculate the length of time the application was connected to the MIS:

```
$delete_time = getTimeStamp();  
$time_connected = $delete_time - $connect_time;  
$time_connected = $time_connected / 1000.00;
```

21.1.4 \$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, if it is, calls `sendEvent()` to post the event to the alarm log.

```
$itType=NameToOid("internetAlarm");  
IF ($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 Chapter 8 in *Management Information Server (MIS) Guide*.

21.1.5 \$messType

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 the following table.

TABLE 21-3 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

TABLE 21-3 Values of \$messType (Continued)

Message	No.	Description
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

21.1.6 \$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 Network Views, 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 = "/InternetSystemId=NULL";  
$pollfdn = appendRdn($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"/InternetSystemId={1 3 6 1 4 1 42 2 2 2 9 1 1 3 6  
1 2 1 1 0}
```

Chapter 15 describes sample templates that change the target of polling during execution.

21.1.7 \$pollFdnSet

When a request is launched against a device selected in the Network Views, \$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 Network Discovery 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\"{ }");
```

RCL Functions

The RCL Built-in functions are described in this chapter.

This chapter describes the following topics:

- Section 22.1 “Summary of RCL Built-in Functions” on page 22-1
- Section 22.2 “The RCL Functions” on page 22-4

Built-in functions take the form of a function call with a set of arguments. The `RCL Functions` list denotes an `Asn1Value` whose type is SET, SEQUENCE, SET OF, or SEQUENCE OF. `<Var>` indicates a variable name. `<Attr>` indicates an attribute name.

Note – The names of RCL functions are not case-sensitive.

22.1 Summary of RCL Built-in Functions

22.1.1 AlarmLog 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`.

22.1.2 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.

22.1.3 Value Check Functions

- `Defined`—True if variable or attribute has a value.
- `Undefine`—Sets variable or attribute to have no value.

22.1.4 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.

22.1.5 Action Functions

- `Unixcmd`—Executes a specified UNIX command.
- `Mail`—Sends an e-mail message.

22.1.6 ASN.1 Conversion Functions

- `AsnToStr`—Builds string representation of an ASN.1 value.
- `StrToAsn`—Builds ASN.1 values from strings.

22.1.7 SunNet Manager RPC Request Functions

- `SnmEventRequest`—Issues request to an SNM RPC agent.
- `SnmKillRequest`—Kills a previously issued SNM request.

22.1.8 Debugging Function

Print—Prints values if `em_debug misc_stdout` option is turned on.

22.1.9 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.

- **CompareLists**—True if the two ASN.1 lists being compared match.
- **Exclude**—Deletes a component from an ASN.1 list.
- **Extract**—Returns value of a component in a list.
- **Include**—Used to construct an ASN.1 value of types SET or SEQUENCE.
- **IsChoice**—True if a variable or attribute is a choice.
- **IsList**—True if variable or attribute is a list.
- **IsMember**—True if a given component is part of a specified ASN.1 list.
- **NumElements**—Returns the number of elements in a list.
- **ReplaceMember**—Replaces a component with another component in an ASN.1 list.

22.1.10 Time Functions

GetTimeStamp—Retrieves the current time of the MIS machine.

22.1.11 Event-Handling Functions

- **Set**—Performs an M-SET on a specified object.
- **Subscribe**—Subscribes to event type specified in string.
- **SubscribeOi**—Subscribes 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.

22.1.12 Request Control Functions

Exit—Causes a request to delete itself.

22.2 The RCL Functions

The Request Condition Language (RCL) built-in functions are listed here in alphabetical order.

22.2.1 AddressStrToAddress

Syntax:

```
AdressStrToAddress(<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");
```

22.2.2 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 Enterprise Manager* (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 the following table.

TABLE 22-1 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

22.2.2.1 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 `AlarmLog` can be viewed and cleared in the Alarms.

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 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.

For more information on the Alarm Service, see Chapter 4.

22.2.3 AlarmOi

Syntax:

```
AlarmOi( <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 22-1). 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 22.2.2.1, “Alarm Logging and Viewer Fault Status” or Chapter 4.

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.

The value of `probableCause` can also be assigned using the possible values defined in an ASN.1 document such as `dmi.asn1`. For example, the following statement clears a previous `nerveCenterAlarm` that had used `thresholdCrossed` as its `probableCause` value:

```
$info = strToAsn("EM-NC-ASN1.NerveCenterAlarmInfo", "{Attribute-
ASN1Module.thresholdCrossed,cleared,\"SNMP is not
responding\",3,1}");
alarmOi($save_pollfdn,$info);
```

22.2.4 AlarmStr

Syntax:

`AlarmStr(<perceivedSeverity>, <additionalText>);`

where `<perceivedSeverity>` is of type `INTEGER` (in the range 0-5) and `<additionalText>` is a text string, or 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 22-1).

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 Alarms tool. 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 22.2.2.1, “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.

When a text string is passed as `<additionalText>`, variables can be interspersed within the text string. For example:

```
alarmStr(3,"Machine $host went down at $timestamp");
```

The textual representation of the value of `$host` and `$timestamp` will be inserted into the text, which becomes the `additionalText` attribute in the `nerveCenterAlarm`.

