

HP OpenView ServiceCenter Automation for HP OpenView Network Node Manager

For the UNIX and Windows Operating Systems

Software Version: 3.1

User Guide

Document Release Date: November 2006

Software Release Date: November 2006



Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

© Copyright 1996-2006 Hewlett-Packard Development Company, L.P.

Trademark Notices

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation. UNIX® is a registered trademark of The Open Group.

HP OpenView ServiceCenter, HP OpenView ServiceCenter Automation, and HP OpenView Network Node Manager are registered trademarks of Hewlett-Packard Company.

Contents

Chapter 1	Preface	5
	Product Overview	5
	Product components.	6
	Prerequisite knowledge	6
Chapter 2	Introduction	7
	Compatibility	7
	Event integration	7
	Inventory integration	9
	GUI integration (cut-throughs)	10
	Context insensitive menu items.	10
	Context sensitive menu items	12
Chapter 3	Installation	15
	Installation Requirements	15
	Installing on Windows	17
	Installing on UNIX	22

Chapter 4	Configuration	25
	Configuring using ECMA scripts	25
	General ECMA techniques used.	26
	ServiceCenter and NNM Messaging Factory objects	28
	ServiceCenter bridge object	29
	OpenView bridge object	33
	NNM topology bridge object	33
	ServiceCenter scEvent object.	34
	NNM Event object	39
	Customizing event integration	39
	Customizing the Interface Queue Manager	40
	Customizing the OpenView Trap Monitor	44
	Customizing the ServiceCenter Events Monitor	48
	Customizing inventory integration	51
Chapter 5	Operation	53
	Starting and Stopping SCAuto for NNM.	53

Preface

This preface covers the following topics:

- Product overview
- Prerequisite knowledge
- Contacting HP

Product Overview

Welcome to HP ServiceCenter Automation (SCAuto) for HP OpenView Network Node Manager (NNM). This product is part of the suite of SCAuto interface products that integrate ServiceCenter with premier network and systems management tools.

This guide describes how to implement SCAuto for NNM for integration with ServiceCenter.

Additional information about SCAuto can be found in the *ServiceCenter Automation Applications for Windows NT and UNIX Guide*.

Product components

SCAuto for NNM 3.1 facilitates integration with ServiceCenter. The product consists of:

- Event Integration – bi-directional events integration that opens, updates, and closes incident tickets in ServiceCenter and posts informational updates back into the NNM Console by default.
- Inventory Integration – allows initial population of manage modes items to ServiceCenter Configuration Management, as well as event driven inventory updates from NNM.
- GUI Integration – allows the NNM operator to launch a ServiceCenter Web client from the NNM Windows menus and icons.

SCAuto for NNM 3.1 is standardized on Sun Microsystem's Java for rapid development and cross platform compatibility. It is integrated to ServiceCenter using Event Services, and with NNM using the OVSNDMP programming API. The API is called from a custom Java Native Interface shared library. The adapter implements an events monitor process that mediates the transport of events to/from ServiceCenter to achieve total connectivity and fault tolerance during outages of either or both ServiceCenter and NNM. The adapter processes are OVSPMD compatible and will be started and stopped through the OVSPMD facility. GUI Integration provides control of the adapter, as well as the ServiceCenter Web client through NNM's Windows pull-down menus and window icons.

Prerequisite knowledge

This guide assumes you have:

- Working knowledge of ServiceCenter applications, ServiceCenter Client/Server, and NNM operating systems. While some procedures for these applications are explained, others are referenced. Refer to the appropriate ServiceCenter documentation for a more detailed explanation.
- (As an Administrator) a thorough knowledge of the operating system where the product will be installed and implemented, as well as a basic understanding of ServiceCenter applications and Event Services.

1 Introduction

CHAPTER

This chapter introduces HP OpenView ServiceCenter Automation (SCAuto) for HP OpenView Network Node Manager (NNM).

The following topics are covered:

- Compatibility
- Event Integration
- Inventory Integration
- GUI Integration

Compatibility

SCAuto for NNM version 3.1 is compatible and tested with NNM (up to version 7.5) on the platforms listed below:

- Windows 2000/XP/2003
- Solaris versions 8 and 9
- HP-UX versions 11.0, 11.11, and 11.23

Event integration

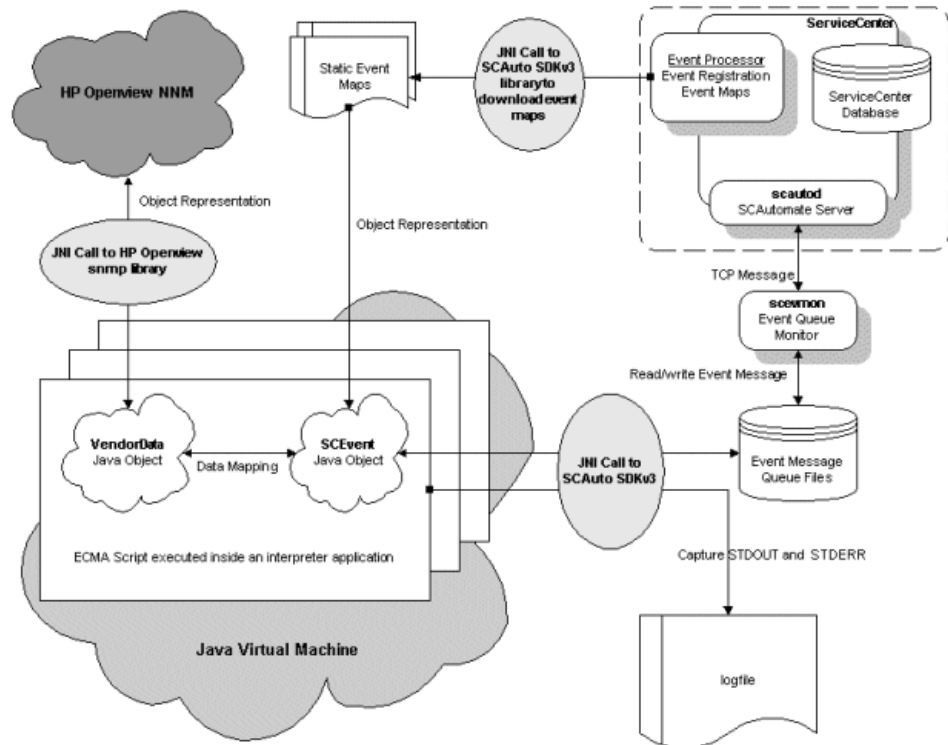
Event integration is implemented using a combination of programmable ECMA scripts (JavaScript) referencing static ASCII map files positionally defining the

event slot names used in ServiceCenter Event Services event types. The ECMA script interpreter is embedded in a Java class and is executed in the Java Virtual Machine environment. This design provides us flexibility in configuration, as well as cross-platform stability in the adapter. The Java embedded interpreter allows the ECMA scripts to access Java class objects and methods directly. Default map files compatible with ServiceCenter event types are provided. The map files by default are downloaded from ServiceCenter every 24 hours. The section Customizing Event Integration in Chapter 3 further describes the procedures for customizing the integration.

The ECMA scripts provide a programmable environment that drive the adapter, looping and blocking for events from NNM. The events from NNM are received in an NNM event object and passed to an ECMA function to be mapped into a ServiceCenter event object that subsequently gets serialized into an event cache queue file. The cached event is then picked up by an event monitoring process and transported to ServiceCenter. To facilitate integration, there are three processes (shown in table below).

