Peregrine

ServiceCenterPlus for Tivoli TEC User's Guide

Version 1.2.a

PART NO: SCA-1.2.a-ENG-01002-00140

Peregrine SYSTEMS

Peregrine Systems, Inc.
Worldwide Corporate Campus and Executive Briefing Center
3611 Valley Centre Drive San Diego, CA 92130
Tel 800.638.5231 or 858.481.5000
Fax 858.481.1751
www.peregrine.com

Table of Contents

Preface **5**
Product Overview 6
Prerequisite Knowledge 7
Contacting Peregrine Systems 8
North and South America 8
United Kingdom regional office 9
France regional office 9
Germany regional office 9
Nordic regional office 10
Benelux regional office. 10
Asia-Pacific regional offices. 11

Chapter 1 Introduction **13**
Product Summary. 14
TEC Component 16
Sample automated problem ticketing scenario. 18
TME 10 Inventory Component 19

Chapter 2 Installation **21**
Installation Notes 22
SCPlus Module De-Installation 23
On UNIX systems. 23
On Windows NT 24
SCPlus Module Installation 25

Tivoli Administrator Setup 31
SCPlus Secondary Installation and Configuration 36
TEC Event Server Customization 41

Chapter 3 TEC Configuration 45
TEC Defined 46
Event server 46
Event classes 47
Rules 49
Configuring TEC for Your Environment 51
Overview 51
Detailed configuration steps 51
Event console 60
Developing Custom Rules 61
ServiceCenter rule summary 62

Chapter 4 ServiceCenter Configuration 65
ServiceCenter Event Services 66
Event registration 68
Event maps 70
Events 72
Event examples 72
Tivoli-generated/SC managed. 72
SC-generated/Tivoli managed. 74
ServiceCenter and Tivoli co-manage 76
Tivoli to ServiceCenter Mapping 80
Event mapping 80
Eventin maps 81
Eventout maps 83
Event Filtering 86
evfilter.sh 86
ServiceCenter Problem Management 88

Chapter 5 Inventory Integration 91
Acquiring Inventory Data from Tivoli 92
Mapping Data 93
TivoliQuery object 93
SCEvent object 96
SCDiscover object 97
Sending Data to ServiceCenter 98
System Architecture 99
System Diagram 102
Installation 103
Operation 104
System Configuration 105
Customizing Tivoli Tables, Views, and Queries 106
Customizing ECMA Scripts 107
Declaring packages that will be used later 107
Creating TivoliQuery objects 107
Loops 108
Creating SCEvent objects 108
Getting data out of TivoliQuery objects and into SCEvent objects 109
Writing the SCEvent object out to the queue file 109
Customizing Static Event Maps 110

Chapter 6 Operation 111
ServiceCenter Configuration 113
Define target ServiceCenter server 113
Start ServiceCenter server 115
Shutdown ServiceCenter server 116
ServiceCenter server locks 117
ServiceCenter server shared memory 118
Add ServiceCenter operator 119
Update ServiceCenter operator 121
Delete ServiceCenter operator 123
ServiceCenter license report 124
ServiceCenter server semaphores 125
ServiceCenter system bulletin 126

ServiceCenter publish/subscribe Information 127
ServiceCenter server status 128
Starting and Stopping a ServiceCenterPlus for Tivoli Session 129
Launching a ServiceCenter client 130
ServiceCenter inventory nodes 132
ServiceCenter processed event archive 133

Chapter 7 Customization 135
Customization 136
Modifying TEC rules 137
Event maps 139
Modifying SC Event Services 139
Sample Scenarios 140
Sample TEC-created and TEC-managed problem scenario 140
Sample ServiceCenter-created and TEC-managed problem scenario . . . 140
Sample ServiceCenter-created and ServiceCenter-managed problem scenario . 141
Monitoring ServiceCenter 143

Chapter 8 Troubleshooting 145

Index 149

Preface

This preface covers the following topics:

- Product overview
- Prerequisite knowledge
- Contacting Peregrine Systems

Product Overview

Welcome to Peregrine Systems' *ServiceCenterPlus for Tivoli TEC*. This product allows you to automate the process of creating, updating, and closing trouble tickets in ServiceCenter, based on Tivoli Enterprise Console (TEC) events. It also supports propagation of TME Inventory data into ServiceCenter's Inventory and Configuration Management (ICM) database.

The ServiceCenterPlus for Tivoli product is part of the suite of SCAutomate (SCAuto) interface products that integrate ServiceCenter with premier network and systems management tools. This interface is based on event transactions sent over a TCP connection to the ServiceCenter server.

This guide provides instruction on the Tivoli Plus module ServiceCenterPlus for Tivoli. This Tivoli Plus module integrates the Tivoli Management Environment (TME) with Peregrine's ServiceCenter product.

This product may also be referred to by the synonymous name of *SCAuto for Tivoli* both in this document and in additional Peregrine Systems literature. Additional information about SCAutomate can be found in the *SCAuto Applications for Windows NT and UNIX* guide.

Prerequisite Knowledge

This guide assumes you have:

- Familiarity with the Tivoli enterprise management system, including TME 10 inventory and TEC configuration.
- Working knowledge of ServiceCenter applications, ServiceCenter Client/Server, and Tivoli. While some procedures for these applications are explained, others are referenced. Refer to the appropriate 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 ServiceCenterPlus for Tivoli module will be installed and implemented, as well as a basic understanding of ServiceCenter applications and Event Services.

Contacting Peregrine Systems

Contact one of the Peregrine Systems Customer Support offices listed here if you have questions about, or problems with, ServiceCenter or SCAuto systems.

For more information about Customer Support, check the support web site: **http://support.peregrine.com** Please contact Customer Support for an account on this site.

Note: Only the European Customer Support staff is multilingual and can provide technical support to customers in their native language.

North and South America

To get help immediately, call Peregrine Customer Support at:

(1) (800) 960-9998 (North America only)

(1) (858) 794-7428 (North and South America)

For ServiceCenter questions or information, send a fax or e-mail to:

Fax: (1) (858) 794-6028

E-mail: **support@peregrine.com**

Send materials that Peregrine Systems Customer Support requests to:

Peregrine Systems, Inc.

ATTN: Customer Support

12670 High Bluff Drive

San Diego, CA 92130

Note: Countries outside North and South America are covered by regional offices. Customers should contact the regional office under which their country is listed.