22.2.5 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");
```

22.2.6 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 = "/internetSystemId=NULL;
            $pollfdn = AppendRdn($dn,$tmp);
            $count = $num+1;
        }
        else
        {
            $count = $count+1;
        }
    }
}
else
{
    $tmp = "/internetSystemId=NULL";
    $pollfdn = AppendRdn($dn,$tmp);
}
true;
```

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"/internetSystemId=NULL/cmipsnmpProxyAgentId="bigguy"
```

The `SetInternetSystem` sample condition then resets the value of `$pollfdn` to point to the RFC 1213 `internetSystem` group object via an `appendRdn` statement:

```
$tmp = "/internetSystemId=NULL";  
$pollfdn = AppendRdn($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");
```

22.2.7 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 rather than its equivalent string form. For example, the textual representation of the `systemId` attribute would be the following:

```
attributeId "Rec. X.721 | ISO/IEC 10165-2 : 1992":systemId
```

The curly brace representation of the `systemId` attribute would be as follows:

```
attributeId { 2 9 3 2 7 4 }
```

The textual option is available only if <*asn1_value*> has been defined in an ASN.1 or GDMO document that the MIS knows about. If <*asn1_value*> is not so defined, only the curly brace representation is available.

The `SetInternetSystem` sample condition uses the following statement to convert an FDN to its textual representation in order to do a string compare on its contents:

```
$dnstr=asnToStr($dn1,TRUE);
```

22.2.8 CompareLists

Syntax:

```
CompareLists(&<list1>, <list2>);
```

Return Value:

BOOLEAN.

This function compares two ASN.1 lists and returns TRUE if they match; the function returns FALSE otherwise. For example:

```
$match = CompareLists(&$eventInfo,$myevent);
```

22.2.9 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 21-1), 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);
```

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 `defined` returned successful.

22.2.10 Exit

Syntax:

```
Exit();
```

Return Value:

None.

When this function is called, this causes the deletion of the request in which it is called. This is useful for deleting a request if it has reached a dead or error state. Doing so reduces the system overhead caused by requests that have been launched against devices that are not configured to support that request. For example, the following condition might be executed as an action after transition to an Error state in a template that polls for SNMP attributes:

```
alarmStr(3,"Template was launched at a non-SNMP device.");  
exit();
```


22.2.11 Exclude

Syntax:

```
Exclude(&<list_variable>, <component>);
```

Return Value:

None.

This function deletes the component specified by *<component>* from the ASN.1 list specified by *<list_variable>*. For example:

```
Exclude(&$list, $pollfdn);
```

22.2.12 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,

```
$sele1=Extract(&collectionInfoList,"1");
```

\$sele1 will assign the first set from *collectionInfoList*, which is a SET OF SEQUENCE.

In the following example, the ASN.1 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 6 of *Management Information Server Guide*.) 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");
```

22.2.13 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");
```

22.2.14 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.

22.2.15 GetTimeStamp

Syntax:

GetTimeStamp;

Return Value:

GeneralizedTime. Current time 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);}
```

22.2.16 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);
```

The second parameter *<value>* must be defined for `include()` to succeed. This is particularly important if the *<value>* is a variable that has been assigned an attribute's value. For example, consider:

```
defined("&"IIMCRFC1213-MIB":sysDescr);  
$element = "&"IIMCRFC1213-MIB":sysDescr;  
$list = strToAsn("SET","{");  
include(&$list,$element);
```

If the `defined()` function is not called, then `include()` will fail.

22.2.17 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");
```

22.2.18 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);
```

22.2.19 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);
```

22.2.20 IsMember

Syntax:

`IsMember(&<list_variable>, <component_variable>);`

Return Value:

BOOLEAN.

Returns TRUE if the component specified by <component_variable> is included in the ASN.1 list specified by <list_variable>. Returns FALSE otherwise. For example:

```
$match = IsMember(&$eventInfo,$myattribute);
```

22.2.21 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");
```

22.2.22 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.

```
$destIpAddress = NameToAddress($host);  
SendTrap($destIpAddress, $eventType, $eventInfo);
```

22.2.23 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");
```

22.2.24 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);
```

22.2.25 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");
```

22.2.26 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);
```

22.2.27 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, see Chapter 16."

22.2.28 ReplaceMember

Syntax:

```
ReplaceMember(&<list_variable>, <old_component>, <new_component>);
```

Return Value: None.

This function replaces the a component in the ASN.1 list specified by *<list_variable>* with a new component. The component being replaced is specified by *<old_component>* and the component that is to replace it is specified by *<new_component>*. For example:

```
ReplaceMember(&$mylist,$oldval,$newval);
```

22.2.29 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>*.

22.2.30 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 Solstice EM event types is described in Chapter 8 of *Management Information Server Guide*.)

<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 `AlarmLog` 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.

22.2.31 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);
```

22.2.32 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.

Because RCL doesn't support callbacks, it's not possible for RCL templates to always check for the arrival of responses. If the response comes back almost instantaneously, the NC template can check for `messType` value (`==9`) within the same state as the `Set()` was invoked. A value of 9 means that the `Set()` response has arrived. If the `messType` is not 9, it doesn't imply anything.

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\"}}},{topoNodeDBI  
d,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`.

Note – Poll attributes after setting them in order to check if SET succeeded.

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);}
```

22.2.33 SnmEventRequest

Syntax:

`SnmEventRequest(<oi>, <EventRequest>, &<snmRequestHandle>);`

where *<oi>* is of type `ObjectInstance`, *<EventRequest>* is of either of type `OCTET STRING` or of type `Asn1Value`. *<snmRequestHandle>* 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.

To identify this SNM event request, the function returns a unique handle in *<snmRequestHandle>*. This handle can be used to kill or stop the SNM event request that was started via the function `snmKillRequest()`. 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 the following table. The order of occurrence in the table is the order within *<EventRequest>*.

