Peregrine

SCAuto

SCAuto for SPECTRUM Guide

Version 5.0

PART NO: DSCA-5-EN08

Peregrine SYSTEMS®

Peregrine Systems, Inc.
3611 Valley Centre Drive San Diego, CA 92130
Tel 800.638.5231 or 858.481.5000
Fax 858.481.1751
www.peregrine.com

Content

Preface

Overview

Welcome to Peregrine Systems' *SCAuto for Aprisma SPECTRUM*. This product is part of the suite of SCAuto (SCAutomate) interface products that integrate ServiceCenter with premier network and systems management tools.

This guide describes how to implement SCAuto for Aprisma's SPECTRUM for integration with Peregrine Systems' ServiceCenter.

Additional information about SCAuto can be found in the *SCAuto Applications for Windows NT and Unix guide*.

Prerequisite knowledge

This guide assumes you have:

- Working knowledge of ServiceCenter applications, ServiceCenter Client/Server, and Aprisma SPECTRUM operating systems. While some procedures for these applications are explained, others are referenced. Refer to the ServiceCenter documentation for a more detailed explanation.
- Working knowledge of a GUI or text-based environment.
- (As an Administrator) a thorough knowledge of the operating system where the product will be installed and implemented.

Important: ServiceCenter installation requirements are specific to the operating system of the computer where ServiceCenter is being installed. These requirements are listed in their respective installation guides.

Contacting Peregrine Systems

For further information and assistance with this release, you can download documentation or schedule training.

Customer Support

For further information and assistance, contact Peregrine Systems' Customer Support at the Peregrine CenterPoint Web site.

To contact customer support:

1. In a browser, navigate to http://support.peregrine.com
2. Log in with your user name and password.
3. Follow the directions on the site to find your answer. The first place to search is the KnowledgeBase, which contains informational articles about all categories of Peregrine products.
4. If the KnowledgeBase does not contain an article that addresses your concerns, you can search for information by product; search discussion forums; and search for product downloads.

Documentation Web site

For a complete listing of current SCAuto documentation, see the Documentation pages on the Peregrine Customer Support Web.

To view the document listing:

1. In a browser, navigate to http://support.peregrine.com.
2. Log in with your login user name and password.
3. Click either **Documentation** or **Release Notes** at the top of the page.
4. Click the SCAuto link.

5 Click a product version link to display a list of documents that are available for that version of SCAuto.

6 Documents may be available in multiple languages. Click the Download button to download the PDF file in the language you prefer.

You can view PDF files using Acrobat Reader, which is available on the Customer Support Web site and through Adobe at http://www.adobe.com.

Important: Release Notes for this product are continually updated after each release of the product. Ensure that you have the most current version of the Release Notes.

Education Services Web Site

Peregrine Systems offers classroom training anywhere in the world, as well as "at your desk" training via the Internet. For a complete listing of Peregrine's training courses, refer to the following web site:

http://www.peregrine.com/education

You can also call Peregrine Education Services at +1 858.794.5009.

CHAPTER 1

Introduction

SCAuto facilitates the integration of Aprisma's SPECTRUM with Peregrine Systems' ServiceCenter Service. The SCAuto for Aprisma's SPECTRUM product consists of:

- Alarm Integration – open, update, and close incident tickets in ServiceCenter based on alarms generated by SPECTRUM.
- Inventory Integration – allows initial (one-time) population of network inventory items to ServiceCenter asset management.
- SCEvMon - sends events to ServiceCenter.
- GUI Integration (cut-throughs) – allows the SPECTRUM operator to open, update, close, and view incident tickets using SPECTRUM's menus and icons.

This chapter is an introduction to SCAuto for Aprisma's SPECTRUM and covers the following topics:

Overview

SCAuto for Aprisma's SPECTRUM is standardized on Sun Microsystem's Java JDK 1.2.2 for rapid development and cross-platform compatibility. It is integrated to ServiceCenter using Event Services and with SPECTRUM via the command line interface (CLI) interface and AlarmNotifier. The CLI interface and AlarmNotifier are called from a custom Java Native Interface shared library. The adapter implements an event that mediates the transport of events to ServiceCenter to achieve total connectivity and fault tolerance during outages of either or both ServiceCenter and SPECTRUM. The adapter processes are started and stopped through the SpectroGRAPH facility. GUI Integration is achieved by customization of SpectroGRAPH's CsStdMenu file to provide control of the adapter, as well as the ServiceCenter client through SpectroGRAPH's pull-down menus and windows icons.

Compatibility

SCAuto for SPECTRUM is compatible and tested with SPECTRUM version 6.0 - 6.6 on Windows 2000, Windows NT, and Solaris 7 and 8.

Alarm Integration

Alarm Integration uses a combination of programmable ECMA scripts (JavaScript) referencing static ASCII map files compatible with ServiceCenter event types. The alarms from SPECTRUM are received in a SPECTRUM 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.

The ECMA script interpreter is embedded in a Java class and executes in the Java Virtual Machine environment. The Java embedded interpreter allows the ECMA scripts to access Java class objects and methods directly. The ECMA scripts provide a programmable environment to drive the adapter, waiting for alarms from SPECTRUM.

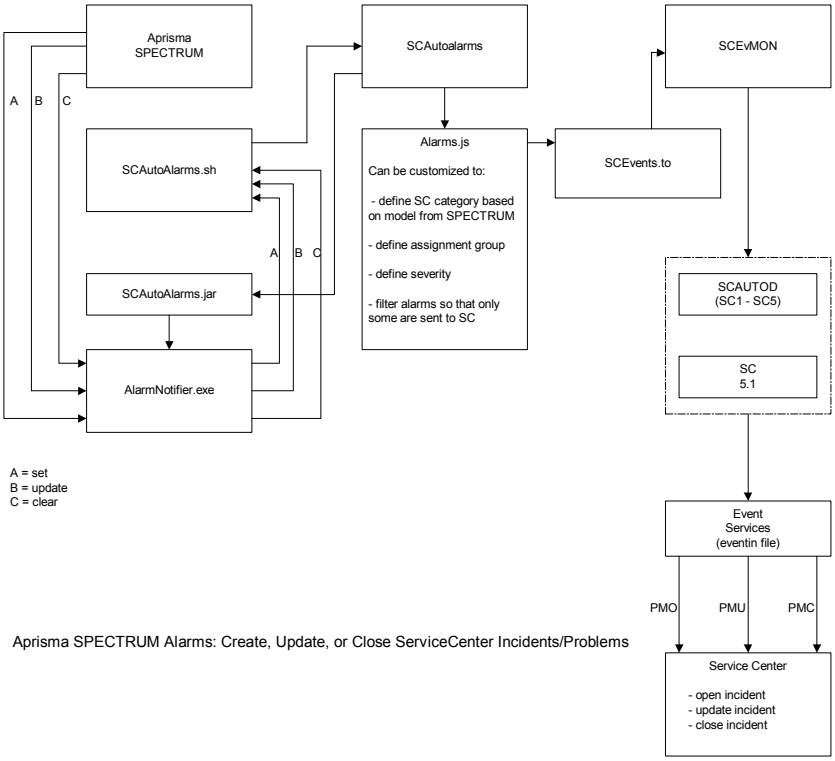
By default, the map files are downloaded from ServiceCenter every 24 hours. This interval can be modified in a configuration file. The section *Chapter 3, Customizing Event Integration* further describes the procedures for customizing the integration.

To facilitate integration, there are two processes (shown in Table 1-1).

Table 1-1: Alarm Integration Processes

Process	Description
Interface Event Queue Monitor (SCEvMon)	This is the gatekeeper process that mediates events between ServiceCenter and SPECTRUM. Therefore, it must rung all the time for events integration to function. You can either start it using the command line or by using the drop-down ServiceCenter-Interface Manager menu item on the SpectroGRAPH Windows. This process is configured through SCAutoSpectrum.ini as well as scevmon.properties.
SPECTRUM Alarm Monitor (SCAutoAlarms)	This process uses SPECTRUM's AlarmNotifier application to process alarms detected by the SpectroServer. It then executes ECMA scripts to map and log these alarms as ServiceCenter events in the *scevents.to.<sc_host><sc_port>* event queue file. Then, the Interface Event Queue Manager will pick it up and forward it to ServiceCenter.

The following figure provides an overview the alarm integration of Aprisma SPECTRUM and ServiceCenter using SCAuto.