United Kingdom regional office

Great Britain, Greece, and South Africa

Peregrine Systems Ltd.
1st Floor
Ambassador House
Paradise Road
Richmond, Surrey, Great Britain, TW9 1SQ
Phone: 0800 834770 (toll free) or 0181 334 5844
E-mail: **uksupport@peregrine.com**

France regional office

France, Spain, Italy, Greece, and Africa (except South Africa)

Peregrine Systems
Tour Franklin-La Défense 8
92042 Paris La Défense Cedex, France
Phone: +33 (0) (800) 505 100 (International Toll Free)
E-mail: **frsupport@peregrine.fr**

Germany regional office

Germany and Eastern Europe

Peregrine Systems GmbH
Bürohaus Atricom
Lyoner Strasse 15,
60528 Frankfurt, Germany
Phone: 0049 6966 80260 or 0800-2773823 (toll free in Germany)
E-mail: **fweidman@peregrine.com**

Nordic regional office

Denmark, Norway, Sweden, Finland, and Iceland

Peregrine Systems A/S
Naverland 2, 12 SAL
DK-2600 Glostrup
Denmark

Denmark	Phone: (+45) 80307676
Sweden	Phone: (+45) 77317776
Norway, Iceland, and Finland	E-mail: **nordic@peregrine.com**

Benelux regional office

Netherlands, Belgium, and Luxembourg

Peregrine Systems BV
Botnische Golf 9a
3446 CN Woerden
Netherlands

Netherlands	Phone: 0800 0230 889 (toll free in the Netherlands)
Belgium and Luxembourg	Phone: 00800 7474 7575(toll free in Belgium and Luxembourg)
	E-mail: **benelux.support@peregrine.com**

Asia-Pacific regional offices

Australia, Hawaii, Hong Kong, Japan, Korea, Malaysia, New Zealand, Singapore

Australia	Phone: (800) 146-849
Hawaii	Phone: (1) (800) 960-9998
Hong Kong	Phone: (800) 908056
Japan	Phone: (0044) 221-22795
Singapore	Phone: (800) 1300-949 or -948 E-mail: **apsupport@peregrine.com**

1 Introduction

CHAPTER

This chapter introduces the ServiceCenterPlus (SCPlus) for Tivoli TEC module. This module adds ServiceCenter Problem Management and Inventory Management functions to the Tivoli enterprise management system, by interfacing TEC event services and TME 10 Inventory with ServiceCenter.

The following topics are coverd:

- Product summary
- TEC component
- TME 10 inventory component

Product Summary

The SCAuto product line is designed to allow ServiceCenter integration with solution partners, providing *turn key* implementations easily adapted to various environment. The SCPlus for Tivoli TEC module enables ServiceCenter applications to utilize the Tivoli Enterprise Management System features.

SCPlus for Tivoli TEC utilizes two primary components, one for event monitoring and routing with the Tivoli TEC, and the other for ServiceCenter Inventory management (TME 10 Inventory). Both components are interfaced with ServiceCenter by an open Event Services facility. This facility provides external event definition services to drive ServiceCenter applications from TEC events, and it allows ServiceCenter applications to create events in the TEC.

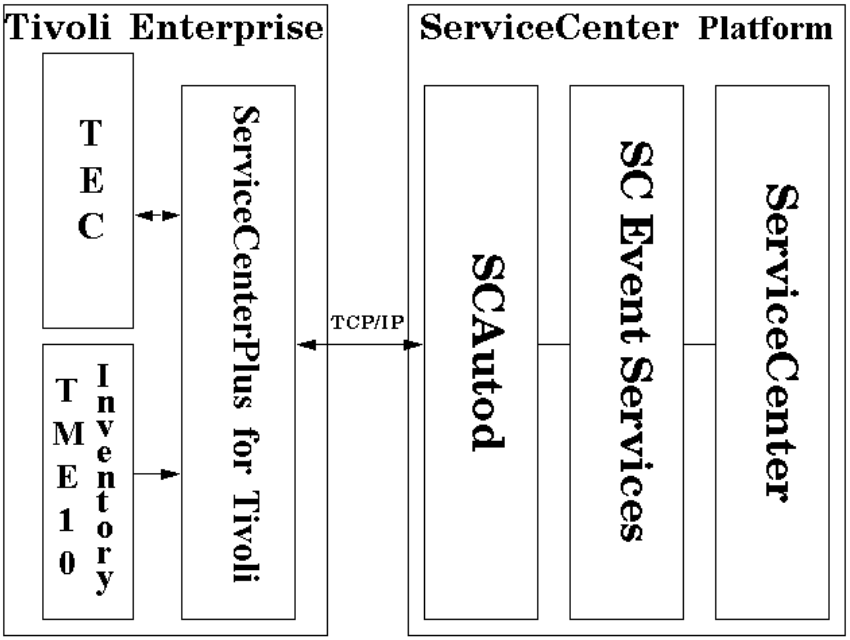


Figure 1: SCPlus for Tivoli flow diagram

Using this facility, TEC events are forwarded and mapped into ServiceCenter events, which drive ServiceCenter applications. ServiceCenter events can also be translated into Tivoli TEC events to drive TEC tasks and external actions.

All SCAuto products connect to ServiceCenter through the use of TCP/IP protocol and a companion SCAuto Base server daemon (*scautod*) on the ServiceCenter platform. The ServiceCenter server need not be active for SCAuto functions. If the ServiceCenter server is down, all TEC-generated events destined for ServiceCenter are held in a sequential file until ServiceCenter is once again available. This provides a completely asynchronous interface, which simplifies operations and maintenance.

Included with this product are SCAuto tasks, ServiceCenter tasks, initial *baroc* files, initial rule files, and required shell scripts to perform various Problem Management scenarios. Some normal Tivoli administration and tailoring will be required to adapt the SCPlus module to your environment (see Chapter 2).

TEC Component

The TEC Component of the SCPlus for Tivoli TEC module converts any Tivoli TEC event into a ServiceCenter event, and also forwards ServiceCenter events to the TEC from ServiceCenter.

The TEC *baroc* file delivered with SCPlus for Tivoli defines several classes of events, including *SCpmOpen*, *SCpmUpdate*, and *SCpmClose*.