Process	Description
Interface Event Queue Monitor (scevmonNNM)	This is the gatekeeper process that mediates events between ServiceCenter and NNM. Therefore, it needs to be running all the time for the events integration to function. The management of this process is done through OVSPMD. You can either start it using the <i>ovstart scevmonNNM</i> command line or by using the drop-down ServiceCenter-Interface Manager menu item on NNM Windows. This process is configured through NNMJ.INI as well as <i>scautoj.properties</i> .
OpenView Trap Monitor (scfromOV)	This is the process that uses the OpenView SNMP API (<i>ovsnmp</i>) to communicate with the NNM server for events received. It will then execute ECMA scripts to map and log these events as ServiceCenter events in the <i>scevents.to.<sc_host><sc_port></i> event queue file. Then, the Interface Event Queue Manager will pick it up and forward it to ServiceCenter.
ServiceCenter Event Monitor (sctoOV)	This is the process that uses the SCAuto SDK API to read and parse the <i>scevents.from.<sc_host><sc_port></i> event queue file. It will then execute ECMA scripts to map and send these events as the SNMP OV_Message type to NNM. This is the default behavior, and is customizable. The original event is received by the Interface Event Queue Monitor and logged to the queue file.

This illustration shows the Event Integration processes.



Inventory integration

Inventory integration consists of two functional sections:

- Initial static inventory gathering
- Dynamic inventory updates

The generated inventory events (icmServer event types in ServiceCenter) are cached in the `scevents.to.<sc host><sc port>` event queue file and forwarded to ServiceCenter by the Interface Queue Manager (`scevmonNNM`) process. Subsequent inventory updates generate ServiceCenter `icmu` and `icmd` events and make use of the Trap Monitor (`scfromOV`) process as well. The initial static inventory gathering makes use of the NNM `ovtopodump -rl` command

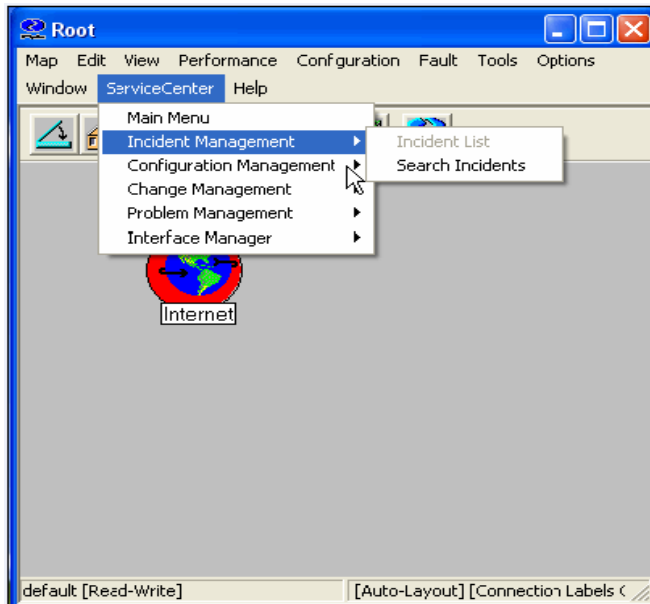
and parses its output for information. The dynamic inventory is received as SNMP traps and processed through the same Events Integration cycle.

GUI integration (cut-throughs)

GUI integration (or cut-throughs) are menu items on the NNM window that allow invocation of the ServiceCenter Web client. There are context insensitive menu items that do not require a node on the OpenView Windows to be highlighted, and there are context sensitive items that are available when a node is highlighted.

Context insensitive menu items

This graphic shows the context insensitive menu items.

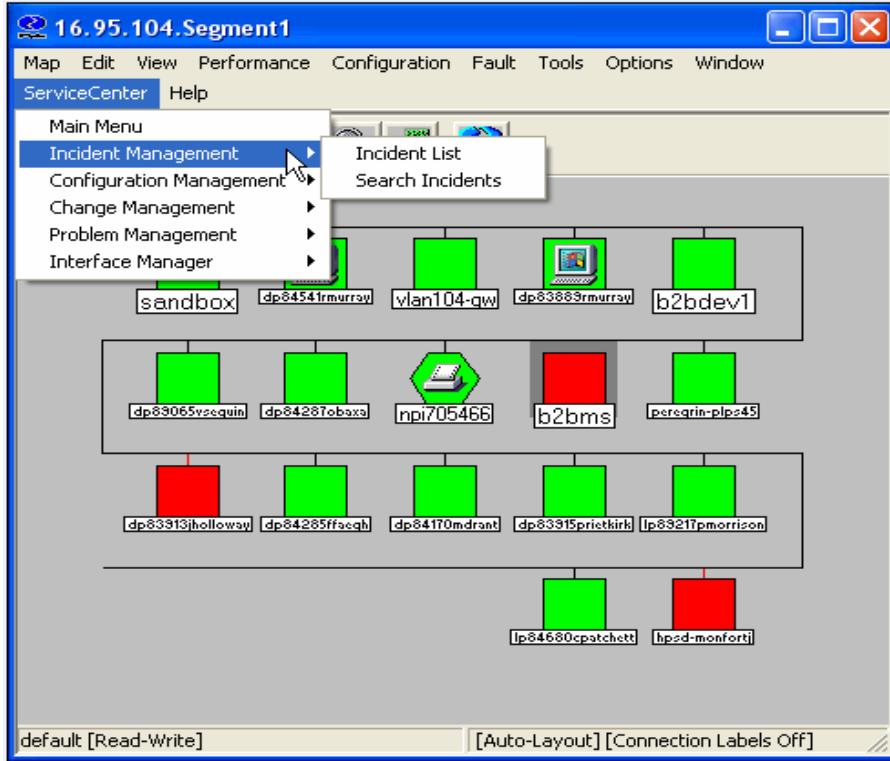


The context insensitive menu items do not depend on node(s) being highlighted to operate because they do not use the node name as an input to a ServiceCenter query/insert.

Menu Item	Description
ServiceCenter - Main Menu	Open a ServiceCenter client and present the ServiceCenter Main Menu.
ServiceCenter - Incident Management	Open a ServiceCenter client and present the Search Incidents option.
ServiceCenter - Configuration Management	Open a ServiceCenter client and show the Configuration Management options.
ServiceCenter - Change Management	Open a ServiceCenter client and show the Change Management options.
ServiceCenter - Problem Management	Open a ServiceCenter client and present the Problem Management options.
ServiceCenter - Interface Manager	Open a ServiceCenter client and present the Interface Manager options.

Context sensitive menu items

This graphic shows the context sensitive menu items.



The context sensitive menu items are only available (not grayed out) when a node or nodes are highlighted in the OpenView windows. The items are executed in the context of the currently highlighted node.

Menu Item	Description
ServiceCenter - Main Menu	Open a ServiceCenter client and present the ServiceCenter Main Menu.
ServiceCenter - Incident Management	Open a ServiceCenter client and present the Incidents List and Search Incidents options.
ServiceCenter - Configuration Management	Open a ServiceCenter client and show the Configuration Management options.
ServiceCenter - Change Management	Open a ServiceCenter client and show the Change Management options.

Menu Item	Description
ServiceCenter - Problem Management	Open a ServiceCenter client and present the Problem Management options.
ServiceCenter - Interface Manager	Open a ServiceCenter client and present the Interface Manager options.

2 Installation

CHAPTER

This chapter explains how to install the HP OpenView ServiceCenter Automation (SCAuto) for HP OpenView Network Node Manager (NNM) adapter. The installation requires a Graphical User Interface (GUI) such as Windows or X-Windows on UNIX systems and has the same look and feel on all OS platforms. The installation uses a wizard-type format that prompts you for installation parameters to configure the adapter. At the end of the installation, a post install ECMA script is executed to finish the configuration.

The following topics are covered:

- Installation requirements
- Installing on Windows
- Installing on UNIX

Installation Requirements

To install SCAuto for HP NNM, the following are required:

- A graphical user interface such as Windows or X-Windows (on UNIX).
- You must logged in as the root user or a user with local administrative rights for Windows.
- For UNIX only, set the JAVAHOME environment variable to the path of the Java JRE. For example, /opt/java1.4/jre or /usr/j2se/jre.