Aprisma SPECTRUM Alarms: Create, Update, or Close ServiceCenter Incidents/Problems

SCAutoAlarms

SCAutoAlarms supports incident integration. This process:

- Launches the AlarmNotifier
- Uses the scripts in the SpectrumScripts subdirectory to format the data
- Executes the **alarms.js** JavaScript in the EventMap/To_SC subdirectory to map the data into a ServiceCenter event.

You can make any customizations to map or filter (by default based on condition and modeltype from the alarm) the alarm data. The **AlarmNotifierParms** file is a customized version of the .alarmrc used by the AlarmNotifier. You can use any of the AlarmNotifier options you want, but the SET/UPDATE/CLEAR scripts should point to the scripts Peregrine provides. (see *Customizing the SPECTRUM Alarm Monitor* on page 53)

To start the process for alarm monitoring:

- select the menu options under the SCAutomate sub-menu, or optionally,
- start the process from the command line.

Inventory Integration

This section describes the SCAutoIntegration process and how to start the inventory integration process.

SCAutoInventory

SCAutoInventory is for inventory integration. The initial static inventory gathering makes use of SPECTRUM's command line interface and parses its output for information. This process uses **vnmsh** commands to retrieve model data. In the `scautoj.properties` file there is a parameter, *scautospectrumj.modeltypes*, to specify the model types that you wish to inventory. The values are case sensitive and are used to determine the JavaScript to be executed. For example, if you have a value of `Host_Sun` then the `/EventMap/To_SC/Host_Sun.js` JavaScript executes to map the data to a ServiceCenter event. When this processed is launched, it gathers and maps all the model types you have specified and then exits.

Inventory Integration consists of initial inventory collected. The generated inventory events (*icmServer* event types in ServiceCenter) are cached in the *scevents.to.<sc host><scauto port>* event queue file and forwarded to ServiceCenter by the Interface Queue Manager *(SCEvMon)* process.

To start the process for Inventory Integration:

- select the menu options under the SCAutomate sub-menu, or optionally,
- start the process from the command line.

To use the menu:

1 From the Tools menu, select Peregrine Systems

Figure 1-1: Tools Menu

2 Click **SCAutomate** to display the SCAutomate menu.

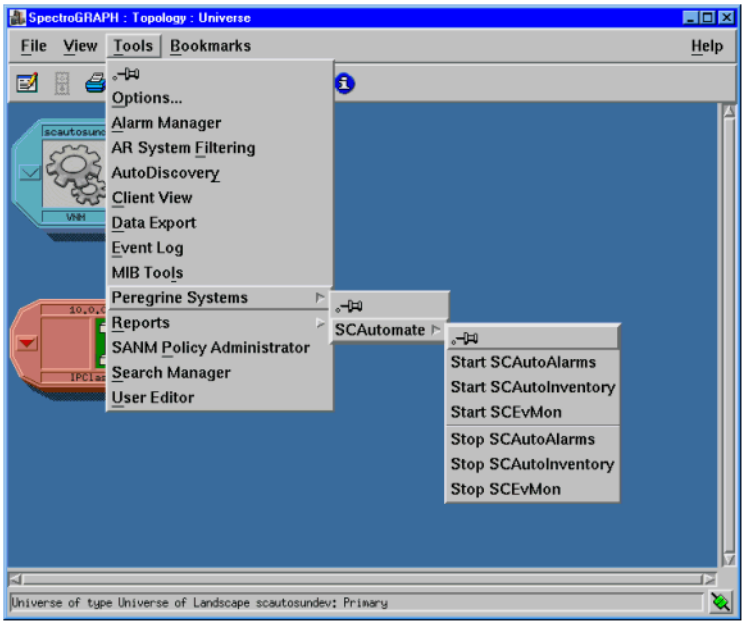


Figure 1-2: SCAutomate Menu

The following list describes the menu items.

Menu Item	Description
Start SCAutoAlarms	Start the Alarm Monitor process by executing the StartSCAutoAlarms.sh script.
Start SCAutoInventory	Start the Inventory Gathering process by executing the StartSCAutoInventory.sh script.
Start SCEvMon	Start the ServiceCenter Event Monitor by executing the StartSCAutoSCEvMon.sh script.
Stop SCAutoAlarms	Stop the Alarm Monitor process by executing the StopSCAutoAlarms.sh script.
Stop SCAutoInventory	Stop the Inventory Gathering process by executing the StopSCAutoInventory.sh script.
Stop SCEvMon	Stop the ServiceCenter Event Monitor by executing the StopSCAutoSCEvMon.sh script.

Note: These scripts are all located in the /<installed>/bin directory.

GUI Integration (cut-throughs)

GUI Integration (or *cut-throughs*) are menu items in SpectroGRAPH that allow access to the ServiceCenter client. There are *context insensitive* menu items that do not require a Model in SpectroGRAPH to be highlighted, and there are *context sensitive* items that are available when a Model is highlighted.

Context insensitive menu items

The following figure shows the context insensitive menu items.

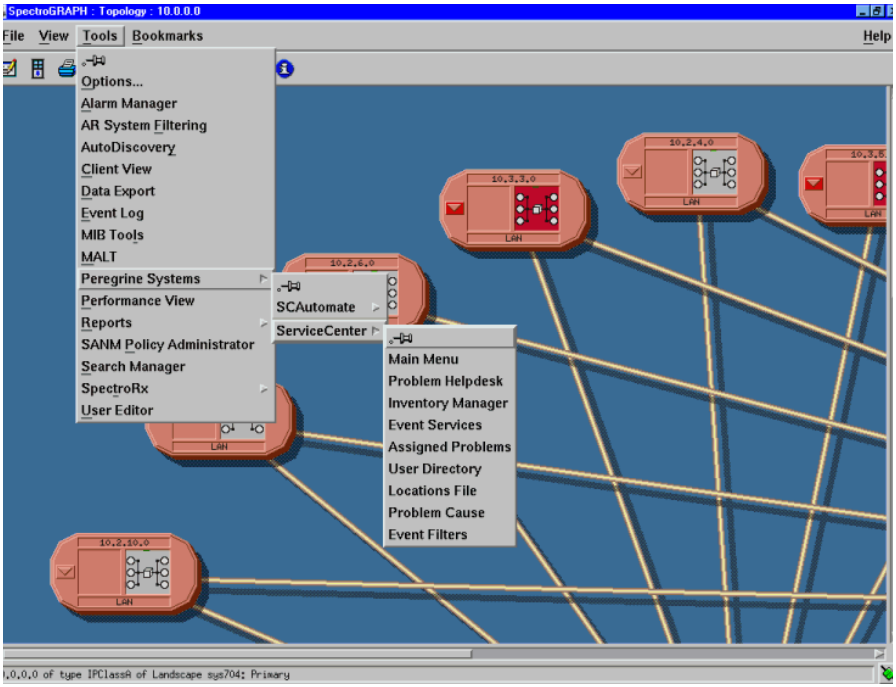


Figure 1-3: Context insensitive menu items

The context sensitive menu items do not depend on a network Model being highlighted (selected) because they do not use the Model name as an input to a ServiceCenter query/insert.

This list shows the static ServiceCenter client screens.

Menu Item	Description
Main Menu	Launch a ServiceCenter client Main Menu.
Incident Help Desk	Launch a ServiceCenter client Help Desk/Incident list.
Inventory Manager	Launch a ServiceCenter client Inventory Menu.
Event Services	Launch a ServiceCenter client Event Services Menu.

ServiceCenter Event Monitor

ServiceCenter Event Monitor (SCEvMon) is the gatekeeper process that mediates events between ServiceCenter and the SCAuto Adapters. It works with SCAuto Java Native Interface Bridge objects, SCAuto Event objects and SCAuto C shared libraries to accomplish this event integration process.