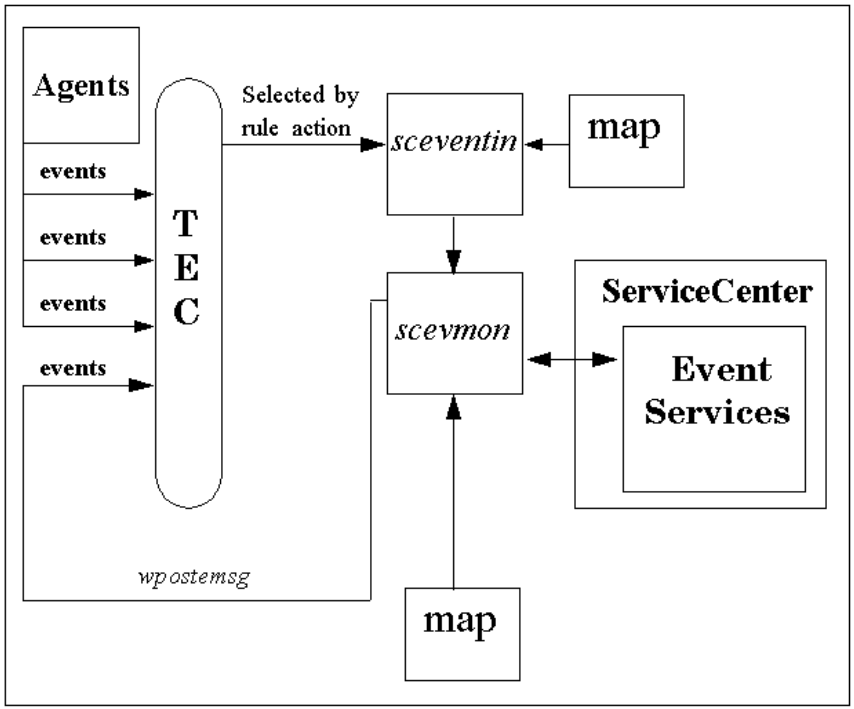


Figure 2: TEC-ServiceCenter event generation flow diagram

ServiceCenter trouble tickets may be generated from arbitrary TEC events by coding rules for events that cause the generation of an SCpmOpen event. The SCpmOpen event functions as a request to open a problem ticket in ServiceCenter.

Following successful generation of the SCpmOpen event, ServiceCenter will generate an SCpmOpen event to indicate completion of the request. The SCpmOpened event contains the problem ticket number issued by ServiceCenter.

The SCpmUpdate event is used to cause an update to a ServiceCenter problem ticket, and the *SCpmUpdated* event is generated by ServiceCenter in response. These extra events can easily be closed by customizing the provided rules, if you do not want to see them in the TEC Event Console window.

An important slot in all these events is called *referenceNo*. This STRING value contains the Message ID of the original event the problem relates to, and is used to subsequently find and update or close a group of related TEC events. This slot defaults to the TEC message ID of the SCpmOpen event itself, unless set to the message id of another event.

ServiceCenter events such as SCpmOpen are sent to ServiceCenter via a background process called *scevmon* (ServiceCenter Events Monitor). The events are constructed by an executable called *sceventin*, which takes TEC slot values from environment variables and maps these into ServiceCenter event slots. Once built, the event record is written to a file called *scevents*. The ServiceCenter Events Monitor *scevmon* reads event records from the *scevents* file and sends them to ServiceCenter.

The ServiceCenter Events Monitor is started and stopped from the SCPlus Application Launch menu. Once started, the monitor connects to ServiceCenter, forwarding all queued events to SC Events Services.

Once the event is in ServiceCenter, the appropriate RAD application is dispatched to process the event.

ServiceCenter events may also be obtained from ServiceCenter and mapped to TEC events. A second set of maps and a mapping facility are provided to map ServiceCenter events to TEC event slots. Once the slot information is complete, scevmon uses the wpostemst command to create an event in the TEC.

Sample automated problem ticketing scenario

A sample automated problem ticketing scenario is as follows:

1. An important event (such as a node down) is received in the TEC.
2. You have customized the current TEC rule base to include a rule that fires for this event, and made this event generate an SCpmOpen event. You set the SCpmOpen event's *referenceNo* slot to the value of the TEC message ID of the original node down or other event, so that the generated SCpmOpen event *points* to that event.
3. The SCpmOpen event is created, and the Peregrine Systems-supplied rules fire, causing an event record to be generated and sent to ServiceCenter.
4. Shortly thereafter, an SCpmOpened event appears in the TEC, with a confirmation message such as: *Problem PM1052 opened....*
5. The problem is managed using ServiceCenter. This could include automated paging of an appropriate technician to fix the problem.
6. When the problem ticket is updated in ServiceCenter, one or more SCpmUpdated events will flow back to Tivoli and appear in the TEC. If the severity was changed in ServiceCenter, the TEC severity of all of the TEC events containing the same *referenceNo* value as the original SCpmOpen event is updated in the TEC.
7. When the problem is resolved and closed in ServiceCenter, an SCpmClosed event appears in the TEC, and all TEC events containing the *referenceNo* value associated with the SCpmClosed event are closed.

This example uses the *SCpmOpen*, *SCpmOpened*, *SCpmUpdated*, and *SCpmClosed* events supplied with this Plus module. Other events are also supplied that relate to ServiceCenter Inventory and Change Management applications. You can also create your own custom events within ServiceCenter.

Creation of custom ServiceCenter events is discussed in the *ServiceCenter Administrator's Guide* and in the SCAutomate technical documentation for each operating platform.

TME 10 Inventory Component

The SCPlus Tivoli Inventory integration consists of a Java 1.1.6 application embedded with a Javascript Interpreter (FESI version 1.1.1) as well as the event monitor *scevmon* that comes with the rest of the product. The output of the Tivoli Inventory data is translated into ServiceCenter compatible events and placed in the *scevents* queue file to be picked up by the event monitor *scevmon*. By default, the generated event types are *icmServer* and *icmWorkstation*. These are not event types that are shipped with ServiceCenter before version 3 and therefore require the loading of an unload file *icmNew.unl* from the CDROM.