TABLE 22-2 Arguments in *<EventRequest>*

Argument	Data Type	Description	Required/ Optional
agentHost	OCTET STRING	Name of target agent system	Required
agentProgram	INTEGER	RPC 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

TABLE 22-2 Arguments in <EventRequest> (Continued)

Argument	Data Type	Description	Required/ Optional
threshold	SEQUENCE of <ul style="list-style-type: none"> • attrName — OCTET STRING • attrType — INTEGER • relop — INTEGER • threshValue — OCTET STRING • priority — ENUMERATED <ul style="list-style-type: none"> — low (1) — medium (2) — high (3) 	<ul style="list-style-type: none"> •Name of the attribute used to define the threshold •Data type of the operands for relop. See TABLE 22-4. •Relational operator used in defining the threshold. See TABLE 22-3. •Threshold value to check for •Priority assigned to an SNMP 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 <ul style="list-style-type: none"> • name — OCTET STRING • value — OCTET STRING 	Optional arguments are agent-specific. For example, requests to na.ping use these to set packet size, time to wait for echo replies, etc.	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

TABLE 22-2 Arguments in *<EventRequest>* (Continued)

Argument	Data Type	Description	Required/ 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 the following table.

TABLE 22-3 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
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

TABLE 22-3 Relational Operators in SNMP Request Thresholds *(Continued)*

Integer Value	Relational Operator
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 the following table.

TABLE 22-4 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
15	RFC 1065 integer
16	RFC 1065 octet string
17	RFC 1065 object identifier
18	RFC 1065 network address
19	RFC 1065 IP address
20	RFC 1065 counter

TABLE 22-4 Data Types for Threshold Operands (Continued)

Integer Value	Data Type
21	RFC 1065 gauge
22	RFC 1065 timeticks
23	RFC 1065 opaque

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 the following table. The SNM Event Forwarder sends `snmAlarmEvents` to the MIS.

TABLE 22-5 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 `$pollFdnSet` can be found in Section 20.9.3, “WHILE Constructs.”) `$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, see Chapter 17.

22.2.34 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.

22.2.35 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:

```
$salarminfo = "{ ";
$salarminfo = strcat($salarminfo,$cause);
$salarminfo = strcat($salarminfo,$trans_domain);
$salarminfo = strcat($salarminfo,$traddr);
$salarminfo = strcat($salarminfo,$access);
$salarminfo = strcat($salarminfo,$ainfo);
$salarminfo = strcat($salarminfo," }");
$internetAlarmInfo = strToAsn("IimcCommonDef.InternetAlarmInfo",$salarminfo);
$eventInfo = $internetAlarmInfo;
SendTrap($destIp,$itType,$eventInfo);
```

22.2.36 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);
```

22.2.37 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."

22.2.38 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."

22.2.39 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.

22.2.40 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 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 Chapter 6 in *Developing C++ Applications*.

22.2.40.1 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:

```
$myFilter = strToAsn("CMIP-1.CMISFilter","or: {item: equality:
{objectClass,mosi}, item: equality: {eventType,
internetAlarm}}");
$index = subscribeFilter($myFilter);

$myFilter = strToAsn("CMIP-1.CMISFilter", "item: equality:
{eventType, internetAlarm}");
$index = subscribeFilter($myFilter);
```

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.

22.2.40.2 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);
```

22.2.41 SubscribeOi

Syntax:

`SubscribeOi(<event_name_str>, <oc_str>, <event_oi>);`

where `<event_name_str>` and `<oc_str>` are of type OCTET STRING and `<event_oi>` is of type ObjectInstance.

Return Value:

INTEGER

For example:

```
$lkidx = subscribeOi("linkDownTrap", "cmipsnmpProxyAgent", $pollfdn);  
$lkType=NameToOid("linkDownTrap");  
true;
```

`subscribeOi()` is 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 `<oc_str>` argument specifies the object class of interest. The function will accept an empty string `"{}"` as the value of `<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.

22.2.42 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);
```

Note – `$eventInfo` in this example must be an `internetAlarm`.

The possible return values are described in the following table.

TABLE 22-6 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.
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.

22.2.43 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);
```

Note – `$eventInfo` in this example must be an `internetAlarm`. This function does *not* apply to the `enterpriseSpecificTrap` event notifications to which Solstice EM maps SNMP enterprise-specific traps by default. To obtain the specific trap number from `enterpriseSpecificTrap` notifications, use the `Extract()` function to obtain the value of the `probableCause` attribute. Refer to Section 22.2.12, “Extract.”

22.2.44 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);
```

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);
```

Once the value of `sysDescr` has been undefined, the `IsSystemDescr` sample condition can once again invoke the `define` function to test for system availability.