- *SCAuto Java Native Interface Bridge (SCAutoJ)*:

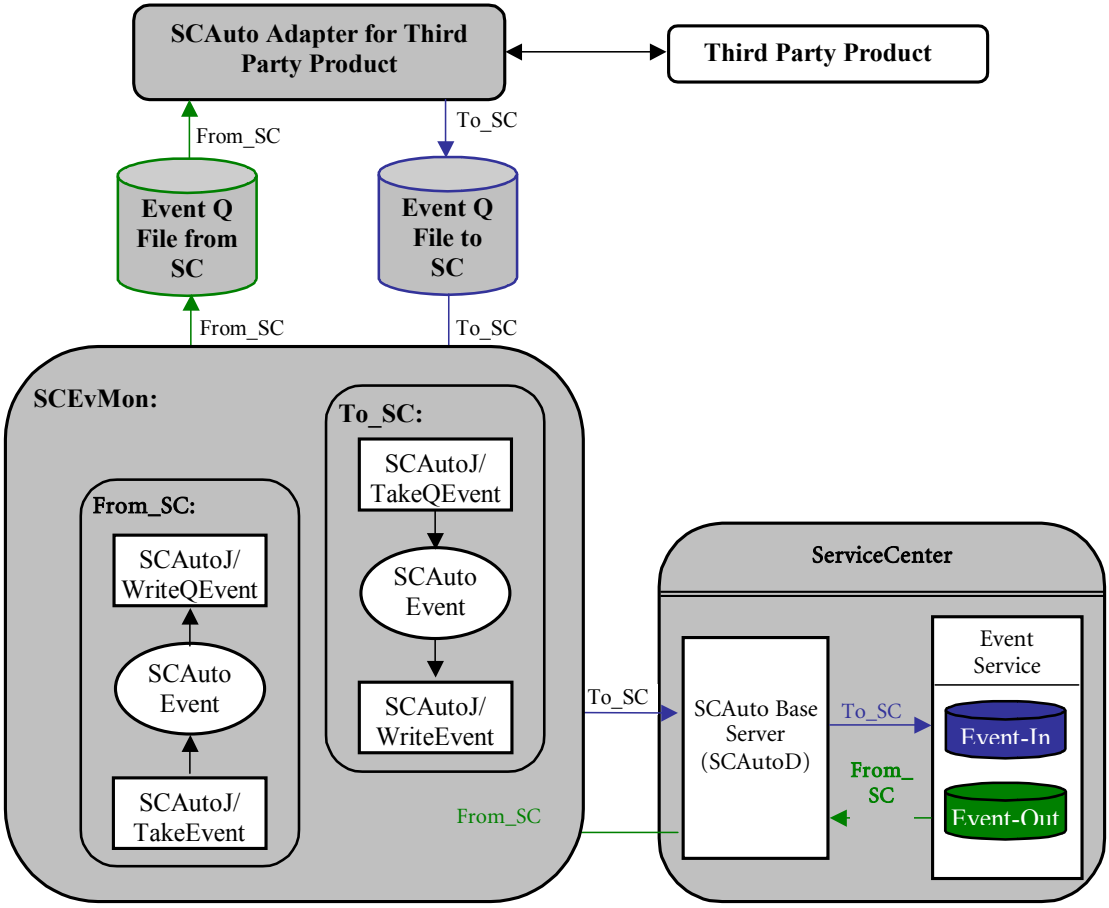
  SCAutoJ contains all the native methods for accessing the C functions in the SCAuto C libraries. It includes all the file I/O operations for SCAuto Adapter's event queue files and ServiceCenter connection and event insert/query operations.

- *ServiceCenter Event Object (SCEvent)*:

  The ServiceCenter Event Object is an event data object that represents the ServiceCenter events.

SCEvMon is a bi-directional events processor and the processes are handled by separate threads. It responds by establishing a connection between SCAuto Adapter and ServiceCenter, querying ServiceCenter events and inserting new SCAuto Adapter events to ServiceCenter. The two events processes are named as To_SC and From_SC. The To_SC reads in events from the SCAuto Adapter's event-to-sc queue file and insert this event into ServiceCenter's event-in queue. The From_SC responses for retrieving ServiceCenter outbound events from the event-out queue and updating the SCAuto Adapter's event-from-sc queue file for SCAuto Adapter to notify the third party product about the change of an event's status.

Note: For SCAuto for SPECTRUM, SCEvMon handles inbound events (events going to ServiceCenter) only.


SCAuto Adapter for Third Party Product
Third Party Product
From_SC
To_SC
Event Q File from SC
Event Q File to SC
From_SC
To_SC
SCEvMon:
From_SC:
SCAutoJ/ WriteQEvent
SCAuto Event
SCAutoJ/ TakeEvent
To_SC:
SCAutoJ/ TakeQEvent
SCAuto Event
SCAutoJ/ WriteEvent
From_SC
To_SC
ServiceCenter
SCAuto Base Server (SCAutoD)
To_SC
From_ SC
Event Service
Event-In
Event-Out

CHAPTER 2 Installation

Overview

SCAuto installation requires a graphical user interface (GUI) such as Windows NT, Windows 2000, or X-Window on UNIX systems. The GUI has the same look and feel on all OS platforms. SCAuto uses an installation wizard that prompts the user for configuration parameters. At the end of the installation, a post-install ECMA script configures the product.

Installation requirements

To install SCAuto for SPECTRUM, the following are required:

- A graphical user interface. X-Window (on UNIX), Windows 2000, or Windows NT.
- You must be executed as the *root* user, or user with local administrative rights for Windows NT or Windows 2000 (i.e., same ID as SPECTRUM or owner of SPECTRUM).
- You must have the required parameter values, as shown in the following table.

Table 2-1: Required Installation Parameters

Parameter Name	Description	Example Value
Destination directory	The installation directory for the SCAuto adapter	C:\winapp32\Spectrum\SCAutoSpectrum \usr\spectrum\SCAutoSpectrum
SPECTRUM Home Directory	The home directory of SPECTRUM. The environment variables of SPECROOT or SSHOME are used by default.	C:\win32app\Spectrum \usr\spectrum
VNM Node Name	The SpectroServer host name.	e.g., suntest_host,10.1.1.110 etc.
VNMSH Path	The path for Spectrum's CLI executables.	C:\win32app\Spectrum\vnmsh \usr\spectrum \vnmsh\
Alarm Notifier	Name and location of SPECTRUM's AlarmNotifier executable.	C:\win32app\Spectrum\Notifier\AlarmNotifier \usr\spectrum\Notifier\AlarmNotifier
Probable Cause Path	The path to SPECTRUM's Probable Cause files.	C:\win32app\Spectrum\SG-Support\CsPCause \usr\spectrum\SG-Support\CsPCause\
SCAuto for SPECTRUM User/Group	The user name and group to own the SCAuto for SPECTRUM files. This has the format of <username/id>:<groupname/id>	spectrum:root, 123:456
ServiceCenter RUN Directory (optional)	The installed ServiceCenter client executable. This is used for cut-through integration.	c:\ProgramFiles\ServiceCenter\RUN \usr\local\ServiceCenter\RUN
ServiceCenter Server Host	The ServiceCenter host for connecting to the server using the ServiceCenter Full Client.	servername, 172.17.7.321
ServiceCenter Client Port (optional)	The ServiceCenter port on the ServiceCenter host for connecting to the server using the ServiceCenter Full Client.	scclient,12670
SCAuto Server Port	The port number for the SCAuto Daemon listening process.	scautod,12690
ServiceCenter Login Name (optional)	This is the ServiceCenter user name to log into the system from a Client. If this user does not have a password, the GUI cut-throughs log in directly to ServiceCenter, otherwise, a password is prompted.	falcon

Installing on Windows NT or Windows 2000

To install SCAuto for SPECTRUM on Windows NT or Windows 2000:

1. Login as *Administrator* (any user with Administrative rights).
2. Insert the SCAuto for SPECTRUM CD into your CD-ROM drive.
3. Change directory to the CDROM drive and execute the installation batch file (*nt_install.bat*).

   The system runs until the Welcome screen appears.

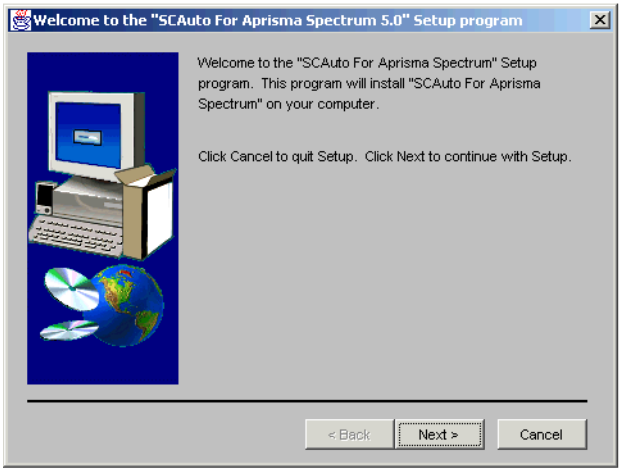


Figure 2-1: Welcome screen

4. Click **Next.**

The Choose Destination Location screen appears.