- The OpenView bin directory must be in the path. For example, /opt/OV/bin or C:\OpenView\bin.
- You must have the required parameter values, as shown below:

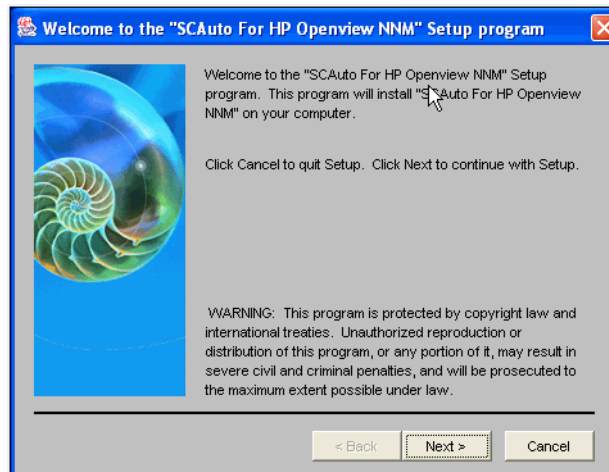
Parameter Name	Description	Example Value
Destination directory	The installation directory for the SCAuto adapter	c:\Program files\Peregrine Systems\SCAutoOpenViewNNM /opt/ov/SCAutoOpenViewNNM
OpenView Home Directory	The home directory of OpenView.	C:\Program Files\HP OpenView /opt/OV
OpenView Server Host Name	This is the name of the entity to which SNMP requests are sent on this session. This may be an IP hostname, an IP address, an IPX address (on Windows NT only) or a non-SNMP entity which is proxied by another SNMP agent system. The SNMP Configuration Database is consulted to determine if the specified peername is proxied by another system. If so, SNMP requests are actually directed to the proxy system on behalf of the peername. For most installations, it is the host name of the NNM server.	hpux_11_host 172.17.7.234
Full path to OpenView JRE	JRE directory installed with HPOpenViewNNM	c:\Program files\HP OpenView\jre\jre1.4 /opt/ov/jre/jre1.4
SCAuto for NNM User/Group	The user name and group to own the ServiceCenter Automation for NNM files. This has the format of <username/id> : <groupname/id>	jdoe:user 123:456 UNIX only.
ServiceCenter Server Host	The ServiceCenter host for connecting to the server using the ServiceCenter Web client.	hpux_11_host 172.17.7.234
SCAuto Server Port	The port number for the ServiceCenter Automation Daemon listening process.	scautod 12690
ServiceCenter Web-tier Host	The ServiceCenter host for connecting to the server using the ServiceCenter Web client.	hpux_11_host 172.17.7.234
ServiceCenter Web-tier Port	The ServiceCenter port on the ServiceCenter host used for connecting to the server using the ServiceCenter Web client.	8080

Installing on Windows

To install SCAuto for HP NNM on Windows:

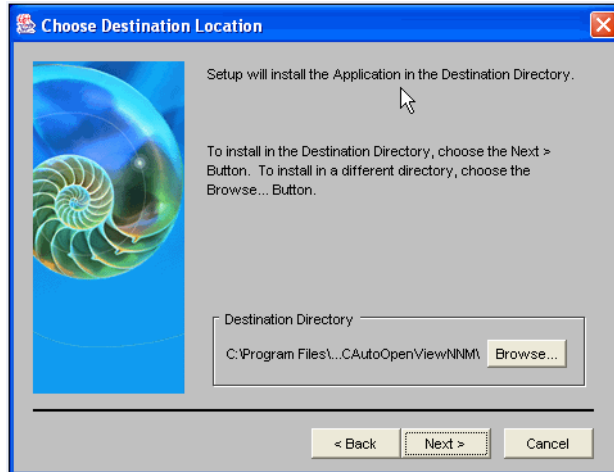
- 1 Login as Administrator (any user with Administrative rights).
- 2 Unzip the file SCAutoNNM_31_windows.zip.
- 3 Execute the installation batch file (install.bat).

The system runs until the Welcome screen opens.



4 Click **Next**.

The Choose Destination Location screen opens.

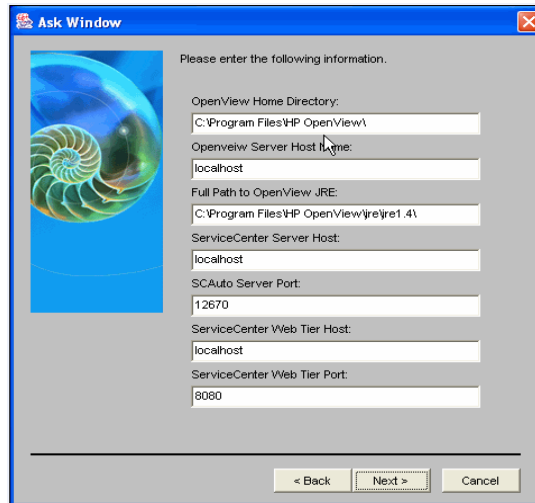


- 5 Select a destination path to install the interface files by performing one of the following steps:
 - Choose the default destination of:
C:\Program Files\Peregrine Systems\SCAutoOpenViewNNM

Click the **Browse** button to install in a different location. A new directory also can be created using Browse, or you can use an existing path.

6 Click **Next**.

The Ask Window screen opens, where you enter the parameters for your installation.



Ask Window

Please enter the following information.

OpenView Home Directory:
C:\Program Files\HP OpenView\

OpenView Server Host Name:
localhost

Full Path to OpenView JRE:
C:\Program Files\HP OpenView\jre1.4\

ServiceCenter Server Host:
localhost

SCAuto Server Port:
12670

ServiceCenter Web Tier Host:
localhost

ServiceCenter Web Tier Port:
8080

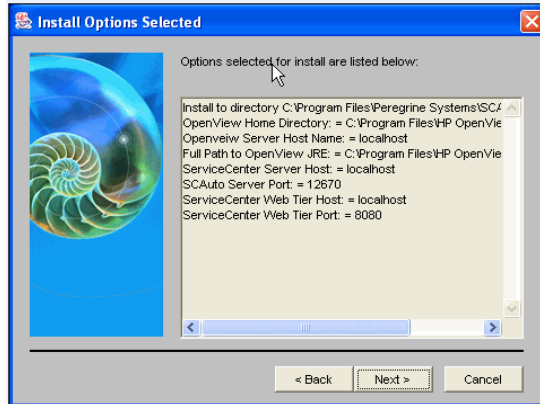
< Back Next > Cancel

7 Choose the parameters that apply to your installation (see the table on page 22 to help you decide).

Note: On UNIX systems, an additional parameter is needed to indicate the username and group that will own the files being installed. When entering this parameter, separate the values with a colon : (for example, user:group).

8 Click **Next**.

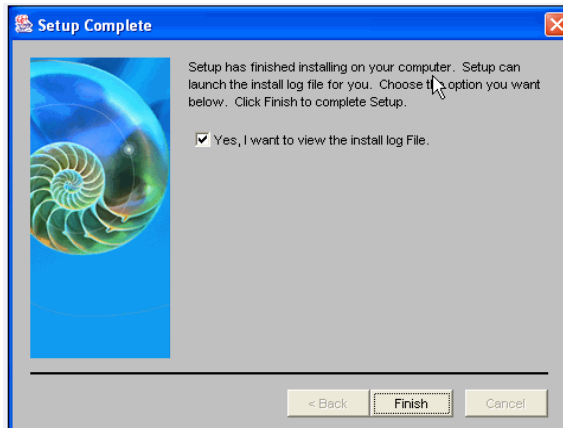
The Install Options Selected screen opens. It shows the values you just entered.



- 9 Confirm the values are correct.
 - If the values are correct, click **Next**.
 - If you want to modify the values, click **Back** to make your changes. Then, click **Next** when you return to this confirmation screen.

A splash screen opens, followed by an installation progress dialog. After all the files are copied over, a post-install Javascript is executed to configure the product.