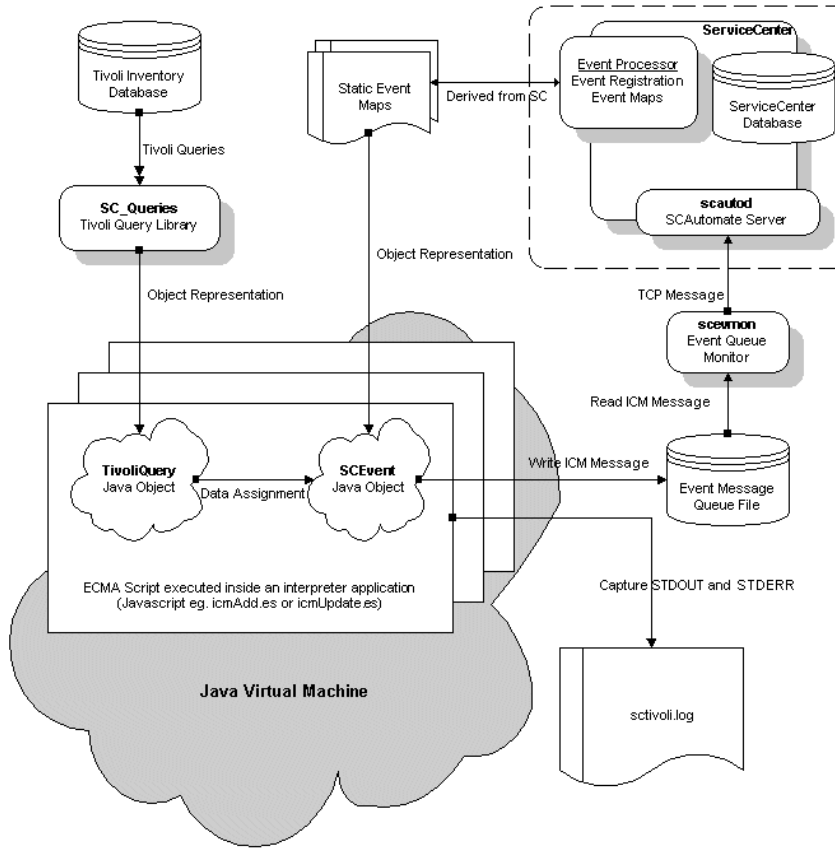


Figure 3: TME 10-ServiceCenter inventory

The system is a *pull* type of integration, which means that it is not driven by any events in the Tivoli TME 10 environment. Rather, the operator issues an **add** or **update** command from the Launch icon of the Plus Module to initiate the process. However, it is feasible to set up a scheduled Tivoli Task to do periodic synchronization of the inventory data.

Currently, TME 10 Inventory performs only static discovery of inventory on systems, which means that it does not create an event when a change in inventory is detected.

SCAutomate supports all database implementations of TME 10 Inventory, and it provides a mapping layer to map ServiceCenter Inventory fields to the TME 10 Inventory Configuration Repository Data Model. Mapping is provided to produce standard mappingas products are shipped. It is possible to modify the ServiceCenter mappings as the Data Model is changed. An open design allows our customers to tailor the system to their requirements.

2 Installation

CHAPTER

This chapter introduces the installation process for the ServiceCenterPlus (SCPlus) for Tivoli TEC module. The module software is distributed on CD ROM and an appropriate device for reading from this medium must be connected at the location where you intend to install the product. The CD ROM contains a subdirectory of files in the standard Tivoli Install media format.

The following topics are covered:

- Installation notes
- SCPlus module de-installation
- SCPlus module installation
- Tivoli Administrator setup
- SCPlus secondary installation and configuration
- TEC event server customization

Installation Notes

You will install the Plus Module only on the TMR server (via the desktop). Then, to install plus module related files into local directories, you will execute the *Install ServiceCenter Plus* task in the Plus Module from any managed node of the TMR.

- If you want to enable problem integration, you will need to execute the *Install ServiceCenter Plus* task on the TEC server managed node for the TMR.
- To remotely manage the ServiceCenter server using the Plus Module tasks you will need to execute the *Install ServiceCenter Plus* task on the ServiceCenter server managed node for the TMR.
- To enable Inventory Integration, you will need to execute the *Install ServiceCenter Plus* task on the managed node that has TME 10 Inventory installed for the TMR.
- Most of the time all three managed nodes reside on the same TMR. In this case, you will only need to install the Plus Module on one TMR.
- It is important that the Plus Module is only installed once in the TMR server (via the Tivoli desktop/Tivoli install) because any subsequent attempts at installing it into a Managed Node again in the same TMR will result in error.
- In the situation where there are multiple TMR servers, the Plus Module needs to be installed on every TMR server that manages nodes that need the Plus Module binaries (needs to execute the *Install ServiceCenter Plus* task).

SCPlus Module De-Installation

Before you install the SCPlus module, the Tivoli administrator may want to de-install previous versions of the product.

Warning: Before de-installing previous versions of the SCPlus module, you should make a backup of the entire Tivoli System.

If you have an adapter product earlier than version 1.0g, you must first un-install it using the shell script *scuninstall.sh* supplied in the *root* directory on the CD-ROM. The script is a bash script and may be run in the root directory in UNIX or Windows NT using the *bash* shell provided by Tivoli. The script must be run by the privileged Tivoli administrator if not the root user on the system.

Note: The de-installation requires a /tmp directory in which it will create an *SCPlus.old* directory to hold the de-installed product. In Windows NT, this directory must be created before de-installation.

If after running the uninstall script, parts of it did not execute correctly, use the following commands (described below for UNIX and Windows NT) to locate residual files and remove manually.

On UNIX systems

1. Change directory to the Tivoli home directory, e.g., */usr/tivoli*.
2. Execute **find . -name "*service*" -print** to locate all files that belong to ServiceCenter and remove.
3. Use upper case "***Service***" and repeat step 2.
4. Use the wildcard "***scplus***" and repeat step 2, then use upper case "***Scplus***".

On Windows NT

1. Use the **Find** command or dialog from Windows Explorer and supply these words as the search files names:
   a. "Service"
   b. "service"
   c. "scplus"
   d. "Scplus"
2. Manually remove files that belong to "*ServiceCenter*" or "*SCplus*".

SCPlus Module Installation

The following installation procedure assumes you are already running the Tivoli enterprise management system as the *ROOT user* and have inserted the SCAutomate products CD-ROM containing SCPlus for Tivoli into the CD-ROM drive of the TME host on which you are installing the module.

To install the SCPlus module:

1 From Tivoli's TME Desktop for Administrators screen, go to the **Desktop** pull down menu and select the **Install Product** option.

Figure 1: TME Desktop for Administrators screen

Important: If the Tivoli Installer's Media setting is currently pointing to a location that does not contain a Tivoli-packaged software product, the Tivoli installer will display an error pop-up screen. This can happen if the media setting is your CD ROM and some non-Tivoli CD is currently loaded, or if the media setting is a temporary disk directory that no longer contains a Tivoli-packaged software product.

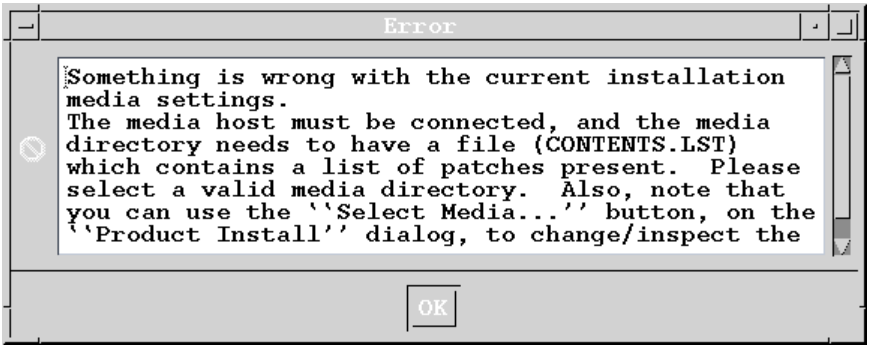


Figure 2: Error box

- If an error box is displayed, click **OK** and go to the next step.
- If an error box is not displayed, then the Tivoli Installer is pointed to the correct directory, and the normal Install Product window will appear as shown in Figure 2. In this case, go on to step 4.