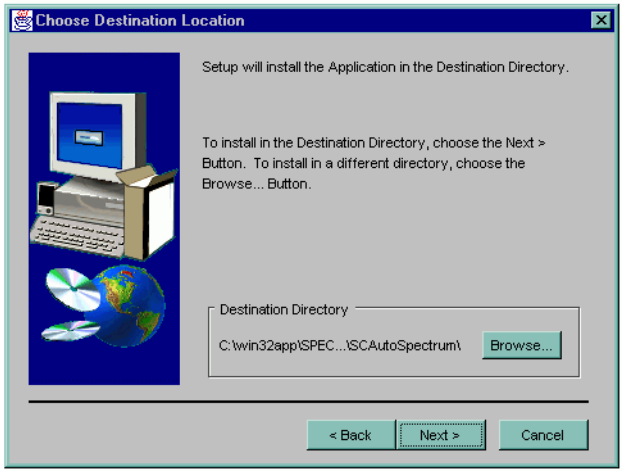


Figure 2-2: Choose Destination Location

5 Select a destination path to install the interface files by:

- Choosing the default destination

  C:\Program Files\Peregrine Systems\SCAutoSpectrum

- Clicking the **Browse** button to install the files in a different location. You can also create a new directory or use an existing path.

6 Click **Next**.

The Host screen appears.

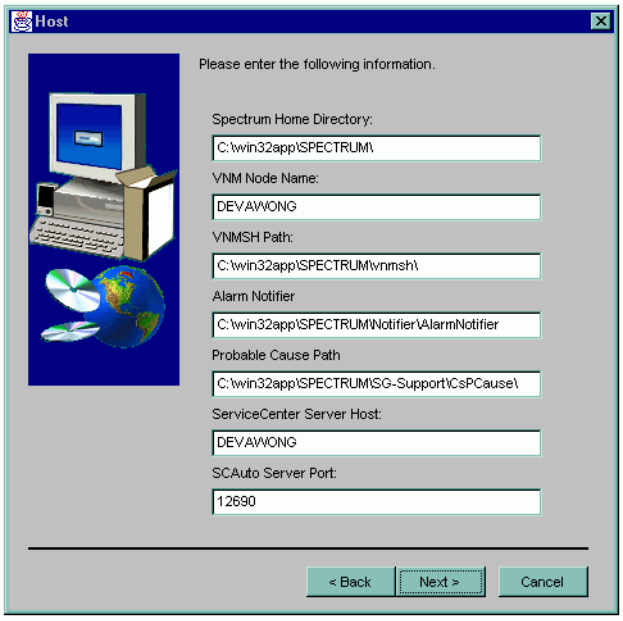


Figure 2-3: Host screen

7 You can accept the defaults or type in new information. Choose the parameters that apply to your installation (see Table 2-1 on page 20 to help you decide).

8 Click **Next**.

The prompt to install cutthru files for launching SC client appears. The system is asking if you want to modify SPECTRUM's **CsStdMenu** file. This adds pull-down menus cut-throughs to access ServiceCenter.

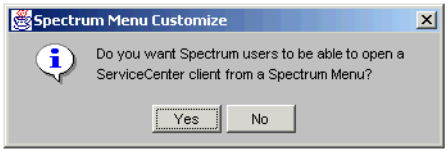
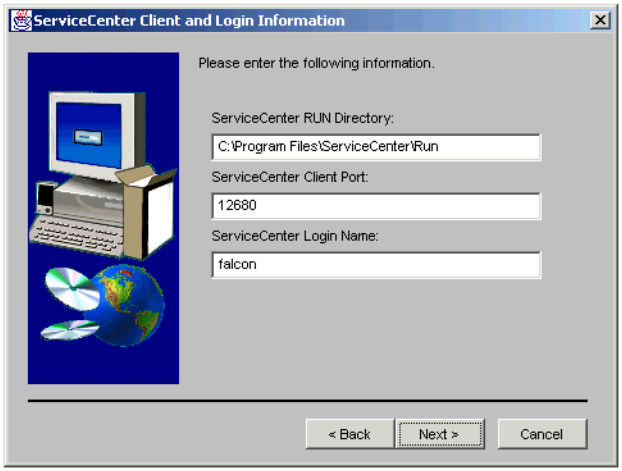


Figure 2-4: Modifying the CsStdMenu files

9 Click **Yes** or **No**.

If you select **Yes**, Cutthru options screen appear. If you select **No**, the Install Options selected screen appears (see Figure 2-5 on page 24).

10 If you select **Yes**, enter the information in the Cutthru screen.



11 Click **Next**.

The Install Options Selected screen appears with the values you entered.

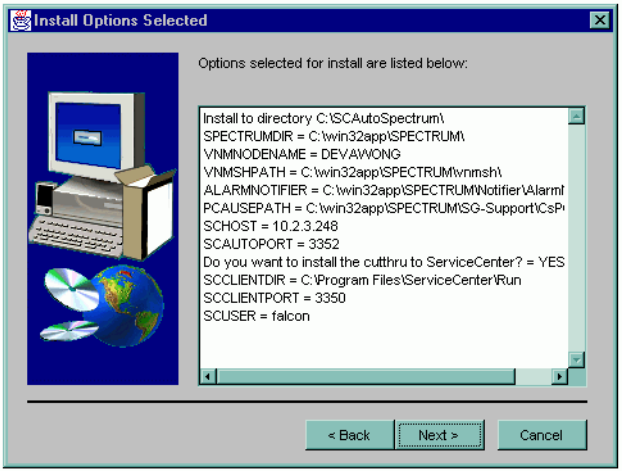


Figure 2-5: Install Options Selected screen

Note: This screen shows installed options without the Cutthru option installed.

12 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 appears, followed by an installation progress dialog. After the files are copied, a post-install Javascript configures the product.

Finally, the Setup Complete screen appears.

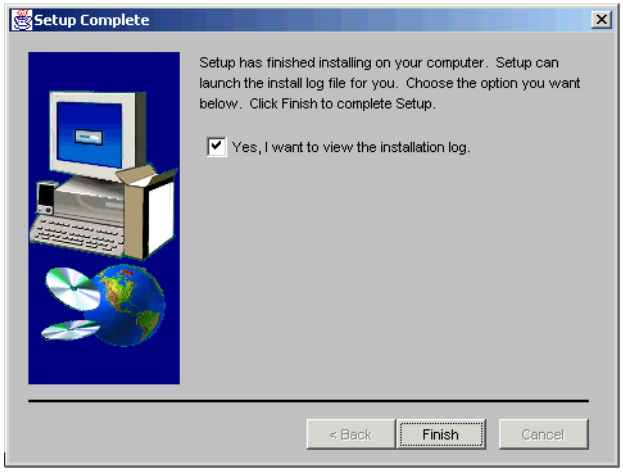


Figure 2-6: Setup Complete screen

13 Indicate whether you want to see the installation log. Peregrine 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 check box checked, the Installation Log File screen appears.

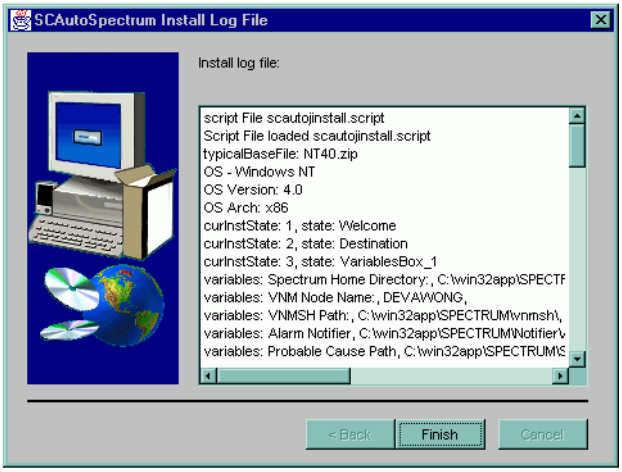


Figure 2-7: Install Log File screen

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

This completes the installation of SCAuto for SPECTRUM.

Installing on Solaris

To install SCAuto for SPECTRUM on Solaris:

1 Login as the superuser *root*.

2 Mount the CD ROM drive onto a partition. For example, use the command:

mount /cdrom

3 Change directory to the CDROM partition and execute the installation script/batch file. The name of the script is *solaris_install.sh*

The system runs until the Welcome screen appears.