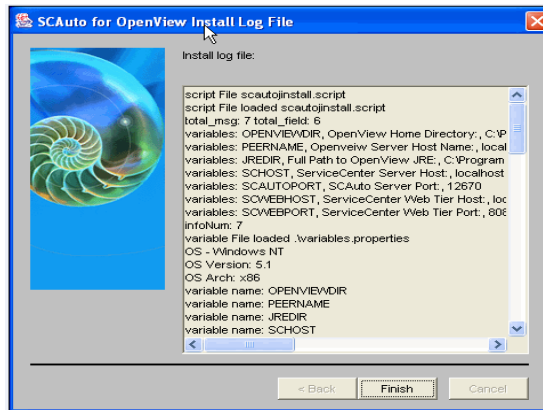
The Setup Complete screen opens.



10 Indicate whether you want to see the installation log. HP strongly recommends you view this file to confirm that installation of the product was successful.

- If you want to see the log, leave the check box checked.
- If you do not want to see the log, un-check the box and click Finish.

If you left the log file checkbox checked, you will see the Installation Log File screen.



11 Scan the log and confirm that the installation was successful. Click **Finish** when you are done.

This completes the installation of SCAuto for NNM.

Installing on UNIX

To install SCAuto for NNM on UNIX:

- 1 Login as the superuser root.
- 2 Depending upon your system, uncompress and use the appropriate file:
 - On Solaris, SCAutoNNM_31_solaris.tar.Z.
 - On HP-UX (11.0 and 11.11), SCAutoNNM_31_hp_11.tar.Z.
 - On HP-UX Itanium (11.23), SCAutoNNM_31_hp_11_ia64.tar.Z.

- 3 Run the installation script file. The name of the script is `install.sh`.

The system runs until the Welcome screen opens.

- 4 Go to [Step 4 on page 18](#) and continue with the installation steps shown. The remaining steps are nearly identical for Windows and UNIX.

3 Configuration

CHAPTER

This chapter describes how to configure HP OpenView ServiceCenter Automation (SCAuto) for HP OpenView Network Node Manager (NNM), which involves modifying:

- ECMA scripts
- ServiceCenter and NNM messaging factory objects
- Event integration

Configuring using ECMA scripts

The ECMA scripting engine used in SCAuto for NNM is FESI v1.1.1 (Free ECMA Script Interpreter). It is an embedded interpreter in Java. This means that from within the ECMA script, it is possible to instantiate Java classes and call Java methods directly. This is the basis of how SCAuto for NNM works.

There are engineered Java object representations of ServiceCenter event objects and OpenView NNM trap/event objects. Each Java object is furnished with methods to connect, retrieve, and send itself through its connection to ServiceCenter or the NNM Server. The availability of these Java objects, combined with the real time programmability of ECMA scripting in a JVM environment, makes it possible to have a system that is flexible, cross-platform, and powerful. The ECMA interpreter is implemented by the ExecuteJS class. An example of executing the script would be:

```
java -classpath  
lib/SCAutoNNMJ.jar:lib/SCAutoJ.jar:lib/xml.jar:lib/fesi.jar  
ExecuteJS writeNNMEvent.js recv_traps.js
```

Putting more than one script on the argument list for the ExecuteJS class allows the execution of scripts from left to right. This means that all functions defined in the first script are available for calling in subsequent scripts.

Besides the regular language syntax for ECMA scripting (<http://www.ecma.ch/stand/ECMA-262.htm>), there are general techniques used in our scripts.

General ECMA techniques used

The general ECMA techniques used are:

- Defining short-cuts for Java classes, variables, and methods.
- Loops
- Keywords Java Class
- writeXML() method in Event objects

These techniques are described in the sections that follow.

Defining short-cuts for Java

This is a convenient syntax to shorten long names into short ones that can be used throughout the scope of the script. It is usually used to shorten class names that contain long package names but can also be used for static Java class methods or variables.

Syntax:

```
<name> = Packages.<package name>.<class name, static method or static variable>;
```

Example:

```
NNMEvent = Packages.NNMEvent;
(define the NNMEvent variable as the Java class NNMEvent - do not use package name)
```

```
String = Packages.java.lang.String;
```

(define String as the Java String)

```
ErrPrintln = Packages.java.lang.System.err.println;
```

(you can use `ErrPrintLn` directly and it will call Java's `System.err.println`)

Loops

Loops are used to implement an endless execution cycle where events/traps are being repeatedly read/processed until a user initiates a shutdown. Because the test condition of the loops are in the beginning of the loop, a user initiated shutdown will not take affect until the last trap/event has been processed.

Syntax:

```
while (<test condition>
{
}
```

Example:

```
while(scBridge.isContinue() && status == 1)
{
    vEvent = new NNMEvent();
    writeLn("waiting for trap ...");
    status = ovsnmpBridge.readEvent(vEvent);
    writeNNMEvent(scBridge, vEvent);
}
```

Keywords Java class

This class is used in conjunction with the `<inst. dir.>/config/scauto.keywords` file. It has a static method `match` that can be called to match a keyword section header (event type) with any string. It is used in the NNM Trap Monitor `writeNNMEvent.js` script to match received SNMP trap format strings with keyword headers to decide which ServiceCenter event to generate at runtime.

Syntax:

```
boolean Keywords.match(arg1, arg2); // arg1 and arg2 are of type
java.lang.String
```

Example:

```
if(Keywords.match("NONE", vEvent.getEvField("format_string")))
{
    writeLn("skipping event " + vEvent.getEventType());
    return;
}
```

writeXML() method in Event objects

This method is available for both the ServiceCenter event object as well as the NNM event object. This method is used to output an XML representation of the event to standard output. In this release, it is used mainly for debugging purposes to show the contents of the events as they flow through the system.

Syntax:

```
eventObj.writeXML();
```

Example:

```
vEvent = new NNMEvent();
writeln("waiting for trap ...");
status = ovsnmpBridge.readEvent(vEvent);
vEvent.writeXML();
```

ServiceCenter and NNM Messaging Factory objects

The ServiceCenter Messaging Factory class defines an interface for creating a ServiceCenter Bridge object dynamically defined by the `scautoj.SCMessagingClassName` and `scautoj.SCMessagingClassFile` properties in the `scautoj.properties` file. This allows new objects to be loaded without changing the base code of the applications..

Class Method: **void SCMessagingFactory()
void VendorMessagingFactory()**

Definition: Constructor, creates a new SCMessagingFactory object.

Arguments: None

Returns: New SCMessagingFactory/VendorMessagingFactory object.

Class Method: **SCAutoJ.JNIBridge newMessagingObject()**

Definition: Factory method that instantiates an object of class SCAutoJ.JNIBridge (by default) and returns it.

Arguments: None

Returns: Instance of SCAutoJ.JNIBridge class (defined by property).

Class Method:	OVSNMPBridge newMessagingObject()
Definition:	Factory method that instantiates an object of class OVSNMPBridge (by default) and returns it.
Arguments:	None
Returns:	Instance of OVSNMPBridge class (defined by property).

ServiceCenter bridge object

The ServiceCenter bridge object provides methods for sending and receiving events from ServiceCenter. It is created by the SCMessagingFactory class object by calling the newMessagingObject() method of the factory class.

Class Method:	int init()
	int init(String sessId, String user, String appName, Stringapp Version, String IniFile, String server)
Definition:	Initializing methods. In future implementations, we might only need to use the non-arg init(), since the other one is needed to interface with SDK. This method is specific to SDK, it initializes the SDK connection to ServiceCenter using Java Native Interface calls to the shared library.
Arguments:	sessId - represents the session ID eg. SC_SMS. user - represents the user of this method. appName - the name of the application calling this method. appVersion - the version number of the application. iniFile - the ini file for SDK to parse for parms. server - the SCAuto server.port identification.
Returns:	status - the status from initializing SDK.
Example:	scFactory = new SCMessagingFactory(); scBridge = scFactory.newMessagingObject(); scBridge.init("OpenView", "NNMJUSER", "SCAutoNNMJ", "3.1", "NNMJ.INI", "godzilla.12690");

Class Method:	int connect(String events, String users)
Definition:	Connects immediately to ServiceCenter so that any subsequent calls to write, read or take events will not be cached or logged.