The File Browser window shown in Figure 3 is displayed.

2 In the *Path Name* field at the bottom of the window, set the path to the cdrom location containing the SCPlus for Tivoli installation files. This will typically be:

/cdrom/servicecenter.image.12a on Solaris

-or-

D:\servicecenter.image.12a on Windows NT

When the correct path is selected, a series of files starting with **contents.lst** is shown in the File Browser window.

Figure 3: Setting Path to Tivoli Install Media

3 Click the **Set Media & Close** button to exit the File Browser window.

The Install Product window is displayed.

4 Verify that the correct product name (*ServiceCenterPlus for Tivoli*) appears in the window. If some other product name appears, you are pointing to the wrong Tivoli-packaged installation media.

5 Install the Plus Module only on the TMR server (via the Tivoli desktop/Tivoli install).

Important: The SCPlus Module should be installed only *once* in the TMR server, because any subsequent attempts to install it into a Managed Node in the same TMR will result in error.

6 When the appropriate target host appears in the **Clients to Install On** window, highlight the SCPlus for Tivoli product name in the **Select Product to Install** window, and then click on **Install & Close**.

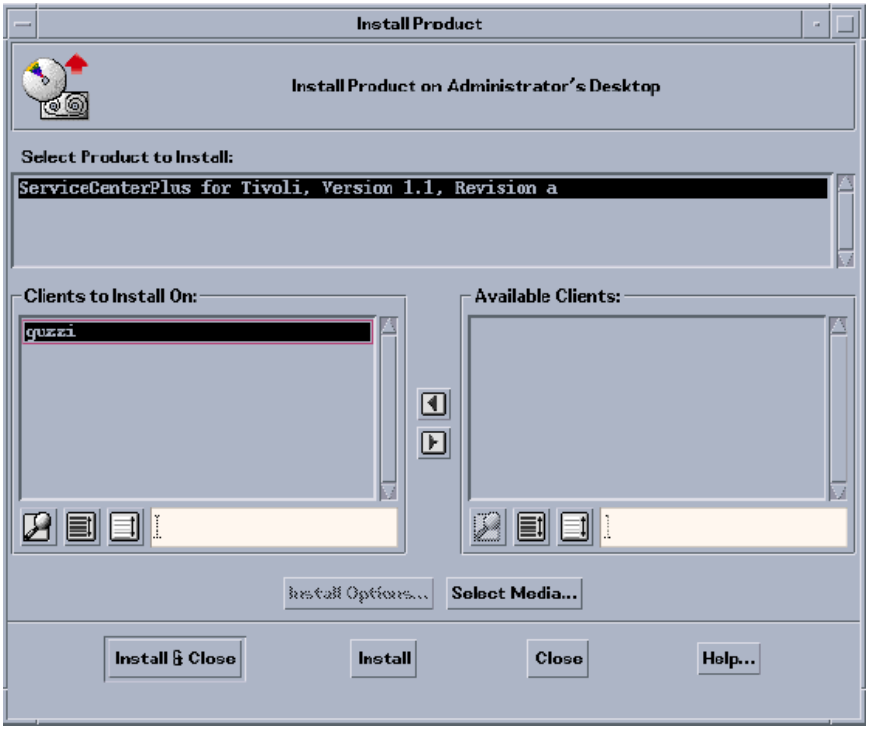


Figure 4: Selecting the client sites and the products to install

An initial output file is displayed stating that you are about to install this Tivoli module on the specified host.

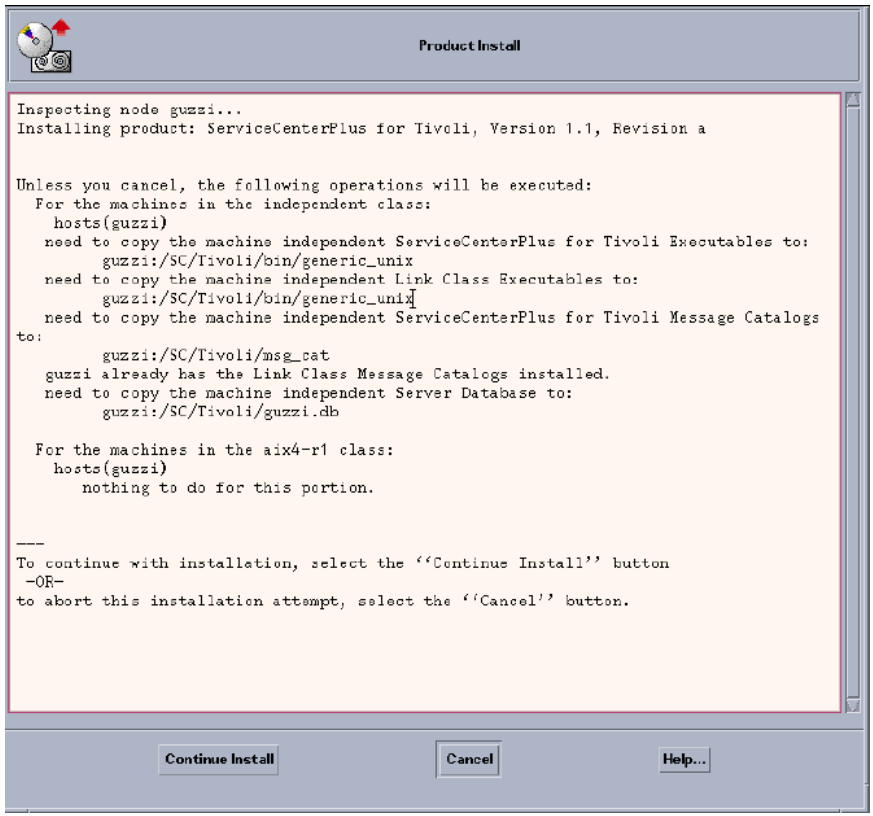


Figure 5: Initial product install output file

7 Click on **Continue Install** to verify your intention to install this Tivoli module.

Note: If you have installed other Plus modules or SCPlus for Tivoli, the messages shown in Figure 5 may be slightly different on your system.