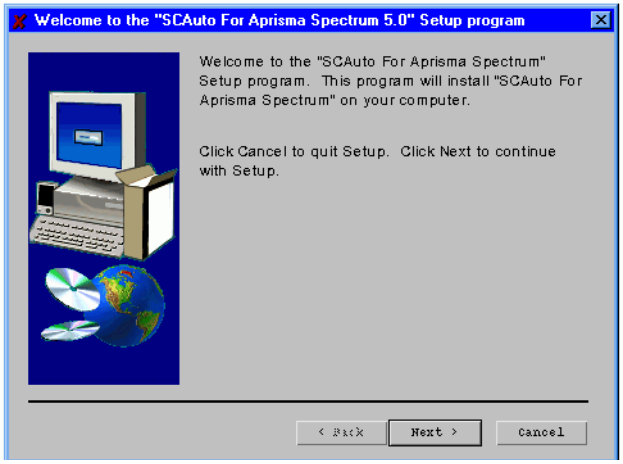


Figure 2-8: Welcome screen

4 Click **Next.**

The Choose Destination Location screen opens.

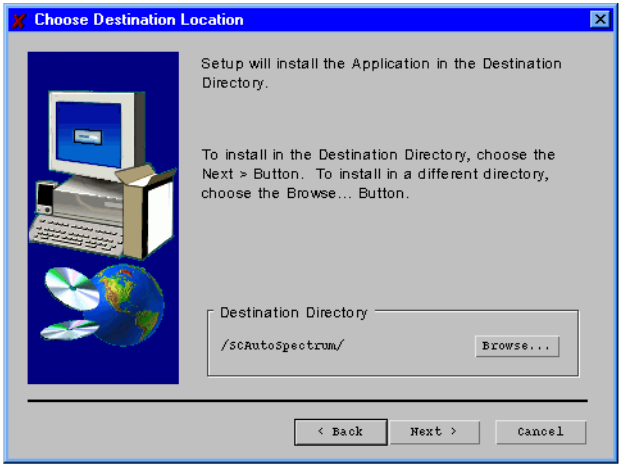


Figure 2-9: Choose Destination Location

5 Select a destination path to install the interface files by:

- Choosing the default destination of

C:\Program Files\Peregrine Systems\SCAutoSpectrum

- Clicking the **Browse** button to install the files in a different location. You can also create a new directory or use an existing path.

6 Click **Next**.

The Host screen opens.

Figure 2-10: Host Information screen

7 You can accept the defaults or type in new information. Choose the parameters that apply to your installation (see Table 2-1 on page 20 to help you decide).

Note: On Solaris 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 prompt to install cutthru files for launching SC client appears. The system is asking if you want to modify SPECTRUM's **CsStdMenu** file. This adds pull-down menu cut-throughs to access ServiceCenter.

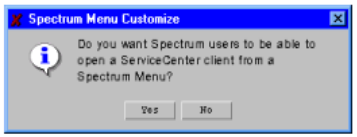


Figure 2-11: Modifying the CsStdMenu files

If you select **Yes**, the ServiceCenter Client and Login Information screen appear. If you select **No**, the Install Options Selected screen appears.

9 If you selected yes, enter the ServiceCenter client and login information.

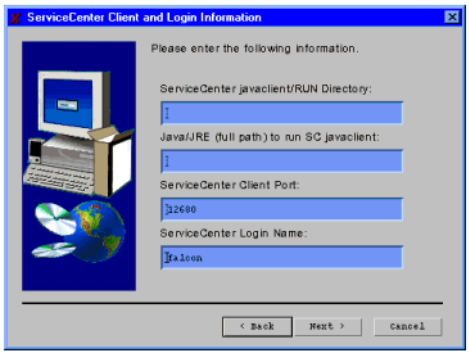


Figure 2-12: ServiceCenter Client and Login Information

10 After you enter the information, click **Next**.

The Install Options Selected screen appears that shows the values you recently entered.

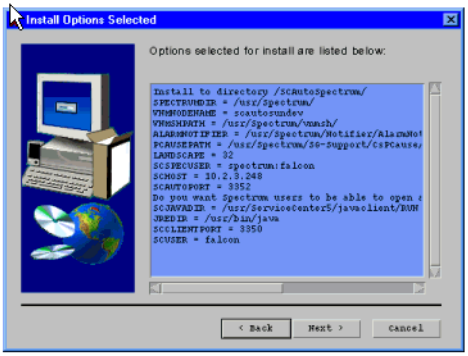


Figure 2-13: Install Options Selected screen

Note: This screen shows installed options without the Cutthru option installed.

11 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 appears, followed by an installation progress dialog. After the files are copied, a post-install Javascript configures the product.

Finally, the Setup Complete screen appears.

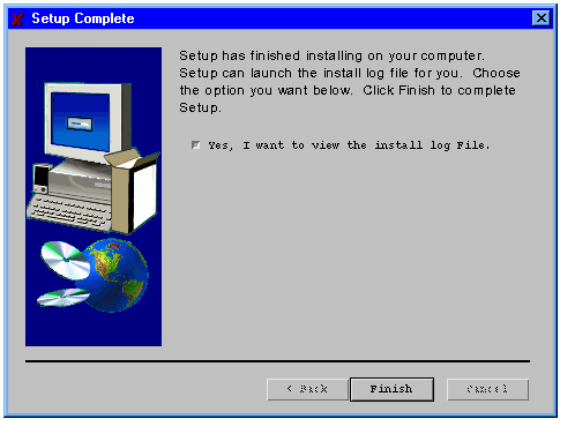


Figure 2-14: Setup Complete screen

12 Indicate whether you want to see the installation log. Peregrine 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 check box checked, the Installation Log File screen appears.

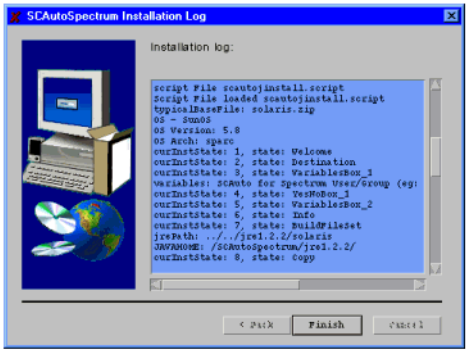


Figure 2-15: Install Log File screen

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

This completes the installation of SCAuto for SPECTRUM.

CHAPTER 3 Configuration

This chapter describes how to configure SCAuto for SPECTRUM. It covers the following topics:

Configuring Using ECMA Scripts

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

There are engineered Java object representations of ServiceCenter event objects and SPECTRUM event objects. Each Java object is furnished with methods to connect, retrieve, and send itself through its connection to ServiceCenter. 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. See Figure 3-1 for a diagram of the overall system flow. ExecuteJS class implements the ECMA interpreter. An example of executing the script:

```
jre -classpath
lib/classes.zip:lib/xml.jar:lib/fesi.jar:/SCAutoJ.jar:lib/SCAutoSpectrumJ.jar:
ExecuteJS writeSpectrumEvent.js recv_alarms.js
```

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

General ECMA techniques used

The general ECMA techniques used are:

- Defining short-cuts for Java classes, variables, and methods.
- Loops

These techniques are described in the sections that follow.