Arguments: events - event filter list.
users - user filter list.

Returns: status

Class Method: **int disconnect()**

Definition: Disconnects immediately from ServiceCenter so that any subsequent calls to write, read or take events will now be cached and logged.

Arguments: none

Returns: status

Class Method: **int writeQEvent(String direction, SCEvent scEvent)**

Definition: Write a queued event message. In SDK implementation, the event is formatted and written to the *scevents.to.** file and picked up by the *scevmonNNM* process. Other implementations will vary.

Arguments: direction - the direction of event flow in respect to ServiceCenter. Valid values are *to* and *from*.
scEvent - the ServiceCenter event object of class *SCAutoJ.SCEvent* containing field values to be written.

Returns: status

Class Method: **int writeEvent(String direction, SCEvent scEvent)**

Definition: Write a non-queued event message. In SDK implementation, the event is sent directly to ServiceCenter. This is as if the connect and disconnect methods were called implicitly.

Arguments: direction - the direction of event flow in respect to ServiceCenter. Valid values are *to* and *from*.
scEvent - the ServiceCenter event object of class *SCAutoJ.SCEvent* containing field values to be written.

Returns: status

Class Method: `int takeQEvent(String direction, SCEvent scEvent)`

Definition: Take a queued event message, the event is removed or marked as read after it is read. In SDK implementation, the event is read from the `scevents.from.file` written by the `scevmonNNM` process. Other implementations will vary.

Arguments: `direction` - the direction of event flow in respect to ServiceCenter. Valid values are *to* and *from*.
`scEvent` - the ServiceCenter event object of class `SCAutoJ.SCEvent` containing field values to be written.

Returns: `status`

Class Method: `int takeEvent (String direction, SCEvent scEvent, String syncFileName)`

Definition: This is a connected mode method to directly read the events from the target without logging. This method implements a *syncfile* to cache the last read record. This method forms a *syncfile* name and calls the overloaded function *takeEvent*. The form of the *syncfile* name is *syncfile.hostname.port*.

Arguments: `direction` - the direction of event flow in respect to ServiceCenter. Valid values are *to* and *from*.
`scEvent` - the ServiceCenter event object of class `SCAutoJ.SCEvent` containing field values to be written.
`syncFileName` - the full path to a syncfile to write the event sequence number (in SDK, it is the sequence number from the *eventout* file).

Returns: `status`

Class Method: `int takeEvent (String direction, SCEvent scEvent)`

Definition: This is a connected mode method to directly read the events from the target without logging. In the SDK implementation, its a hit or miss thing whether SC is up or not. This method implements a *syncfile* to cache the last read record. This method forms a *syncfile* name and calls the overloaded function *takeEvent*. The form of the *syncfile* name is *syncfile.hostname.port*.

Arguments: `direction` - the direction of event flow in respect to ServiceCenter. Valid values are *to* and *from*.

Returns: `status`

Class Method:	int readEvent(String direction, SCEvent scEvent, String sequenceNumber)
Definition:	This is a connected mode method to directly read the events from the target without logging. In the SDK implementation, its a hit or miss thing whether SC is up or not. This method will not remember where it last read from and will always start from the sequence number provided.
Arguments:	direction - the direction of event flow in respect to ServiceCenter. Valid values are <i>to</i> and <i>from</i> . scEvent - the ServiceCenter event object of class <i>SCAutoJ.SCEvent</i> containing field values to be written. sequenceNumber - the sequence number of the event to start reading from (in SDK, this is the sequence number from the <i>eventout</i> file).
Returns:	status
Class Method:	void logPrint(String logMessage)
Definition:	Write a log message to the default log file.
Arguments:	logMessage - the message to print out to the log file.
Returns:	none
Class Method:	void terminate()
Definition:	Terminating the session. In SDK implementation, this calls <i>scauto_term</i> that cleans up most things including freeing the <i>scautoHandle</i> .
Arguments:	none
Returns:	none
Class Method:	boolean isContinue()
Definition:	Checks to see if the user has requested termination of the process, if <i>yes</i> , then returns <i>true</i> , otherwise returns <i>false</i> .

Arguments: direction - the direction of event flow in respect to ServiceCenter. Valid values are *to* and *from*.
 Returns: continue - *true* or *false*.

OpenView bridge object

The OpenView bridge object provides methods for sending and receiving events from NNM. It is created by the VendorMessagingFactory class object by calling the newMessagingObject() method of the factory class. It provides instance methods for communicating with NNM and returns/reads its data in the form of the NNM Event Object.

Class Method: `int writeEvent(String direction, VendorEvent vEvent)`

Definition: Writing the NNM event out. This method creates a new event in the NNM product.

Arguments: direction - this is currently not used and may be any string.
 vEvent - the NNM event to write out (send).

Returns: status

Class Method: `int readEvent (VendorEvent vEvent)`

Definition: Reads an event (trap) from NNM.

Arguments: vEvent - the NNM event (trap) that is read in.

Returns: status

Class Method: `void terminate()`

Definition: Terminate the session.

Arguments: none

Returns: none

NNM topology bridge object

This is the object used to receive topology data from NNM on demand. It actually executes the NNM command `ovtopodump -r1` and parses its output for

device information. This command is configured in the `scauto nmj.ovtotoDumpCommand` property in the `scautoj.properties` file.

Class Method: `OVTOTOBridge()`

Definition: Constructor. Instantiates a new object of the same class.

Arguments: none

Returns: OVTOTOBridge object with call to *new*.

Class Method: `int init(String name)`

Definition: Initializes the object with a session name.

Arguments: name - the name of this session.

Returns: status

Class Method: `int readEvent(VendorEvent vEvent)`

Definition: Reads a topology event from OpenView. Subsequent calls to `readEvent` will return devices until the return status != 1 which means that there are no more events.

Arguments: vEvent - the NNM event containing device information.

Returns: status - 1 = success, otherwise no events returned.

Class Method: `void terminate()`

Definition: Terminates this session.

Arguments: none

Returns: none

ServiceCenter scEvent object

The ServiceCenter Event Object is the data communication medium for sessions connecting to ServiceCenter server using the ServiceCenter Bridge Object. Internally, it auto detects the event type and formats its field names automatically to positionally match with ServiceCenter's position dependent

event type definitions. For the user, it provides instance methods for accessing and assigning values to these fields.

Class Method: **SCEvent()**
SCEvent(String eventType, String mapFileName)

Definition: Constructor. Creates a new instance of this event object.

Arguments: eventType - the string representing a ServiceCenter event type eg. pmo, pmu, pmc, ICMServer, etc.
mapFileName - the absolute path name of the static mapfile to use.

Returns: new instance of SCEvent object.

Class Method: **String toString()**

Definition: Returns a formatted string of field values of the event.

Arguments: none

Returns: String

Class Method: **void setEventSystemSequence(String eventSystemSequence)**

Definition: Sets the system sequence number of this event being sent to ServiceCenter. It is equivalent to the system sequence number of any inbound ServiceCenter event.

Arguments: eventSystemSequence - the sequence number. Max characters is 1.

Returns: none

Class Method: **void setEventUserSequence(String eventUserSequence)**

Definition: Sets the user sequence number of this event being sent to ServiceCenter. It is equivalent to the user sequence number of any inbound ServiceCenter event.

Arguments: eventUserSequence - the sequence number. Max characters is 33.

Returns: none

Class Method: `void setEventSystemOption(String eventSystemOption)`

Definition: Sets the system option string of this event being sent to ServiceCenter. It is equivalent to the system option string of any inbound ServiceCenter event.

Arguments: eventSystemOption - the system option. Max characters is 25.