Additional text is written to the output window, displaying the progress of the SCPlus for Tivoli installation process. The module installation terminates with a statement verifying the complete install and creation of each component of the SCPlus for Tivoli module.

Important: Contact Peregrine Systems Technical Support if you receive output text that indicates the product may not have installed properly and you are unable to correct the problem.

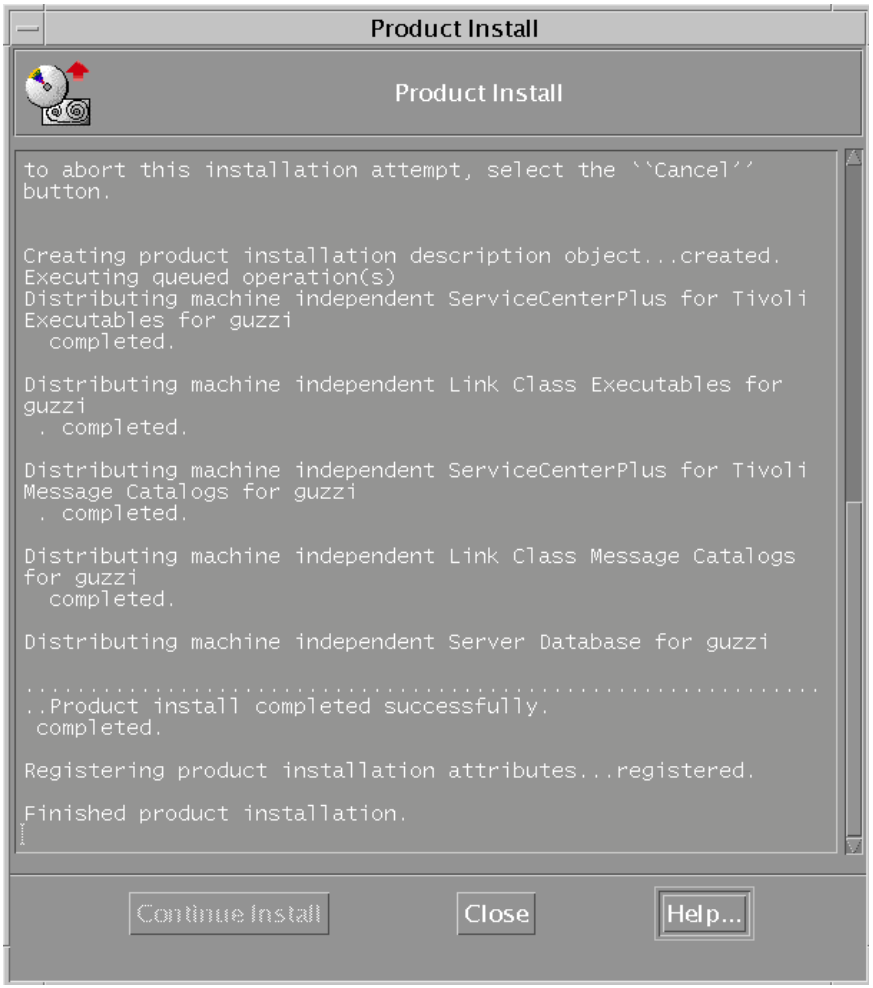


Figure 6: Product install screen

Basic installation of the SCPlus for Tivoli product is now complete. You must now perform the secondary installation and customization procedures that follow.

Tivoli Administrator Setup

The SCPlus module needs a UNIX or Windows NT userid to be recognized by Tivoli. Specifically, it requires a Tivoli administrator with *admin* and *user* roles to be associated with the UNIX or Windows NT userid of the ServiceCenter Events Monitor process.

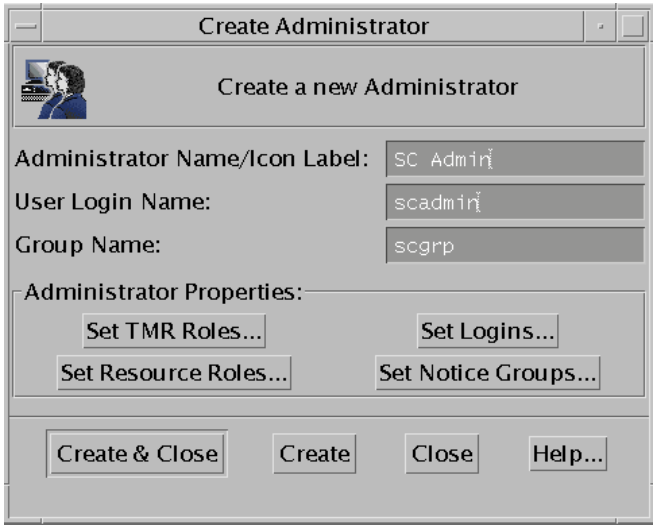
Note: You must define the administrator's userid in the operating system environment *before* you begin this process.

To setup a new Tivoli administrator:

1 Double click on the Administrators icon to open the Administrators window from the TME Desktop.

2 Create a new Tivoli administrator by clicking on **Create Administrator** from the **Create** menu.

The Create Administrator screen is displayed.



3 Type in the userid of the Tivoli Administrator in the *Administrator Name/ Icon Label* field. This is the userid the module will assume as it connects and interacts with Tivoli. Use the userid that will be used to own the ServiceCenterPlus directory.

Important: This userid will later be applied to the Secondary Installation task install of SCPlus.

4 Fill in the remainder of the *Create Administrator* data entry fields appropriately.

5 Click the **Set Logins...** button.

A window is displayed from which you can register new login IDs with Tivoli.

6 Type your User Login Name in the *Add Login Name* field.

7 Press **Enter** to create the login and move it into the Current Login Names window.

8 Click on **Set & Close** to confirm the add, and return to the Create Administrator window.

To set up the TMR Roles:

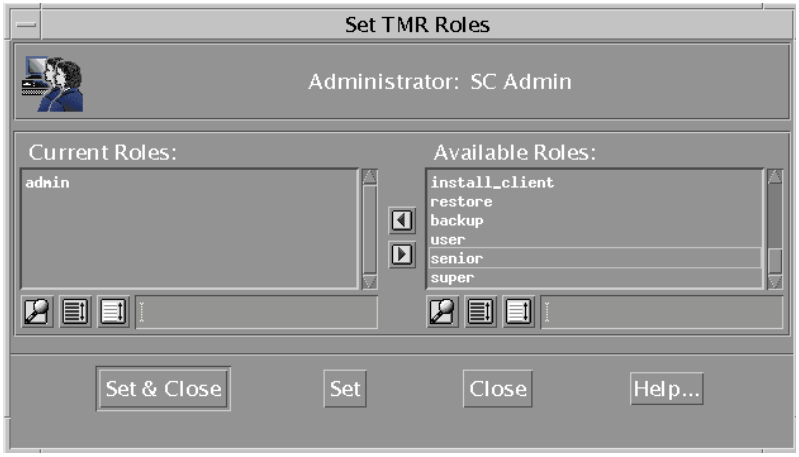
1 Click the **TMR Roles** button to access the set window.