Defining short-cuts for Java Classes, Variables, and Methods

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:

```
SpectrumEvent = Packages.SpectrumEvent;
```

(define the SpectrumEvent variable as the Java class SpectrumEvent - 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 to 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 SpectrumEvent();
        writeln("waiting for trap ...");
        status = spectrumBridge.readEvent(vEvent);
        writeSpectrumEvent(scBridge, vEvent);
    }
```

SPECTRUM Event Object

The SPECTRUM Event Object is the data communication medium for sessions connecting to SPECTRUM. It provides instance methods for accessing and assigning values to the event fields.

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 getEventType(String eventType)		
Definition:	Gets the event type from the SPECTRUM Alarm.		
Arguments:	None		
Returns:	The event type	SET	New alarm
		CLEAR	Closed alarm
		UPDATE	Updated alarm

SPECTRUM SpectrumModel Object

The SpectrumModel Object contains the model handle, model name, type handle, and type name for a SPECTRUM model. For the user, it provides instance methods for accessing values for these fields.

Class Method:	SpectrumModel()	
Definition:	Constructor	
Arguments:	None	
Returns:	New SpectrumModel Object.	

Class Method:	getMHandle()	
Definition:	Gets the model handle from the SpectrumModel Object.	
Arguments:	None	
Returns:	Mhandle	The model handle.

Class Method:	getMName()	
Definition:	Gets the model name from the SpectrumModel Object.	
Arguments:	None	
Returns:	MName	The model name.

Class Method:	getMTypeHnd()	
Definition:	Gets the model type handle from the SpectrumModel Object.	
Arguments:	None	
Returns:	MTypeHnd	The model type handle.

Class Method:	getMTypeName()	
Definition:	Gets the model type name from the SpectrumModel Object.	
Arguments:	None	
Returns:	MTypeName	The model type name.

SPECTRUM SpectrumAttributes Object

The SpectrumAttributes Object is used to create a hashtable of all SpectrumAttribute Objects for a SpectrumModel Object. Specific attributes can be looked up using either the attribute name or id.

Class Method:	SpectrumAttributes()	
Definition:	Constructor. Uses the *show attributes* cli command and creates a new hashtable of SpectrumAttribute Objects that can be parsed to create inventory items for ServiceCenter.	
Arguments:	None	
Returns:	New hashtable of the SpectrumAttribute Objects.	

Class Method:	getSpectrumAttribute(String value)	
Definition:	Gets the attribute for a SpectrumModel Object based on the attribute name or id.	
Arguments:	Value	The attribute name or id.
Returns:	SpectrumAttribute Object	

SPECTRUM SpectrumAttribute Object

The SpectrumAttribute Object contains the attribute id, name, instance id, and value for a Spectrum model. For the user, it provides instance methods for accessing values to these fields.

Class Method:	SpectrumAttribute()	
Definition:	Constructor	
Arguments:	None	
Returns:	New SpectrumModel object.	

Class Method:	getAId()	
Definition:	Gets the attribute id from the SpectrumModel Object.	
Arguments:	None	
Returns:	AId	The attribute id.

Class Method:	getAName()	
Definition:	Gets the attribute name from the SpectrumModel Object.	
Arguments:	None	
Returns:	AName	The attribute name.

Class Method:	getAIid()	
Definition:	Gets the attribute instance id from the SpectrumModel Object.	
Arguments:	None	
Returns:	AIid	The attribute instance id.

Class Method:	getAValue()
Definition:	Gets the attribute value from the SpectrumModel Object.

Arguments:	None	
Returns:	AValue	The attribute value.

SCEvMon Configuration

SCEvMon is configured through scevmon.properties file. There are two fixed key properties that provide the information for SCEvMon to identify a Peregrine Systems's SCAuto Adapter: *scevmon.Key* and *scevmon.SessionID*. All the other properties are configurable and may be set when starting SCEvMon. The properties are:

scevmon.Key	This is the license key that enables SCEvMon to function. **Not Modifiable.**
scevmon.SessionID	This is the session ID used by SCAuto Base Server to identify the incoming SCAuto Adapter. **Not Modifiable.**
scevmon.IniFile	This property contains the name of the SCAuto Adapter's initialization file.
scevmon.SleepInterval	This is a sleep interval in number of seconds for SCEvMon to pause before attempting to read the next event from both the *to* and the *from* queues when there are no events available. If there are events available in any queue, it finishes processing these events before sleeping. Default is 5.
scevmon.To_SC	This property specifies whether to enable incoming event process to ServiceCenter. Default is off. on - enable. off - disable.
scevmon.From_SC	This property specifies whether to enable ServiceCenter outbound event process. Default is off. on - enable. off - disable.

scevmon.EventList	This property specifies a ServiceCenter event type or list of ServiceCenter event types to be queried at the connection time. A value of ***all*** causes queries to fetch all event types. Multiple event types are specified using the format (*type1, type2, ...*). Default is all.
scevmon.UserList	This property specifies a user name or list of user names to be used for querying the ServiceCenter events. The value *all* causes events owned by any user to be retrieved, and multiple user names in the format (*user1,user2,...*) can be used. Default is all.
scevmon.ReconnectSC	Retry to connect to SCAuto Base Server after the server restarted. Default is 0. 0: Do not retry to connect to the SCAuto Base Server. 1: Retry to connect to the SCAuto Base Server.
scevmon.ReconnectSCInterval	Number of seconds to sleep between retrying. Default is 120 (two minutes).
scevmon.NumOfRetryConnect	Number of retries before exiting. 0 = retry forever. Default is 0.
scevmon.InterrupterInterval	Number of seconds to sleep for SCAutoJ's Interrupter object. This object checks the existing status of the SCEvMon's running files. If the running file has been removed, it signals SCEvMon and cause that particular process to stop. Default is 5.
scevmon.LogError	This is an error logging flag to enable error codes to be output to a log file. The name of the log file is configured in the initializing file specified by the *scevmon.IniFile* property. Default is 0. 0 - run log off. 1 - run log on.

scevmon.LogErrString	This is an error logging flag to enable error strings to be output to a log file. The name of the log file is configured in the initializing file specified by the *scevmon.IniFile* property. Default is 0. 0 - run log off. 1 - run log on.
scevmon.debug	A debug flag for SCEvMon. Default is 0. 0 - turn debug off. 1 - level 1 debugging 2 - level 2 debugging
scevmon.debugJNI	Debug flag for SCAutoJ. Default is 0. 0 - turn debug off. 1 - level 1 debugging
scevmon.DeleteSCEvent	This flag enable a ServiceCenter event to be deleted from the event-out queue after this event is successfully written to the SCAuto Adapter's event queue file. Default is 0. 0: do not delete this event. 1: delete a ServiceCenter event from the event-out queue.
scevmon.IgnoreWriteQError	This flag allows the From_SC process continue running even after the write event to queue file operations failed. Default is 0. 0: process stops after a write event to queue file operation failed. 1: ignore the write error and continue the process
scevmon.ShowSCAutoVer	Log a string that describes the current SCAuto version and the date the SCAuto library or DLL was built. It is intended to be used for debugging or for presenting descriptive text upon application start-up. Default is 0.

0: do not log SCAuto Library or DLL version string

1: log SCAuto Library or DLL version string

Customizing Event Integration

SCAuto for SPECTRUM can be customized for startup/shutdown behavior and data mapping between ServiceCenter and SPECTRUM. These customizations can be done by modifying certain initializing files, Java property files, as well as ECMA scripts (JavaScript). The tables in 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 SPECTRUM. The queue manager continues to cache all events when one of the connecting software is down. It acts as an event *pump* that accumulates events during a system down and continues pumping events when the system is up again.

File Name:	<inst. dir.>/bin/StartSCEvmon.sh
Description:	This is the script file specified to start the Interface Queue Manager. Its main purpose is to set up the correct library paths (UNIX) or DLL paths (Windows NT) to enable the Java Runtime Environment to execute.
Customization:	Peregrine suggests that customers not modify this file.

File Name:	<inst. dir.>/bin/StopSCEvMon.sh
Description:	This is the script file that is specified to stop the Interface Queue Manager.
Customization:	Peregrine suggests that customers not modify this file.

File Name: <inst. dir.>/SCAutoSpectrum.ini

Description: This initialization file contains traditional SCAutomate parameters that are used by the SCAutomate SDK v3 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: no default
scevent.server	The SCAutomate 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.
debug	A debug flag for SCAutomate. 1 - turn debug on. 0 - turn debug off. Default: 0
debugscautoevents	A debug level flag for SCAuto events. 0 - off. 1 - level 1. 2 - level 2. Default: 0

Java properties file (scautoj.properties)

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

Description: This is the Java property file 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

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. Default: SCAutoJ.JNIBridge
scautoj.SCMessagingClassFile	This is the absolute path name to the class file that contains the ServiceCenter Messaging class. Not modifiable by user. Default: JNIBridge.class
scautoj.VendorMessagingClassName	This is the java class name including the package name of the class that defines the SPECTRUM Messaging class. This should be provided by the installer and not modified after installation. Default: SpectrumBridge
scautoj.VendorMessagingClassFile	This is the absolute path name to the class file that contains the ServiceCenter Messaging class. Not modifiable by user. Default: SpectrumBridge.class
scautospectrumj.vnmshpath	This is the path to SPECTRUM's CLI directory. Used for inventory integration only.
scautospectrumj.pcausepath	This is the path to SPECTRUM's Probable Cause directory.
scautospectrumj.alarmnotifier	This is the path to SPECTRUM's AlarmNotifier executable.
scautospectrumj.alarmnotifierparms	This is the path to the parameters file used by the SPECTRUM's AlarmNotifier executable started by the SCAutoAlarms process. Default: AlarmNotifierParms
scautospectrumj.modeltypes	Lists the model types to inventory (Host_Sun, Host_NT, Host_Compaq, GnSNMPDev, Pingable). Used for inventory integration only.

scautospectrumj.alarmTraceLevel
scautospectrumj.inventoryTraceLevel

Customizing the SPECTRUM Alarm Monitor

The SPECTRUM Alarm Monitor process has the responsibility of receiving Alarms from SPECTRUM and logging them to the queue file *scevents.to.<sc host>.<scauto port>*. The Interface Queue Manager process (if running) picks up each alarm and forwards it to ServiceCenter.

The following describes the files and their values related to customizing this process.

File Name: <inst. dir.>/bin/StartSCAutoAlarms.sh

Description: This is the shell script executed when you choose the Start SCAutoAlarms menu option under the SCAutomate sub-menu in SpectroGraph. Its main purpose is to set up correct library paths (UNIX) or DLL paths (Windows NT) to enable the Java Runtime Environment to execute.

.

File Name: <inst. dir.>/bin/StopSCAutoAlarms.sh

Description: This is the shell script executed when you choose the Stop SCAutoAlarms menu option under the SCAutomate sub-menu in SpectroGraph. It stops SCAutoAlarms.

File Name: <inst. dir.>/bin/alarms.js

Description:

The SPECTRUM Alarm Monitor process can also use the scautoj.properties file (see *Java properties file (scautoj.properties)* on page 43) for customization.

alarms.js

The following shows the contents of the alarm.js file.