Returns: none

Class Method: `void setEventUser(String eventUser)`

Definition: Sets the event user of this event being sent to ServiceCenter. It is equivalent to the event user of any inbound ServiceCenter event.

Arguments: eventUser - the event user. Max characters is 25.

Returns: none

Class Method: `void setEventPassword(String eventPassword)`

Definition: Sets the event password of this event being sent to ServiceCenter. It is equivalent to the event password of any inbound ServiceCenter event.

Arguments: eventPassword - the event password. Max characters is 25

Returns: none

Class Method: `void setEventSeparatorCharacter(char eventSeparatorCharacter)`

Definition: Sets the event separator character of this event being sent to ServiceCenter. It is equivalent to the event separator character of any inbound ServiceCenter event. The default is the ^ character.

Arguments: eventSeparatorCharacter - the event separator character. Max characters is 1.

Returns: none

Class Method: `void setEvField(String name, String value)`
Definition: Gets the event field of this event to contain the name/value pair. If the name does not exist, it will create a new field, if it exists, it will overwrite the value. This is equivalent to the evfields value of an inbound ServiceCenter event.
Arguments: name - the event field name
value - the value of the field.
Returns: none

Class Method: `String getEventSystemSequence()`
Definition: Sets the system sequence number of this event. It is equivalent to the system sequence number of any outbound ServiceCenter event.
Arguments: none
Returns: the sequence number. Max characters is 33.

Class Method: `String getEventUserSequence()`
Definition: Gets the user sequence number of this event. It is equivalent to the user sequence number of any outbound ServiceCenter event.
Arguments: none
Returns: the user sequence number. Max characters is 33.

Class Method: `String getEventSystemOption()`
Definition: Gets the system option of this event. It is equivalent to the system option of any outbound ServiceCenter event.
Arguments: none
Returns: the system option. Max characters is 25.

Class Method: **String getEventUser()**

Definition: Gets the event user of this event. It is equivalent to the event user of any outbound ServiceCenter event.

Arguments: none

Returns: the event user. Max characters is 25.

Class Method: **String getEventPassword()**

Definition: Gets the event password of this event. It is equivalent to the event password of any outbound ServiceCenter event.

Arguments: none

Returns: the event password. Max characters is 25.

Class Method: **char getEventSeparatorCharacter()**

Definition: Gets the separator character of this event. It is equivalent to the separator character of any outbound ServiceCenter event.

Arguments: none

Returns: the separator character. Max characters is 1.

Class Method: **String getEvField(String name)**

Definition: Gets the event field of this event for the specified event field name. If the name does not exist, it will return a null string. This is equivalent to the *evfields* value of an outbound ServiceCenter event.

Arguments: name - the event field name.

value - the value of the field.

Returns: none

NNM Event object

The NNM Event Object is the data communication medium for sessions connecting to NNM Server using the OVSNMP API. It provides instance methods for accessing and assigning values to the event fields.

Class Method: **VendorEvent()**

Definition: Constructor. Creates a new instance of a *VendorEvent* that can be used to contain NNM specific fields.

Arguments: none

Returns: new instance of the *VendorEvent* object.

Class Method: **String getEvField(String name)**

Definition: Gets the event field value, given the field name.

Arguments: name - the name of attribute value to return as a string.

Returns: the value of the attribute (field) as a String.

Class Method: **void setEvField(String name, String value)**

Definition: Sets the corresponding *EventField* object in the *eventFields* Vector identified by the name to the value.

Arguments: name - the name of the attribute to set.

value - the value of the attribute to set.

Customizing event integration

SCAuto for NNM can be customized for startup/shutdown behavior, and data mapping between ServiceCenter and NNM. These customizations can be done by modifying certain initializing files, Java property files as well as ECMA scripts (JavaScript). The following pages show the areas of customization and their related files.

Customizing the Interface Queue Manager

The Interface Queue Manager is the process that monitors and caches events from/to ServiceCenter and NNM. The queue manager will continue to cache all events in the situation when one of the connecting software is down. It can be described as an event pump that will accumulate events during a system down, and continue pumping events when the system is up again..

File: `<inst. dir.>/config/scevmon.lrf`

Description: This file contains an NNM Local Registration File entry for registering with OVSPMD. After modifying this file, you must execute the `ovdelobj <LRF filename>` command to remove the previous object, and `ovaddobj <LRF filename>` to add the new definition back in. Basically, the default definition registers the `scevmonNNM` object to not start by default (`OVs_NO_START`), as an independent executable daemon process (`OVs_DAEMON`) startable by executing the script `scevmon.sh` (`scevmon.bat` on Windows), and is stopped by executing `stopscevmon.sh` (`stopscevmon.bat` on Windows).

Customization:

- 1) To make the Interface Queue Manager start automatically during `ovstart`, replace the keyword `OVs_NO_START` with `OVs_YES_START`.
- 2) To make this object dependent on another object or process, enter the object's name in the second field of the definition line after `OVs_NO_START`, using the colon (`:`) as the field delimiter.
- 3) After making these modifications, execute `ovdelobj scevmon.lrf` and then `ovaddobj scevmon.lrf` to affect your changes.

File: `<inst. dir.>/bin/scevmon.sh(bat)`

Description: This is the script or batch file that is specified to execute in the `scevmon.lrf` LRF file. Its main purpose is to set up the correct library paths (UNIX) or DLL paths (Win) to enable the Java Runtime Environment to execute.

Customization: It is not suggested that the customer modify this file.

File:	<inst. dir.>/scautoj.properties	
Description:	This is a Java property file that is used by all the Java processes including the Interface Queue Manager. The entries are name/value pairs separated by the colon (:) character.	
Customization:	Property Name	Property Value
	scevmon.Key	This is the license key that enables the product to function. Not Modifiable.
	scevmon.SleepInterval	This is a sleep interval in number of seconds for the queue manager to pause before attempting to read the next event from both the <i>to</i> and the <i>from</i> queues when there are no events available. If there are events available in any queue, it will finish processing these events before sleeping. Default: 5
	scevmon.IniFile	This variable contains the name of the initializing file to read for the SCAuto SDK . This property facilitates backward compatibility with the SCAuto SDK through the use of JNI (Java Native Interface). Default: NNMJ.INI
	scevmon.LogError	This is an error logging flag to enable debug messages to be output to the logfile <i>NNMJ.log</i> . The name of the log file is configured in the initializing file specified by the <i>scevmon.IniFile</i> property. A value of <i>0</i> in this field will turn off all debugging. 0 - debugging off. 1 - level 1 debugging. Default: 1
	scevmon.SessionID	This is the session ID used by ServiceCenter Event Services to identify incoming SCAuto communication sessions. Not Modifiable.

scevmon.From_SC	This flag specifies whether to enable incoming event processing from ServiceCenter to NNM. on - enable. off - disable. Default: on
scevmon.To_SC	This flag specifies whether to enable outgoing event processing from NNM to ServiceCenter. on - enable. off - disable. Default: on

File: <inst. dir.>/NNMJ.INI

Description: This initializing file contains traditional SCAuto parameters that are used by the SCAuto SDK library functions. The entries are name/value pairs separated by the colon (:) character.