22.2.45 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");
```

22.2.46 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.

Index

SYMBOLS

- \$eventInfo, 21-2
- \$eventOI
 - definition of, 21-4
- \$eventType variable, 21-4
- \$eventTypes
 - that MIS knows by default, 21-5
- \$messType
 - checking after MSet, 22-26
 - checking after SendAction, 22-24
 - possible values of, 21-6
- \$pollfdn
 - use of Append_rdn to set, 22-10
- \$pollfdn system variable in RCL, 14-10
- \$pollfdn variable, 21-7
- \$pollFdnSet, 21-8

A

- access
 - application, 3-5
 - application feature, 3-5
 - database, 3-5
 - object, 3-5
- access to
 - managed resources via CMIP, 12-10
 - SNM agents, 8-29
- access, data, 1-15
- action
 - at a transition, 18-16
 - three types in request templates, 18-2
 - types of, 18-16
- Action Request System
 - using with EM, 8-13
- additionalText*
 - specifying in alarmStr function, 22-8
- additionalText
 - Information, 11-12
 - additionalText information, 11-12
- AdminOperStatusUp, 6-8
- advanced requests
 - See also Nerve Center Requests, 3-7
- agent
 - agentVersion number
 - how to find, 22-29
 - definition, 2-1
- agent/station model, 1-9
- agentOverloadAlarm, 13-7
- agents
 - communications, 1-10
 - description, 1-5, 1-9
 - legacy support, 1-17
 - MPAs, 1-10, 1-13
 - protocol descriptions, 1-16
- alarm colors, 3-3
 - assigning to icons, 3-5
 - changing, 3-4
 - process for changing, 3-6
 - updated in Network Views, 3-6
- Alarm Manager
 - how it displays alarm notifications, 4-8
- Alarm Service
 - in action, a scenario, 4-4
 - logging non-default alarms to the AlarmLog, 4-6
 - Overview, 4-1
 - perceived severity
 - SNMP traps, 11-10

- perceived severity, 7-8
- turning off, 4-8
- what it does, 3-5
- what it is, 4-1
- which alarm logs to monitor, 4-6
- AlarmLog, 3-7, 4-3
 - discriminator construct, 4-3
 - non-default types, 4-3
- alarms
 - counters, 4-2
 - function, 22-4
 - logging via RCL SendEvent function, 22-24
 - SNM, 7-7
- ampersand
 - use of in RCL, 20-7
- ancillary services, MIS, 1-14
- APIs, 1-5
- appendRdn
 - syntax of function, 22-10
 - use of, 21-8
- architecture, EM, 1-6
- ASN.1, 9-3
 - values, converting to strings in RCL, 22-12
- Asn1Type
 - use in RCL, 20-5
- associativity
 - of operators in RCL, 20-10
- asynchronous event reports (traps), 10-7
- attribute
 - referenced in RCL conditions, 20-6

B

- basic concepts, EM, 1-9
- bc_map file
 - updated by em_snm_type_import, 8-7
- bitwise operators
 - use of in RCL, 20-10
- browse log records
 - how to, 5-1

C

- CheckCPU, 6-8
- clear alarms
 - generated by glyph reset on SNM Console, 7-3
- cleared CMIP MPA threshold, 13-2

CMIP

- agents
 - use in device management, 3-11
- changing timeout values, 12-9
- configuring, 12-8
- messages, 12-10
- MPA, 12-11
- Notifications, Logging to AlarmLog, 4-3
- Proxy Agent, 12-10
- support, in Solstice EM, 2-15
- CMIP MPA
 - configurations, 2-16
 - starting, 12-11
 - versions of, 12-10
- CMIP MPA Overload, 13-1
- CMIP MPA, configuration parameters, 13-2
- CMIP protocol
 - supported, 1-16
- CMIS Filter, 3-6
 - accepting all notifications, 5-18
 - accepting notifications, 5-15
 - defining, 5-13
 - not accepting notification, 5-18
 - sample filters, 5-18
 - with multiple ANDs, 5-17
- CMIS filters
 - use in RCL event subscriptions, 22-35
- CMIS-like, 6-12
- colors
 - mapping
 - to severity, how to change, 18-23
 - mapping, changing the default, 4-9
 - mapping, default, 4-9
- Common Management Information Service (CMIS)
 - requests, 9-3
- compilers, 1-19
- components, EM, 1-4
- Concise MIB compiler (em_cmibp), 9-3
- condition
 - definition of, 18-1, 20-1
 - role as an action, 18-2
 - role in defining transition, 18-1
- Condition Language, 20-1
 - assignment operators, 20-8
 - Constants, 20-3
 - operator
 - assignment, 20-8
 - operator symbols, 20-8
 - Operators, 20-8

- System variables, 20-5
- value check functions
 - defined, 22-13
- Variables, 20-3
- Condition language
 - See Request Condition Language (RCL), 20-1
- conditions
 - evaluating in request, 14-7
 - two roles of, 20-1
- configuration parameters, CMIP MPA, 13-2
- Configuration Tool, 7-4
- configuring
 - SunLink CMIP, 12-8
- containment relationship, 22-10
- converting SNM schema files, 8-29
- Cooperative Consoles
 - event types, 7-3
 - filtering capabilities, 7-3
 - glyph traps, 7-3
 - installing packages, 7-4
 - operation, 7-7
 - periphery-to-center configuration, 7-1
 - Receiver application, 7-4
 - receiving station, definition, 7-4
 - Sender daemon, 7-4
 - remote, 7-7
 - sending station, 7-4
 - setting up for EM, 7-4
 - SNM database traps, 7-3
 - SNM events, 7-2
 - SNM glyph traps, 7-3
 - SNMP Traps, 7-3
 - using with EM, 7-1
- Cooperative Consoles and SNM, 7-1