```
// alarms.js
File = Packages.java.io.File;
String = Packages.java.lang.String;
```

```
Util = Packages.JSInterpreter.Util;

scEvent = Packages.SpectrumAlarms.scEvent;
vEvent = Packages.SpectrumAlarms.vEvent;
pcausepath =  Packages.SpectrumAlarms.pcausepath;
SpecUtil =  Packages.SpectrumUtil;

defaultcategory = "network";

//get the fields from the alarm

alarmdate       = vEvent.getEvField("DATE");
alarmtime       = vEvent.getEvField("TIME");
modeltype       = vEvent.getEvField("MTYPE");
modelname       = vEvent.getEvField("MNAME");
alarmid         = vEvent.getEvField("AID");
causecode       = vEvent.getEvField("CAUSE");
condition       = vEvent.getEvField("COND");
repairperson    = vEvent.getEvField("REPAIRPERSON");
alarmstatus     = vEvent.getEvField("STATUS");
spectroserver   = vEvent.getEvField("SERVER");
landscape       = vEvent.getEvField("LANDSCAPE");
modelhandle     = vEvent.getEvField("MHANDLE");
modeltypehandle = vEvent.getEvField("MTHANDLE");
ipaddress       = vEvent.getEvField("IPADDRESS");
securitystring  = vEvent.getEvField("SECSTR");
alarmstate      = vEvent.getEvField("ALARMSTATE");
acknowledged    = vEvent.getEvField("ACKD");
userclearable   = vEvent.getEvField("CLEARABLE");

// you only get the following if you have SANM.....

if(vEvent.getEvField("SANM") == "Y" || vEvent.getEvField("SANM") == "enabled")
{
  SANMEnabled     = true;
  flashgreen      = vEvent.getEvField("FLASHGREEN");
  probablecause   = vEvent.getEvField("PCAUSE");
  location        = vEvent.getEvField("LOCATION");
  alarmage        = vEvent.getEvField("AGE");
  notificationdata= vEvent.getEvField("NOTIFDATA");
  eventmessage    = vEvent.getEvField("EVENTMSG");
}
else
{
  SANMEnabled     = false;
}

//check to see if we have already processed this alarm

if (vEvent.getEventType()=="SET" && alarmstate=="EXISTING")
{
  SpecUtil.logprintA(0,"Alarm "+alarmid+" already processed");
  scEvent.setEventStatus("filtered");
  exit;
}


//check to see if we want to process this alarm

scProcessAlarm=true;
```

```
if(
    (condition == "RED"    || condition == "CRITICAL") //process RED alarms
 || (condition == "ORANGE" || condition == "MAJOR")    //process ORANGE alarms
//|| (condition == "YELLOW" || condition == "MINOR")    //uncomment to process YELLOW alarms
 )
 scProcessAlarm=scProcessAlarm && true;
else
 scProcessAlarm=false;

if(
    (modeltype == "Host_Compaq") //process Host_Compaq modeltypes
 || (modeltype == "Host_Sun")    //process Host_Sun modeltypes
 || (modeltype == "Host_NT")     //process Host_NT modeltypes
 || (modeltype == "Host_Device") //process Host_Device modeltypes
 || (modeltype == "GnSNMPDev")   //process GnSNMPDev modeltypes
//|| (modeltype == "Pingable")    //uncomment to process Pingable modeltypes
 )
 scProcessAlarm=scProcessAlarm && true;
else
 scProcessAlarm=false;

if (!scProcessAlarm)
{
 scEvent.setEventStatus("filtered");
}
else
{
// map the alarmdata to a servicecenter event
// SET = pmo
// CLEAR = pmc
// UPDATE = pmu

 if(vEvent.getEventType()=="SET")
 {
  scEvent.setEventType("pmo");
  scEvent.createEventFieldNamesFromMapFile("EventMap"+ File.separator +"To_SC"+ File.separator
+"pmo.map");
  scEvent.setEvField("logical.name", modelname);
  scEvent.setEvField("network.name", modelname);
  scEvent.setEvField("reference.no", alarmid);
  scEvent.setEvField("assignee.name", repairperson);
  scEvent.setEvField("cause.code", causecode);
  scEvent.setEvField("network.address", ipaddress);
  scEvent.setEvField("category", defaultcategory);
  scEvent.setEvField("model", modeltype);

// SANM will provide the probable cause data and the event message data
  if(SANMEnabled)
  {
   action = "Local Alarm Time: " + alarmdate + " " + alarmtime + "||" +
            "PCAUSE"   + "|" + probablecause + "|" +
            "EVENTMSG" + "|" + eventmessage  + "|";
   scEvent.setEvField("_ax.field.name", action);
  }
// Without SANM, build the probable cause filename from the causecode value
// fill in leading zeroes between "Prob" and the causecode value
  else
  {
   while (causecode.length < 8)
```

```
    {
      causecode = "0" + causecode;
    }

    pcausefile = pcausepath + File.separator + "Prob" + causecode;

    util = new Util();
    tmpdata = new String(util.run("cat " + pcausefile));
    pcausedata = tmpdata.replace('\n', '|');
    action = "Local Alarm Time: " + alarmdate + " " + alarmtime + "||" + pcausedata + "|";
    scEvent.setEvField("_ax.field.name", action);
   }

// set the priority.code value based on the color/severity in the alarm

   if(condition == "RED" || condition == "CRITICAL")
    scEvent.setEvField("priority.code", "1");
   else if(condition == "ORANGE" || condition == "MAJOR")
    scEvent.setEvField("priority.code", "2");
   else if(condition == "YELLOW" || condition == "MINOR")
    scEvent.setEvField("priority.code", "3");
   else
    scEvent.setEvField("priority.code", "4");
  }
  else if(vEvent.getEventType()=="CLEAR")
  {
    scEvent.setEventType("pmc");
   scEvent.createEventFieldNamesFromMapFile("EventMap"+ File.separator +"To_SC"+ File.separator +"pmc.map");
    scEvent.setEvField("logical.name", modelname);
    scEvent.setEvField("network.name", modelname);
    scEvent.setEvField("reference.no", alarmid);
    action = "Local Alarm Cleared Time: " + alarmdate + " " + alarmtime;
    scEvent.setEvField("resolution", action);
  }
  else if(vEvent.getEventType()=="UPDATE")
  {
    scEvent.setEventType("pmu");
    scEvent.createEventFieldNamesFromMapFile("EventMap"+ File.separator +"To_SC"+ File.separator
+"pmu.map");
    scEvent.setEvField("logical.name", modelname);
    scEvent.setEvField("network.name", modelname);
    scEvent.setEvField("reference.no", alarmid);
    scEvent.setEvField("assignee.name", repairperson);
    action = "Local Alarm Updated Time: " + alarmdate + " " + alarmtime +
           "|Alarm Status : " + alarmstatus +
           "|Acknowledged : " + acknowledged;
    scEvent.setEvField("update.action", action);
  }
}
```

Customizing Inventory Integration

Inventory integration gathers all the models specified in the modeltype parameter from the SPECTRUM database. The following files are involved in inventory gathering:

File Name: <inst. dir.>/bin/StartSCAutoInventory.sh

Description: This is the shell script executed when you choose the Start SCAutoInventory menu option under the SCAutomate sub-menu in SpectroGraph. Its main purpose is to set up the correct library paths (UNIX) or DLL paths (Windows NT) to enable the Java Runtime Environment to execute, set up a unique CLI sessid id, and connect to the SPECTRUM CLI.

File Name: <inst. dir.>/bin/StopSCAutoInventory.sh

Description: This is the shell script executed when you choose the Stop SCAutoInventory menu option under the SCAutomate sub-menu in SpectroGraph. Its main purpose is to stop ScAutoInventory.

File Name: <inst. dir.>/bin/model_name.js

Description:

Inventory integration can also be customized using the **scautoj.properties** file.

Example of map for Host_Compaq model.