Customization:	Property Name	Property Value
	log	The log file name where all debugging information as well as error messages will be redirected to. Default: NNMJ.log
	scevent.server	The SCAuto server hostname/port number to connect to. It is in the format of <host name.><port number>. This value is modified during installation. If after installation, the SCAuto host/port has changed, please modify this entry to reflect the change. Default: modified during installation.
	debug	A debug flag for SCAuto. true - turn debug on. false - turn debug off. Default: commented out, false.

debugscautoevents	<p>A debug level flag for SCAuto events.</p> <p>0 - off. 1 - level 1. 2 - level 2. Default: commented out, 0.</p>
scevents	<p>Specifies an optional list of event types which are to be retrieved from ServiceCenter's EVENTOUT queue. The default is to retrieve all types, however, this can be inefficient, because it can result in bringing over certain types of events, such as outbound page messages or email messages, which have no meaning to NNM. To restrict the types of events that are retrieved, code them in a list separated by commas and enclosed in parentheses. For example, scevents : (pmo , pmu , pmc).</p>
scevusers	<p>This parameter restricts the events which are to be retrieved from ServiceCenter's EVENTOUT queue based on the value in evuser field. The default is to retrieve all events regardless of the value of the evuser field, however, this can be inefficient, because it can result in bringing over certain events which are not intended for NNM. To use this parameter, code the values in a list separated by commas and enclosed in parentheses. For example, scevusers : falcon only get events that have falcon as the evuser. scevusers : (falcon , NNMJUSER) only get events that have falcon or NNMJUSER as the evuser.</p>

Customizing the OpenView Trap Monitor

The OpenView Trap Monitor process has the responsibility of receiving SNMP traps from the NNM Server and logging it to a queue file, `scevents.to.<host.port>`. The Interface Queue Manager process (if running) then picks it up and forwards it to ServiceCenter..

File: `<inst. dir.>/config/scfromOV.lrf`

Description: This file contains an NNM Local Registration File entry for registering with OVSPMD. After modifying this file, you must execute the `ovdelobj <LRF filename>` command to remove the previous object, and `ovaddobj <LRF filename>` to add the new definition back in. Basically, the default definition registers the `scfromOV` object to not start by default (`OVs_NO_START`), as an independent executable daemon process (`OVs_DAEMON`) startable by executing the script `scfromOV.sh` (`scfromOV.bat` on Windows), and is stopped by executing `stopscfromOV.sh` (`stopscfromOV.bat` on Windows).

Customization:

- 1) To make the OpenView Trap Monitor start automatically during `ovstart`, replace the keyword `OVs_NO_START` with `OVs_YES_START`.
- 2) To make this object dependent on another object or process, enter the object's name in the second field of the definition line after `OVs_NO_START`, using the colon (`:`) as the field delimiter.
- 3) After making these modifications, execute `ovdelobj scfromOV.lrf` and then `ovaddobj scfromOV.lrf` to affect your changes.

File: `<inst. dir.>/bin/scfromOV.sh(bat)`

Description: This is the script or batch file that is specified to execute in the `scfromOV.lrf` LRF file. Its main purpose is to set up the correct library paths (UNIX) or DLL paths (Win) to enable the Java Runtime Environment to execute.

Customization: It is not suggested that the customer modify this file.

File: `<inst. dir.>/scautoj.properties`

Description: This is a Java property file that is used by all the Java processes including the OpenView Trap Monitor. The entries are name/value pairs separated by the colon (`:`) character.

Customization:	Property Name	Property Value
	scautoj.SCMessagingClassName	This is the java class name including the package name of the class that defines the ServiceCenter Messaging class. This should be provided by the installer and not modified after installation.
	scautoj.SCMessagingClassFile	This is the absolute path name to the class file that contains the ServiceCenter Messaging class. Not modifiable by user.
	scautoj.VendorMessagingClassName	This is the java class name including the package name of the class that defines the NNM Messaging class. This should be provided by the installer and not modified after installation.
	scautoj.VendorMessagingClassFile	This is the absolute path name to the class file that contains the ServiceCenter Messaging class. Not modifiable by user.
	scautonnmj.peerName	This is the name of the entity to which SNMP requests are sent on this session. This may be an IP hostname, an IP address, an IPX address (on Win only) or a non-SNMP entity which is proxied by another SNMP agent system. The SNMP Configuration Database is consulted to determine if the specified peername is proxied by another system. If so, SNMP requests are actually directed to the proxy system on behalf of the peername. This variable is set during installation and not modifiable by the user. For most installations, it is the host name of the NNM server.
	scautonnmj.eventFilter	This is an SNMP OID style wildcard for filtering the trap IDs that should be received by this interface. Valid syntax: *, *.123.*, 123.*, 123* etc. Default: *

scautonnmj.trapdConfFileName	This is the absolute path name of the NNM <i>trapd.conf</i> file. It is parsed for trap details and format strings. This value is modified upon installation and should not be modified unless the file has moved.
scautonnmj.keywordsFileName	This is the absolute path name of the keywords used to assign event types by matching the format string in the trap to Perl 5 style regular expression matching. See the <i>scauto.keywords</i> entry below for detail. This should not be modified unless you want to use a different keywords file.

File: `<inst. dir.>/bin/recv_traps.js`

Description: This is the ECMA script (Java Script) that drives the OpenView Trap Monitor process. This script is executed after *writeNNMEvent.js* (see below), because it uses the *writeNNMEvent(..)* function defined in *writeNNMEvent.js* to generate the incident ticket event. The content of this script can be functionally separated into 3 sections:

- 1) Initializing variables and objects.
- 2) Implementing a blocking read in a loop to read SNMP traps from NNM.
- 3) Terminating connections and exiting.

File: <inst. dir.>/bin/writeNNMEvent.js

Description: This script basically contains a JavaScript function that will be called by *recv_traps.js*. It is *sourced* by placing it as the first argument to *ExecuteJS* followed by *recv_traps.js*. In this function *writeNNMEvent (...)* is where decisions are made in regards to which ServiceCenter event type to generate from the event/trap received from NNM. This is also the function that maps fields from NNM to ServiceCenter events. By default decisions on which event type to map to are decided by calling the *match* method of the *Keywords* class.

The *Keywords* class is used in conjunction with the <inst. dir.>/config/scauto.keywords file. It has a static method *match* that can be called to match a keyword section header (event type) with any string. It is used in the NNM Trap Monitor *writeNNMEvent.js* script to match received SNMP trap format strings with keyword headers to decide which ServiceCenter event to generate at runtime.

Syntax: boolean Keywords.match(arg1, arg2); // arg1 and arg2 are of type java.lang.String

Example:

```
if(Keywords.match("NONE",
vEvent.getEvField("format_string")))
{
  writeln("skipping event " + vEvent.getEventType());
  return;
}
```

At the end of this function, the *writeQEvent* method of the ServiceCenter Bridge Object (that was passed in) is called to *write* the event to a queue file, that will be picked up by the Interface Queue Manager to be forwarded to ServiceCenter.

File: `<inst. dir.>/config/scauto.keywords`

Description: This is a keywords file that uses Perl5 regular expression syntax to specify wildcard matching specifications. Please view this file for more information and tips on how to expand it.

Format:
The sections labelled PROBOPEN, PROBCLOSE, TOPOADD, TOPODEL, and NONE are provided by default and are used in the mapping ECMA scripts (eg., *writeNNMEvent.js*).
Inside each section is a list of keywords, each on its own line. Each section is terminated with a semicolon (;). These keywords are Perl5 regular expression compatible. Anything appearing after a # is treated as a comment.
Order is not important. You may extend the number of keywords in this file by simply adding more so that you may use it in the mapping ECMA script.

Example:

```
...
NONE:
# Ignore these keywords.
(?:)^\.*OV_Message.*$
#(?:)^\.*Temperature.*$
TOPOADD:
(?:)^\.*Node added.*$
(?:)^\.*Seg added.*$
(?:)^\.*Net added.*$
(?:)^\.*IF.*added.*$
(?:)^\.*SNA object discovered.*$
(?:)^\.*SNA object configuration changed.*$
....
```

Customizing the ServiceCenter Events Monitor

The ServiceCenter Events Monitor process has the responsibility of receiving ServiceCenter Events from ServiceCenter through reading a queue file, *scevents.from.<host.port>*. The Interface Queue Manager process (if currently running) connects to ServiceCenter, retrieves any outbound events, and writes to this queue file.

File: `<inst. dir.>/scautoj.properties`

Description: This is a Java property file that is used by all the Java processes including the ServiceCenter Events Monitor. The entries are name/value pairs separated by the colon (;) character.