- critical, CMIP MPA threshold, 13-2
- custom mangement applications, 1-18
- custom mapping of SNMP traps, 11-25

D

- data access, in MIS, 1-15
- data types
 - in SNM event requests, 22-30
- database, MIS, 1-12
- database, SNM
 - loading into EM, 8-25
- debugging
 - request templates

- RCL Print function, 22-22
 - via RCL Print() function, 22-22
- device
 - availability, 3-8
 - status, 3-8
- DeviceReachablePing, 6-8
- Discriminator Construct
 - defining, 5-13
- discriminator construct
 - define event logging, 3-6
 - See also Log Filters, 3-6
- discriminator construct, 5-1
- DiskPartitionsFull, 6-9
- Domain Manager
 - See SunNet Manager, 7-1

E

- element.schemas, 8-6
- EM architecture
 - client/server, 2-2
- em_cmibp compiler, 9-3
- em_cmipconfig
 - replaced by -cmip option of em_oct, 12-17
- em_debug
 - utility, 22-22
- em_debug
 - use with RCL print() function, 22-22
- EM_HOME environment variable, 5-21
- em_ncexport utility
 - options, 19-1
- em_ncimport utility
 - options, 19-1
- em_schema2gdmo compiler, 6-12
- em_snm_type_import utility
 - syntax of, 8-7
- em_snmdb_import utility, 8-25
- em_snmp-trap, 11-9
 - See also trap daemon, 11-4
- em_trapd, 11-4
- emAlarmLogList, 4-6
- enterprise field, SNMP trap, 11-6
- enterprise mapping blocks, 11-14
- enterpriseSpecific trap, 11-12
- environment variable
 - EM_SERVER, 8-8, 8-9, 8-27
 - LD_LIBRARY_PATH, 8-2, 8-8, 8-27
- Event Dispatcher (na.event)

- role in Cooperative Consoles, 7-6
- event information
 - two types, 3-7
- event logging, 14-8
- Event Logs, 3-6
 - configuration files, 5-21
 - sample filters, 5-18
- Event Notification
 - definition, 2-2
 - SNMP Traps, 3-12
- event notification, 3-7
- event notifications
 - as handled by requests, 14-3
 - default type, 11-12
 - use in fault management, 3-11
- event subscription
 - RCL statements for, 14-5
 - via RCL functions, 22-34
 - what it is, 3-14
- event types
 - Cooperative Consoles, 7-3
 - for trap mapping, 11-17
 - known to MIS by default, 21-5
- events
 - extracting attributes of in RCL, 22-15
 - logging via RCL `SendEvent` function, 22-24

F

- fault management
 - definition of, 3-1
- fault status
 - indication, in SunNet Manager, 7-7
 - monitoring, 3-3
 - procedure for monitoring, 3-1
 - viewing, 3-3
- features, EM, 1-3
- filtering, Cooperative Consoles, 7-3
- forwarding
 - SNMP traps
 - in a request template, 14-8
- forwarding SNMP traps, 11-2
- fully distinguished name (FDN), 22-10

G

- GDMO, 6-12, 9-3

- description, determines names of attributes, 20-6
 - document, 22-38
- generic trap
 - values, 11-10
- graphical display
 - for Request Designer, 18-25
 - maximum number of states displayed for Request Designer, 18-26
- Guidelines for the Definition of Managed Objects (GDMO), 2-18

H

Hosts

- set up CiscoWorks on host with new name or IP address, 8-12
- set up EM on host with new name or IP address, 8-11
- set up Landmark on host with new name or IP address, 8-12
- set up Remedy on host with new name or IP address, 8-12

I

icons

- converting SNM glyphs to EM format, 8-5
- identifier field, traps, 11-13
- IF construct
 - example of in condition, 22-26
- IF constructs
 - use in RCL conditions, 22-24
 - use of in RCL, 20-11
- IF ELSE constructs
 - nesting of in RCL, 20-14
 - use of in RCL, 20-12
- IIMC standard for event notification, 11-14
- incoming SNMP traps, 11-13
- installing Cooperative Consoles, 7-4
- `InternetActionInfo`, 20-5
- `internetAlarms`
 - logging to `AlarmLog`, 4-3
- introduction, EM, 1-1 to ??

K

- Konfig

- working with Em, 8-14

Konfig 2.4

- using with EM, 8-14

L

Landmark

- set up on host with new name or IP address, 8-12
- working with EM, 8-22

Landmark's Performance Works

- using with EM, 8-22

LD_LIBRARY_PATH

- setting for SNM API access, 8-5

legacy support, 1-17

libnetmgmt library, 8-26

libnetmgt.so library, 8-2

libnetmgt_db.so compatibility library, 8-2

libnetmgt_db.so library, 8-2

log

- notification type, 5-2

log discriminator, 5-3

Log Manager

- Deleting selected log objects, 5-9

- discriminator construct, 5-5

- optional command line parameters, 5-4

- starting, 5-4

- verify event types, 11-26

- what it is, 5-1

log objects

- adding to the AlarmLog, 4-7

- attributes, 5-2

- name, 5-2

- creation of, 4-3

- deleting, 4-7

- number in MIS, 5-1

- removing from emAlarmLogList using Log Manager, 4-7

- removing from emAlarmLogList using OBED, 4-8

- what is it, 5-1

log records

- creation, notifying the Alarm Service, 4-2

- Event Logs tool, 5-1

- from event mapping

- location of, 5-2

Log Viewer

- configuration files, 5-5

logging

- discriminator construct

- object identifiers, 5-16

- notification records, 5-3

- notification types, 5-15

Logs

- accessing on a remote MIS, 5-5

- creating, 5-7