The Set TMR Roles screen is displayed.

2 Locate the **admin** and **user** roles in the Available Roles column. These are the only required roles for all systems.

3 Double click to move these roles into the Current Roles column.

4 Click the **Set & Close** button when you have the appropriate roles moved to the Current Roles column.



Finally, you need to specify the Resource Roles for the new Tivoli Administrator. The Set Resource Roles screen prompts you to select the Resources, system applications, and tasks to which this user will have access.

To specify Resource Roles:

1. Select **EventServer** in the Resources column.
2. Select **admin** from the Available Roles column.
3. Double-click **admin** to move it to the Current Roles column.

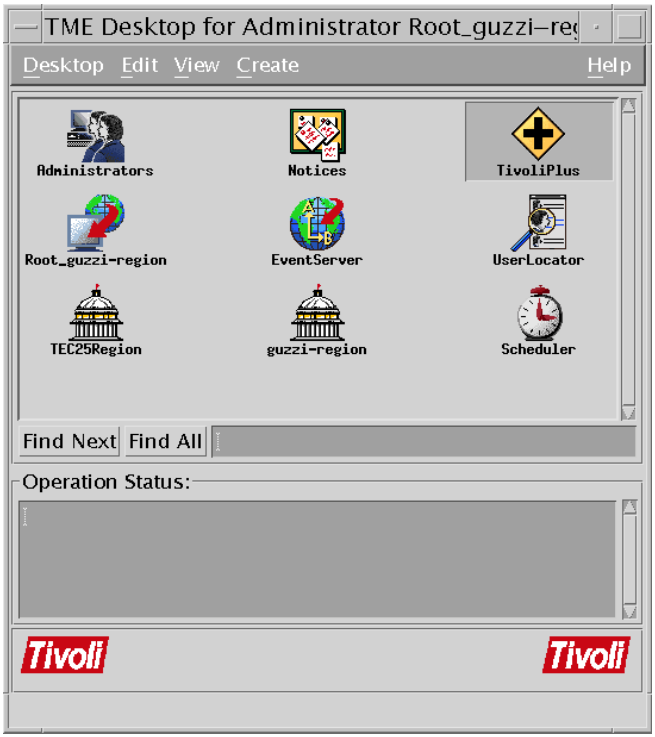
4. Click the **Set & Close** button to confirm the specific Resource Roles of this Administrator.
5. Click on the **Create & Close** button after you have filled in all the appropriate subwindows and fields. The Administrators screen is immediately updated to show the new icon for this administrator.

Important: It is best to verify the creation of this new Tivoli administrator via a command-line interface with the system. In some cases, you may be required to manually create this user on the Tivoli system via the command-based interface.

SCPlus Secondary Installation and Configuration

When the basic installation of the SCPlus module is complete and a Tivoli account has been created for it, it is necessary to run some secondary installation and configuration scripts created during the basic installation.

1 Double click the TivoliPlus icon from the the TME Desktop for Administrators window.



The TivoliPlus window is displayed.

2 Double-click on the ServiceCenterPlus for Tivoli icon.

The SCPlus for Tivoli screen is displayed, which contains the icons and applications for the Plus module.

If the *Install ServiceCenterPlus* task is being run on multiple target hosts, which it often must be, it can be run simultaneously on all hosts; or one host at a time.

To specify which host:

a. With the right mouse button, click the Install ServiceCenterPlus task icon.

b. Choose *Modify Job*.

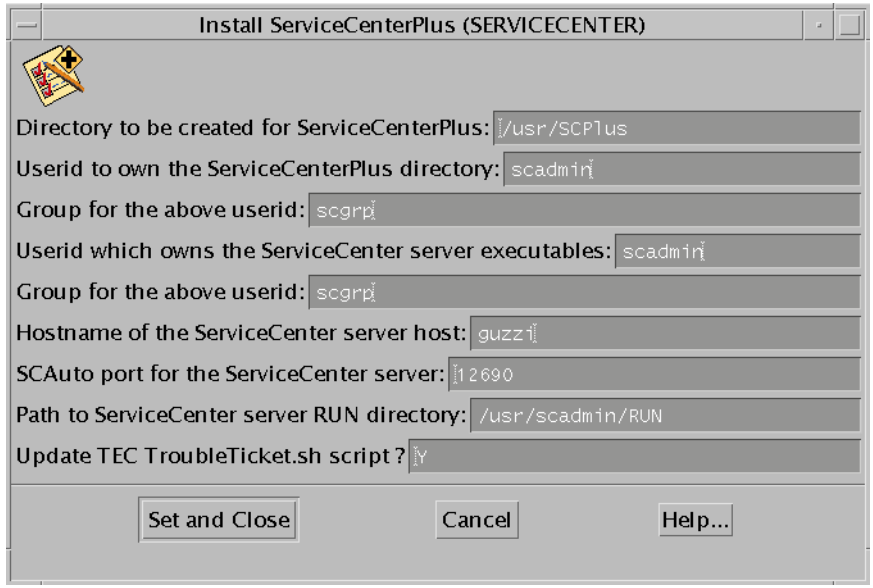
c. Select the target hosts.

When deciding whether to run the task on more than one host at a time, consider these issues:

- If the destination directory for SCPlus is to be something other than the default of */usr/SCPlus*, such as */apps/SCPlus*, and the task is to be run simultaneously on multiple hosts, then the customer must ensure that the directory */apps* (or whatever you chose) actually exists or can be created on each of the target hosts.
- It may not be appropriate to update the TEC default trouble ticket shell script *TroubleTicket.sh* on some hosts.

3 Double click on the Install ServiceCenterPlus icon.

A small input form is displayed, where you are asked to specify certain IDs and addresses.



4 Enter data in the fields of this form as follows:

Directory to be created...

ServiceCenterPlus requires its own directory, called */usr/SCPlus* by default. Enter the name of the directory to be created. In Windows NT, you may enter the drive letter as well (for example, *c:/usr/SCPlus*). You may also use forward or backward slashes (UNIX or Windows NT conventions).

Userid to own...

This field requires the *userid* of the Admin-level user who will own all ServiceCenterPlus Executable files. This is the userid of the Tivoli Administrator created in the previous section.