```
scEvent.setEvField("type","server");
scEvent.setEvField("logical.name",modelname);
scEvent.setEvField("location",ssmodelobject.getSpectrumAttribute("Location").getAValue());
scEvent.setEvField("network.address",ssmodelobject.getSpectrumAttribute("Network_Address").getAValue());
scEvent.setEvField("network.name",modelname);
scEvent.setEvField("vendor",ssmodelobject.getSpectrumAttribute("Manufacturer").getAValue());
scEvent.setEvField("model",ssmodelobject.getSpectrumAttribute("Model_Number").getAValue());
scEvent.setEvField("serial.no.",ssmodelobject.getSpectrumAttribute("Serial_Number").getAValue());
scEvent.setEvField("contact.name",ssmodelobject.getSpectrumAttribute("ContactPerson").getAValue());
scEvent.setEvField("subtype",ssmodelobject.getSpectrumAttribute("DeviceType").getAValue());
scEvent.setEvField("subnet.mask",ssmodelobject.getSpectrumAttribute("Network_Mask").getAValue());
scEvent.setEvField("protocol.addr",ssmodelobject.getSpectrumAttribute("Network_Address").getAValue());
scEvent.setEvField("mac.address",ssmodelobject.getSpectrumAttribute("MAC_Address").getAValue());
scEvent.setEvField("description",ssmodelobject.getSpectrumAttribute("0x10052").getAValue());
scEvent.setEvField("model",ssmodelobject.getSpectrumAttribute("cpqSiProductName").getAValue());
scEvent.setEvField("operating.system",ssmodelobject.getSpectrumAttribute("cpqHoName").getAValue());
scEvent.setEvField("os.version",ssmodelobject.getSpectrumAttribute("cpqHoVersion").getAValue());
comments  = "Model Handle: "     + modelhandle + "|";
comments += "Model Name: "       + modelname   + "|";
comments += "Model Type Name: "  + modeltype   + "|";
comments += "Model Type Handle: " + modeltypehandle;
scEvent.setEvField("comments",comments);
```

4 SCAuto SPECTRUM Files

CHAPTER

This chapter lists the files in SCAuto for SPECTRUM.

SCAutoSpectrum

SCAutoSpectrum includes the following files

-rw-rw-rw-	1 spectrum root	264 Apr 7 02:47	AlarmNotifierParms
-rw-rw-rw-	1 spectrum root	1937 Apr 7 02:47	CsStdMenu
-rw-rw-rw-	1 spectrum root	558 Apr 7 02:47	CsStdMenu.no.cutthru
drwxrwxrwx	4 spectrum root	512 Apr 7 02:46	EventMap
-rw-rw-rw-	1 spectrum root	81 Apr 7 02:47	SpectrumJ.ini
drwxrwxrwx	2 spectrum root	512 Apr 7 02:46	SpectrumScripts
drwxrwxrwx	2 spectrum root	512 Apr 7 02:47	UnInst
drwxrwxrwx	2 spectrum root	512 Apr 7 02:47	bin
drwxrwxrwx	4 spectrum root	512 Apr 7 02:47	jre1.2.2
drwxrwxrwx	2 spectrum root	512 Apr 7 02:46	lib
-rwxrwxrwx	1 spectrum root	338 Apr 7 02:47	modFiles.sh
-rw-rw-rw-	1 spectrum root	557 Apr 7 02:47	scautoj.properties
-rw-rw-rw-	1 spectrum root	372 Apr 7 02:46	scevmon.properties

SCAutoSpectrum/EventMap:

```
drwxrwxrwx    2 spectrum root     512 Apr  7 02:46
drwxrwxrwx    2 spectrum root     512 Apr  7 02:46       To_SC
```

SCAutoSpectrum/EventMap/To_SC:

```
-rwxrwxrwx    1 spectrum root    1625 Apr  7 02:46       GnSNMPDev.js
-rwxrwxrwx    1 spectrum root    1914 Apr  7 02:46       Host_Compaq.js
-rwxrwxrwx    1 spectrum root    1616 Apr  7 02:46       Host_NT.js
-rwxrwxrwx    1 spectrum root    1617 Apr  7 02:46       Host_Sun.js
-rwxrwxrwx    1 spectrum root     540 Apr  7 02:46       ICMdevicenode.map
-rwxrwxrwx    1 spectrum root     447 Apr  7 02:46       ICMrouter.map
-rwxrwxrwx    1 spectrum root     822 Apr  7 02:46       ICMserver.map
-rwxrwxrwx    1 spectrum root     759 Apr  7 02:46       ICMworkstation.map
-rwxrwxrwx    1 spectrum root    1149 Apr  7 02:46       Pingable.js
-rwxrwxrwx    1 spectrum root     193 Apr  7 02:46       alarm.map
-rwxrwxrwx    1 spectrum root    5593 Apr  7 02:46       alarms.js
-rwxrwxrwx    1 spectrum root     266 Apr  7 02:46       pmc.map
-rwxrwxrwx    1 spectrum root     256 Apr  7 02:46       pmo.map
-rwxrwxrwx    1 spectrum root     270 Apr  7 02:46       pmu.map
```

SCAutoSpectrum/SpectrumScripts:

The following are the SCAutoSpectrum scripts.

```
-rwxrwxrwx    1 spectrum root     158 Apr  7 02:46       pmcscript
-rwxrwxrwx    1 spectrum root     156 Apr  7 02:46       pmoscript
-rwxrwxrwx    1 spectrum root     159 Apr  7 02:46       pmuscript
```

SCAutoSpectrum/bin:

-rwxrwxrwx	1 spectrum root	501 Apr 7 02:47	StartSCAutoAlarms.sh
-rwxrwxrwx	1 spectrum root	945 Apr 7 02:47	StartSCAutoInventory.sh
-rwxrwxrwx	1 spectrum root	510 Apr 7 02:47	StartSCEvmon.sh
-rwxrwxrwx	1 spectrum root	69 Apr 7 02:47	StopSCAutoAlarms.sh
-rwxrwxrwx	1 spectrum root	72 Apr 7 02:47	StopSCAutoInventory.sh
-rwxrwxrwx	1 spectrum root	67 Apr 7 02:47	StopSCEvmon.sh
-rwxrwxrwx	1 spectrum root	3023 Apr 7 03:05	scelogin.sh

SCAutoSpectrum/jrel.2.2:

drwxrwxrwx	3 spectrum root	512 Apr 7 02:47	bin
drwxrwxrwx	9 spectrum root	2048 Apr 7 02:47	lib

SCAutoSpectrum/lib:

-rwxrwxrwx	1 spectrum root	42637 Apr 7 02:46	SCAutoJ.jar
-rwxrwxrwx	1 spectrum root	28586 Apr 7 02:46	SCAutoSpectrumJ.jar
-rwxrwxrwx	1 spectrum root	21543 Apr 7 02:46	SCEvMonJ.jar
-rwxrwxrwx	1 spectrum root	1358806 Apr 7 02:46	fesi.jar
-rwxrwxrwx	1 spectrum root	417457 Apr 7 02:46	libSCAutoJNIBridge.so
-rwxrwxrwx	1 spectrum root	301396 Apr 7 02:46	xml.jar

Index

X


Peregrine
SYSTEMS