- creating for enterprise-specific traps, 3-16

logs, creating

- example of, 3-15

M

M-ACTION

- sent by RCL SendAction function, 22-23

major, CMIP MPA threshold, 13-2

managed objects

- description, 1-5

Management Information Base, 1-17

Management Information Bases (MIBs), 9-3

management overview, 1-2

Management Protocol Adapter (MPAs), 6-11, 9-3

Management Protocol Adaptors (MPAs), 2-17

Management Tree of Overload Control Objects, 13-5

manager-to-manager capability, 10-10

managing

- devices, using RPC agents, 6-5

- objects, attributes of, accessing in RCL, 22-15

mapping

- blocks, 11-18

- records, 11-18

- SNM for perceived severity, 7-7

MetaData Repository (MDR), 20-6

MIBs, 1-17, 9-3

- adding to MIS, 2-19

Minimum Threshold Parameter, 13-4

minor, CMIP MPA threshold, 13-2

MIS

- ancillary services, 1-14

- data access, 1-15

- description, 1-5

- Nerve Center, 1-13

- object orientation, 1-15

- overview, 1-11

MIS requests

- advanced, 15-13

MIT, 22-21

modules, EM API, 1-18

Monitor
 ability to generate alarms, 3-8
 use in tracking device availability, 3-8
MPA, 1-10, 1-13, 6-11, 9-3
mpaOverloadController, 13-7

N

na.event
 role in SNM event requests, 6-3
na.snmp.schemas, 8-27
na.snmp-trap, 11-2
Nerve Center, 1-13
 definition of, 14-1
 Message types, 20-6
 Request Condition Language, 20-1
Nerve Center Requests, 3-7
nerveCenterAlarms
 logged by RCL functions, 22-7
 Logging to AlarmLog, 4-3
 structure of, 21-3
network management
 role of requests in, 3-2
 steps in performing, 3-1
network management applications, custom, 1-18
network management software, description, 1-9
network management, protocols, 1-16

O

object creation events
 subscribing for in RCL, 22-38
object instance
 changing attribute values, 5-20
 creating, 5-19
 deleting, 5-19
object orientation, 1-15
Optivity
 working with EM, 8-16
Optivity 7.0
 using with EM, 8-16
OR
 use of in RCL conditions, 20-10
OSI stack
 use with CMIP MPA, 2-16
overload control parameter, 13-3
Overload Notification Parameter, 13-3

Overload Threshold Parameter, 13-3
overloadControlContainerName, 13-6
Overview, EM, 1-1
overview, EM
 application development support tools, 1-19
 components, 1-4
 concepts, 1-9
 data access, 1-15
 description, 1-1
 features, 1-3
 MIS, 1-11
 MIS services, 1-14
 MPAs, 1-13
 Nerve Center, 1-13
 object orientation, 1-15
 PMI, 1-13
 protocols, 1-16
 task overview, 1-2

P

perceivedSeverity
 permissible values of, 21-3
 values, SNMP trap, 11-10
perceivedSeverity
 in snmAlarmTraps, 7-7
Performance Works
 using with EM, 8-22
 working with EM, 8-22
periphery-to-center configuration
 using Cooperative Consoles, 7-1
PingUpOrDown, 6-9
PMI, 1-13
poll
 device, 3-8
poll interval parameter, 13-4
poll rate
 definition of, 18-20
polling, 3-8
 offloading to RPC agents, 6-3
 responses to a request, 14-4
polls, 3-7
Portable Management Interface, 1-13
precedence
 of operators in RCL, 20-11
print
 as RCL function, 22-22
priority

- in SNM, 7-7
- probable cause values, 11-11
- protocol
 - adapter (MPA)
 - for CMIP, 2-17
 - network management
 - support for, 2-7
 - operations, 10-10
- protocol adapter (MPA)
 - for CMIP, 2-15
 - for legacy or proprietary protocols, 2-17
 - for RPC, 2-8
 - for SNMP, 2-11
- protocols
 - network management, 1-16
- proxy agents, 6-11, 8-29

Q

- Q3 connection, 2-17

R

- RCL
 - Exit function, 22-14
- Receiver application, 7-4
- receiving SNM alarms, 7-7
- receiving station
 - definition, 7-4
- relational operators
 - in SNM event requests, 22-29
- Remedy
 - configuring to work with EM, 8-13
 - set up on host with new name or IP address, 8-12
- Remedy's Action Request System
 - using with EM, 8-13
- Remote Procedure Call (RPC), 6-11
 - support in Solstice EM, 2-8
- Remote Procedure Call (RPC) protocol
 - See RPC, 6-2
- request
 - definition of (for Nerve Center), 18-1
 - evaluating conditions in, 14-7
 - poll responses to, 14-4
 - polling and event subscription, 14-3
 - scope of variables in, 14-6
 - use of variables and attributes in, 14-5

Request Condition Language (RCL)

- AddressStrToAddress function, 22-4
- alarm function, 22-4
- alarm logging functions
 - Send_Event, 22-24
- alarmOi function, 22-6
- alarmStr function, 22-8
- anysr function, 22-9
- appendRdn function, 22-10
- arithmetic operators, 20-9
- ASN.1 conversion functions
 - sasnToStr, 22-12
 - StrToAsn, 22-32
- asnToStr function, 22-12
- attributes in, 20-9
- bitwise operators, 20-10
- built-in functions, 22-4
- components of, 20-1
- debugging function
 - print, 22-22
- defined function, 22-13
- event handling functions
 - SendAction, 22-23
 - SendTrap, 22-25
 - Subscribe, 22-34
 - SubscribeFilter, 22-35
 - SubscribeOi, 22-37
 - TrapGenericType, 22-38
 - TrapSpecificType, 22-39