Customization:

Property Name	Property Value

scautoj.SCMessagingClassName	This is the java class name including the package name of the class that defines the ServiceCenter Messaging class. This should be provided by the installer and not modified after installation.
scautoj.SCMessagingClassFile	This is the absolute path name to the class file that contains the ServiceCenter Messaging class. Not modifiable by user.
scautoj.VendorMessagingClassName	This is the java class name including the package name of the class that defines the NNM Messaging class. This should be provided by the installer and not modified after installation.
scautoj.VendorMessagingClassFile	This is the absolute path name to the class file that contains the ServiceCenter Messaging class. Not modifiable by user.
scautonnmj.oveventCommand	This property sets the absolute path to the ovevent command that is used to send NNM traps to NNM.

File: <inst. dir.>/config/sctoOV.lrf

Description: This file contains a Local Registration File entry for registering with OVsPMD. After modifying this file, you must execute the `ovdelobj <LRF filename>` command to remove the previous object, and `ovaddobj <LRF filename>` to add the new definition back in. Basically, the default definition registers the `sctoOV` object to not start by default (`OVs_NO_START`), as an independent executable daemon process (`OVs_DAEMON`) startable by executing the script `sctoOV.sh` (`sctoOV.bat` on Win), and is stopped by executing `stopsctoOV.sh` (`stopsctoOV.bat` on Win).

Customization:

- 1) To make the ServiceCenter Events Monitor start automatically during `ovstart`, replace the keyword `OVs_NO_START` with `OVs_YES_START`.
- 2) To make this object dependent on another object or process, enter the object's name in the second field of the definition line after `OVs_NO_START`, using the colon (`:`) as the field delimiter.
- 3) After making these modifications, you must execute `ovdelobj sctoOV.lrf` and then `ovaddobj sctoOV.lrf` to affect your changes.

File: `<inst. dir.>/bin/sctoOV.sh(bat)`

Description: This is the script or batch file that is specified to execute in the *sctoOV.lrf LRF* file. Its main purpose is to set up the correct library paths (UNIX) or DLL paths (Win) to enable the Java Runtime Environment to execute.

Customization: It is not suggested that the customer modify this file.

File: `<inst. dir.>/bin/recv_scevents.js`

Description: This is the ECMA script (Java Script) that drives the ServiceCenter Events Monitor process. This script is executed after *sendNNMEvent.js* (see below), because it uses the *sendNNMEvent(..)* function defined in *sendNNMEvent.js* to generate the OpenView trap. The content of this script can be functionally separated into 3 sections:

- 1) Initializing variables and objects.
- 2) Implementing a read in a loop to read ServiceCenter events from the *from* queue file.
- 3) Terminating connections and exiting.

File: `<inst. dir.>/bin/sendNNMEvent.js`

Description: This script is *sourced* before the *recv_scevents.js* script so that *recv_scevents.js* can use the function *sendNNMEvent()*. It is *sourced* by placing it as the first argument of *ExecuteJS*.

The function *sendNNMEvent(...)* contained in this script is used to map ServiceCenter event fields into an NNM trap (Warnings trap by default) that can be sent to the NNM server. Internally, it calls the *ovevent* NNM command to send the trap. This command is configured in the *scautoj.properties* file as the *scautonnmj.oveventCommand* property name.

File: `<inst. dir.>/EventMap/From_SC/*.map`

Description: These are the static map files used to construct the ServiceCenter Event Objects. By default, a handful of static maps is shipped with the product, however, maps are downloaded every 24 hours from the last time it was installed to make sure that the event configurations are current. This functionality is hidden from the user and does not need configuration.

Customizing inventory integration

Inventory integration is defined as two parts:

- An initial Inventory gathering that pulls all the known devices from OpenView Topology database.
- A subsequent SNMP trap based update that modifies ServiceCenter's inventory database when nodes are added/removed and notified via SNMP traps.

Since the dynamic update of Inventory is SNMP trap based and falls under the same area as Event Integration, please refer to that section for information about the files for configuring it.

File: `<inst. dir.>/bin/inventory.sh(bat)`

Description: This is the shell script (Win batch file) that gets executed when you choose the Gather Inventory menu option under the ServiceCenter-Interface Manager menu. Its main purpose is to set up the correct library paths (UNIX) or DLL paths (Win) to enable the Java Runtime Environment to execute.

File: `<inst. dir.>/bin/inventory.js`

Description: This is the ECMA script that drives the initial inventory gathering. The `writeNNMEvent.js` script is *sourced* first, so that the `writeNNMEvent(...)` function can be used in this script to generate ServiceCenter events and write to the *to* queue file.

4 Operation

CHAPTER

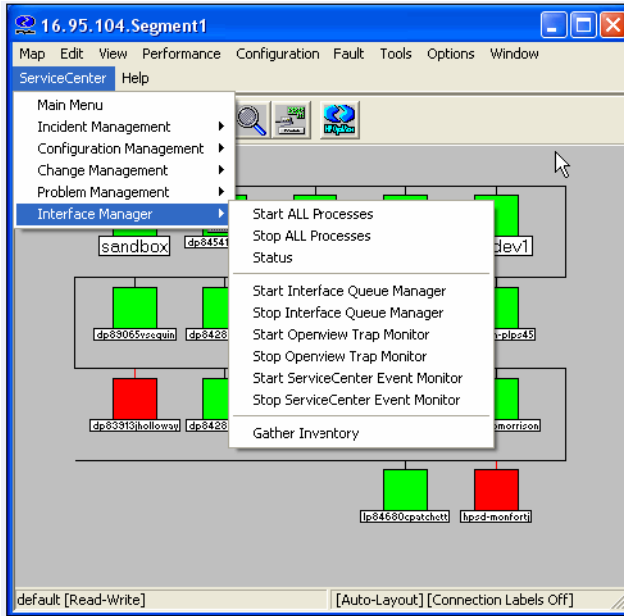
This chapter describes the how to start and stop HP OpenView ServiceCenter Automation (SCAuto) for HP OpenView Network Node Manager (NNM).

Starting and Stopping SCAuto for NNM

Starting and stopping the SCAuto for NNM adapter can be done either through the OVSPMD ovstart and ovstop commands, or through the Interface Manager Menu items in NNM. The processes are added to the NNM OVSPMD database using custom LRF files provided in the config directory. You may use these template LRF files to customize start up and shutdown behavior for your site. See NNM's online help (or LRF man pages on UNIX) for customizing options.

Once the processes are started, it is operational by default installation and will begin to open incident tickets in ServiceCenter based on OV_Node_Down and OV_DataCollectThresh SNMP traps received by NNM. See the section [Customizing Event Integration in Chapter 3](#) for details on modifying this configuration. After the incident ticket has been opened by ServiceCenter, an OV_Message event will be posted to NNM confirming the status of the incident.

Subsequent updates and closure of the incident posts similar OV_Message events to NNM.



The Interface Manager menu has the following command options:

Command	Description
Start ALL Processes	Starts the Interface Queue Manager, OpenView Trap Monitor, and ServiceCenter Events Monitor processes.
Stop ALL Processes	Stops the Interface Queue Manager, OpenView Trap Monitor, and ServiceCenter Events Monitor processes.
Status	Lists the start/stop status for the Interface Queue Manager, OpenView Trap Monitor, and ServiceCenter Events Monitor processes.
Start Interface Queue Manager	Starts the Interface Queue Manager Process.
Stop Interface Queue Manager	Stops the Interface Queue Manager Process.

Command	Description
Start OpenView Trap Monitor	Starts the OpenView Trap Monitor.
Stop OpenView Trap Monitor	Stops the OpenView Trap Monitor.
Start ServiceCenter Event Monitor	Starts the ServiceCenter Event Monitor.
Stop ServiceCenter Event Monitor	Stops the ServiceCenter Event Monitor.
Gather Inventory	Starts the initial Inventory Gathering process to prime ServiceCenter <i>icm</i> database.



i n v e n t

11/30/06