 - Unsubscribe, 22-41
- Extract function, 22-15
- FinalStr function, 22-16
- FOREACH constructs, 20-13
- GetTimeStamp function, 22-17
- IF constructs, 20-11
- IF ELSE in, 20-12
- include function, 22-17
- InitialStr function, 22-18
- IsChoice function, 22-18
- IsList function, 22-19
- logical operators, 20-9
 - examples, 20-9
- Mail function, 22-20
- minus sign, use of, 20-9
- MSet function, 22-25
- name conversion functions
 - Append_rdn, 22-10
 - NameToAddress, 22-20
 - NameToOid, 22-20

- NameToAddress function, 22-20
- NameToOid function, 22-20
- nesting of constructs, 20-14
- NumElements function, 22-21
- OiNameToOi function, 22-21
- OiToOiName function, 22-22
- operators
 - and timestamp arithmetic, 20-16
 - arithmetic, 20-9
 - equality, 20-9
 - logical, 20-9
 - relational, 20-9
- operators in, 20-8
- precedence of operators, 20-10
- Print function, 22-22
- relational operators, 20-9
- SendAction function, 22-23
- SendEvent function, 22-24
- SendTrap function, 22-25
- SnmEventRequest function, 22-27
- SnmKillRequest function, 22-32
- statement blocks, 20-11, 20-12
- StrCat function, 22-33
- string handling functions
 - AnyStr, 22-9
 - FinalStr, 22-16
 - InitialStr, 22-18
 - StrCat, 22-33
- StrToAsn function, 22-32
- Subscribe function, 22-34
- SubscribeFilter function, 22-35
- SubscribeOi function, 22-37
- summary of functions, 22-1
- syntax checking, 20-16
- syntax of attribute names in, 20-7
- system variables, 20-5
- timestamp arithmetic operators, 20-16
- TrapGenericType function, 22-38
- TrapSpecificType function, 22-39
- type checking, 20-17
- types of operands in, 20-2
- Undefine function, 22-40
- Unixcmd function, 22-41
- Unsubscribe function, 22-41
- use of ampersand in, 20-8
- variables, dynamic typing of, 20-5
- WHILE constructs in, 20-13
- Request Condition Language(RCL)
 - attributes, 20-2
 - capabilities of, 14-1
 - constants, 20-2
 - operators
 - precedence of, 20-10
 - valid severities, 22-5
 - variables, 20-2
- Request Designer, 18-1
 - attribute names, 14-6
 - graphical display, 18-24
 - performing tasks using, 18-26
 - how to start, 18-5
 - log object, 14-8
 - logging an event, 14-8
 - notification, 14-4
 - notification arrival, 14-7
 - poll
 - conditions, 14-4
 - rate, 14-4
 - request
 - attribute name, 14-5
 - notification arrival, 14-6
 - poll response arrival, 14-6
- Request Condition Language, 20-1
- request notification
 - response to a poll, 14-3
- request template, 14-5
- starting, 18-5
- variable name, 14-6
- request template
 - forwarding a trap in, 14-8
 - high-level procedure for creation of, 15-27, 18-4
 - logging an event, 14-8
 - procedure for creating in graphical display, 18-25
 - sample explained, 15-19
 - system and user variables, 14-6
- request templates
 - exporting to ASCII file, 19-1
 - for RPC agents, 6-7
 - importing from ASCII file, 19-1
- request-related applications, 14-1
- Requests
 - use of Exit function to terminate, 22-14
- requests
 - advanced, 15-13
- retry-interval for SNMP proxy, 10-6
- RPC, 6-12
 - based agents, 6-11
 - calls, 6-12
 - legacy support, 1-17

- MPA, 6-11
- PDM, 6-12
- PDM and manageable SNM agent
 - communication, 6-12
- Protocol Driver Module, 8-2
- request, 6-12
- supported, 1-16
- RPC agents
 - building request templates for, 6-7
 - direct polling of, 6-2
 - type of machine supported on, 6-5
 - using, 6-1
 - using Discover to configure support for, 6-7
- RPC request, 6-12
- RPC_Diskinfo_ DiskPartitionsFull, 6-9
- RPC_Diskinfo_ WatchAllPartitions, 6-9
- RPC_Hostperf_VerifyProxyAgent, 6-9
- RPC_Hostperf_WatchCPU, 6-9
- RPC_MibII_Interface
 - CollisionDetection, 6-10
- RPC_MibII_InterfacePing Triptime, 6-10
- RPC_MibII_InterfaceStatus, 6-10
- RPC_MibII_sysUpTime_AgentUp, 6-10
- RPC_Ping_VerifyProxy Agent, 6-9
- RPC_SNMP_VerifyProxy Agent, 6-9

S

- schema compiler (em_schema2gdmo), 6-12
- schema files
 - relationship among .. shipped with SNM, 10-3
- schemas
 - See* SNM schemas, 6-5
 - SNMP, 10-3
- security, 3-5, 10-10
- Sender daemon, 7-4
 - filters for information forwarding, 7-3
- sending station
 - definition, 7-4
- severity
 - alarm, 3-3
 - assigning to icons, 3-5
 - assignments, changing, 11-11
 - changing colors, 3-4
 - color-coding of, 3-4, 11-10
 - defined, 3-3
 - in alarm logging functions, 22-5
 - in Request Designer, definition of, 18-23

- mapping, 4-6
- propagation, 4-2
 - topoNodeParents attribute, 4-2
 - topoNodePropagatePeers attribute, 4-2
- See* perceivedSeverity, 7-7
- short-circuiting
 - not implemented in RCL, 20-9
- Simple Network Management Protocol (SNMP), 9-1, 9-3
- SNM
 - agent, 6-12
 - API, use by Cooperative Consoles, 7-4
 - application, definition of, 8-1
 - applications, compatibility with EM, 8-3
 - configuration files, 8-26
 - default locations, 8-26
 - console fault indications, table, 7-7
 - database, importing into EM, 8-25
 - elements, converting schemas for third-party
 - elements, 8-6
 - event requests, 6-3
 - events, 7-2
 - fault status
 - mapping to perceivedSeverity, 3-22
 - fault status, mapping to perceivedSeverity, 7-8
 - glyph reset
 - generates clear alarm via Cooperative Consoles, 7-3
 - generating clear alarms on EM, 7-8
 - schemas, adding new object classes based on, 6-5
 - See also* SunNet Manager, 22-27
- SNM agent and schema files, 8-27
- SNM and SNMP traps, 11-2
- SNM application
 - requirement for EM compatibility, 8-5
 - setting LD_LIBRARY_PATH for, 8-8
- SNM applications
 - adding to EM, 8-5
- SNM database
 - traps, 7-3
- SNM elements
 - types, adding third-party SNM types to EM, 8-7
- SNM events, 7-2
- SNM glyphs
 - converting third-party glyphs to EM format, 8-5
 - traps, 7-3
- snm.conf file, 10-3, 10-6
- snm.conf file, 10-3, 10-5, 10-6
- snm2gdmo compiler, 8-29

- snmAlarmTraps
 - how perceivedSeverity is determined, 7-7
- SNMP, 9-1, 9-3
 - daemon, killing, 4-5
 - daemon, starting, 4-6
 - MIBs, 1-17
 - MIBs, compiling, 9-3
 - MPA, proxy agent, 9-3
 - proxy
 - receiving responses, 10-6
 - retry-interval, 10-6
 - requests, 9-3
 - schemas, 10-3
 - SendTrap condition, 14-8
 - support, in Solstice EM, 2-11
 - supported, 1-16
 - trap
 - additional text information, 11-12
 - changing severity, 11-11
 - default event notification, 11-12
 - default trap-mapping, 11-8
 - enterprise field, 11-6
 - enterprise mapping blocks, 11-14
 - generic values, 11-10
 - incoming, 11-13
 - mapping blocks, 11-18
 - mapping file location, 11-26
 - mapping records, 11-18
 - mapping records format, 11-19
 - probable cause values, 11-11
 - severity values, 11-10
 - specifying source of alarm, 11-9
 - user-configurable capability, 11-14
 - variable bindings, 11-6
 - trap daemon
 - starting, 11-4
 - trap daemon translation, 11-2
 - trap_maps file, 11-5
- SNMP MPA, 9-3
- SNMP Trap
 - daemon, 3-13
 - mapping, 3-13
 - severity, 3-13
- SNMP trap
 - and Cooperative Consoles, 7-3
 - custom mapping, 11-25
 - extracting generic type in RCL, 22-38
 - extracting specific type in RCL, 22-39
 - forwarding, 11-2
 - generic types of, 22-39
 - monitoring with Nerve Center requests, 3-14
 - operationem_snmp-trap
 - daemon, 11-1
 - structure, 11-5
 - use in device management, 3-12
- SNMP trap types, table, 11-6
- SNMP Traps, 3-12
 - enterprise-specific logs, creating, 3-16
- snmp.schema, 10-3
- snmp-mibII.schema, 10-3
- SnmpPingBackoffReachable, 6-11
- SNMPv2
 - files, 10-11
 - SMI, 10-10
 - translation program, 10-11
- Solaris x86
 - RPC agents for, 6-5
- specifying source of alarm, 11-9
- starting/stopping the trap daemon, 11-4
- statement blocks
 - in Request Condition Language, 20-12
- states
 - definition of, 18-11
- subscription
 - use in requests, 3-14
- SunLink
 - CMIP 8.2, 2-16
 - configuring CMIP, 12-8
- SunNet Manager
 - 2.2 compatibility, 6-12
 - alarms, receiving via Cooperative Consoles, 7-7
 - event priorities, mapping to
 - perceivedSeverity, 22-31
 - event requests, data types for thresholds, 22-30
 - glyph reset traps, 7-3
 - topology traps, 7-3
- SunNet Manager event requests
 - initiating via RCL, 22-27
 - relations for thresholds, 22-29
 - specifying thresholds in, 22-28
 - structure of, 22-27
- support tools, for application development, 1-19
- system variable
 - \$count, 14-6
 - \$eventInfo, 14-8
 - \$eventTime, 14-6
 - \$eventType, 14-8
 - \$messageType, 14-6

T

- TCP/IP network
 - use of RPC agents within, 6-1
- Telecommunications Management Network (TMN), 2-16
- Telecommunications Management Networks, 1-16
- Third Party Applications
 - Konfig, 8-14
 - Landmark Performance Works, 8-22
 - Optivity, 8-16
 - Remedy, 8-13
 - set up on hosts with new IP address or name, 8-11
- threshold
 - cleared, 13-2
 - critical, 13-2
 - major, 13-2
 - minor, 13-2
 - warning, 13-2
- time
 - on MIS host, retrieving in RCL, 22-17
- timeout values
 - changing, 12-9
- TMN, 1-16
- tools, custom application development, 1-19
- Topology Nodes, 4-1
- topoNodeMOSet attribute, 4-1
- topoNodeSeverity
 - synchronization with highest uncleared alarm, 4-2
- transition
 - definition of, 18-12
- trap daemon
 - how to start/stop, 11-4
- trap mapping
 - event types, 11-17
 - file location, 11-26
 - formats, 11-19
- trap variable bindings, 11-6
- trap_maps file, 11-13
- trap_maps file, 11-5, 11-13
- trap-mapping
 - default, 11-8
- traps, SNMP
 - See SNMP traps, 3-12
- TrapSpecificType, 20-5

- user variables, 14-6
- user-configurable trap-mapping, 11-14

V

- Viewer configuration file
 - udpating for SNM types, 8-7

W

- warning, CMIP MPA threshold, 13-2
- WHILE loops in RCL
 - example of, 20-13

U

- uptime, 3-8