Group for the above userid

In UNIX, this field requires the *groupname* of the specified user from the previous field. This is the group to which the Tivoli Administrator created in the previous section belongs. In Windows NT, this field is ignored.

Userid which owns...

For correct functioning of the ServiceCenter operational tasks delivered with this Plus module, ServiceCenterPlus must know the User ID and group under which ServiceCenter server was installed. Locate the actual ServiceCenter server installation and determine the owner of the files contained in the RUN directory.

Group for the above userid

In UNIX, this field requires the *groupname* of the specified user from the previous field. In Windows NT, this field is ingored. For convenience and clarity, it is best to make the same userid available for both of the above userid inputs. This is not required, but is recommended. If the User ID's differ, they must at least have executable rights to each other (for example, they should have the same user group and have group executables).

Hostname of the ServiceCenter...

Supply the TCP/IP hostname of the ServiceCenter server host.

ScAuto port...

Consult the *sc.ini* file in the RUN directory of the ServiceCenter server installation referenced above, and determine the value coded for the **scauto**: parameter. Put that value here. This is the TCP port number or name on which the **scautod** daemon at the ServiceCenter server listens in order to receive events.

Path to ServiceCenter...

On the hostname given above, this field is the specific path to the ServiceCenter RUN directory of the Managed Node on which ServiceCenter is installed.

Update TEC TroubleTicket.sh script?

This field relates to whether you want to replace the default Tivoli TroubleTicket.sh so that trouble tickets may be opened manually from the TEC. The values are **Y** (yes) and **N** (no).

Peregrine Systems highly recommends that you select **Y**; this allows the manual creation of Problem Tickets from *any* TEC event, not just the specially coded events.

If the ServiceCenter server host is on a host that does *not* contain a TEC component, the field labelled *Update TroubleTicket.sh?* should be left blank or set to **N**. If it is left as **Y**, then the *Install ServiceCenterPlus* script tries to update a nonexistent file, and the following error message occurs.

```
cp:/u002/tivoli/tme/bin/hpux10/TME/TEC/: No such file or directory
chmod: can't access /u002/tivoli/tme/bin/hpux10/TME/TEC/TroubleTicket.sh
```

This is a benign error and can be ignored if it occurs.

1 Click **Set and Close** to confirm the configuration and perform secondary installation and configuration of SCPlus for Tivoli. If you later need to change any of these values, you can rerun this script, or if you only need to change ServiceCenter-related items, use the Task called *Define Target ServiceCenter Server*.

Important: You MUST run the *Install ServiceCenterPlus* Task on ALL of the hosts on which the basic installation was performed earlier. This task must run on the ServiceCenter host, the TEC host, and the TMR host, unless these are the same machine. In the case where these are separate hosts, the task must be modified to run on all the hosts. Do this by clicking the right mouse button on the icon and selecting the **Modify Job** command. Then select the appropriate hosts from the *Available Clients:* list and move them to the *Clients to Install On:* list. Save and close, then double click on the task to execute it.

2. The remainder of the configuration steps assume that the target ServiceCenter server is correctly defined and is operational. Verify this by running the ServiceCenter Server Status task from the ServiceCenterPlus tasks window.
3. Before starting the task, click the right-mouse button over the task icon to modify the ServiceCenter Server Status task to run on the correct Tivoli-managed host.
4. Double click the ServiceCenter Server Status task icon.

   If the items below are *true*, then a list of processes and IPC resources is displayed in the task output. This may take a few moments.

   - ServiceCenter is correctly installed at the location you specified previously.
   - The *Install ServiceCenterPlus* task was run on that host.
   - ServiceCenter is actually running.

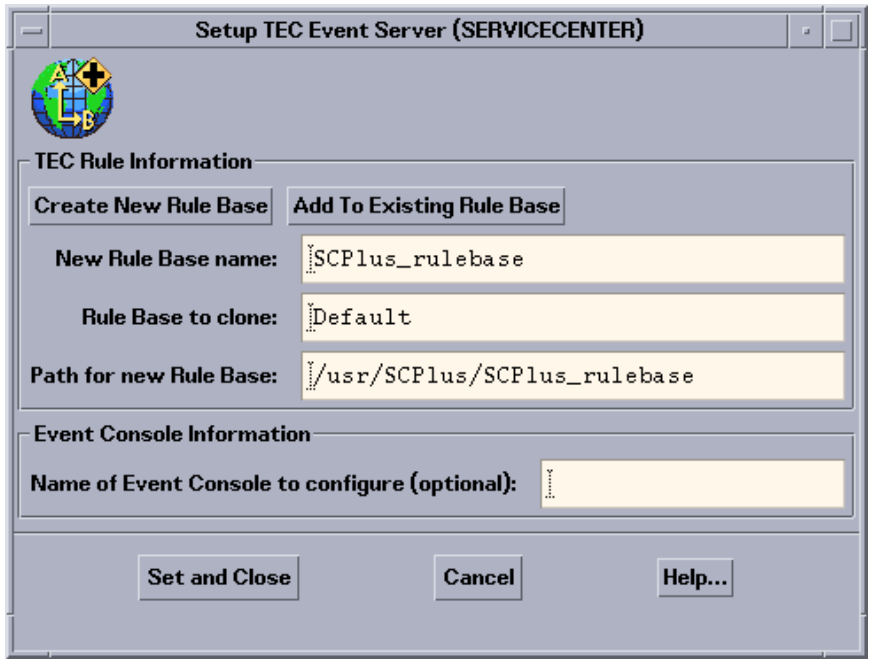
5. Proceed to the next section, *TEC Event Server Customization*, where you integrate the ServiceCenterPlus rules and classes with the Tivoli TEC.

TEC Event Server Customization

To customize the TEC event server:

1 Click the Setup TEC Event Server icon from the TivoliPlus window.

The Setup TEC Event Server dialog box is displayed.



The purpose of this step is to integrate all the *.rls* and *.baroc* files that are delivered with SCPlus for Tivoli into your current rule base. The choices you make here depend on your particular environment. For example, if this is a new test system containing no particular installation-specific rule customization, you may want to simply create a new rule base, using the Default rule base as a starting point, containing the ServiceCenterPlus rules (*scenter.rls*) and classes (*scenter.baroc*). Alternatively, you may have performed significant customization and you may want to add the ServiceCenterPlus rules and classes to an existing rule base.

2 Click on either **Create New Rule Base** or **Add to Existing Rule Base** accordingly.

- If you are installing on a new test system containing no TEC event customization you care about, it is recommended that you click on **Create New Rule Base**. Keep the present default values, and click the **Set and Close** button.

  This creates a new rule base at the default location, with a name of *SCPlus.rulebase*. It will contain a copy of the default rule base, with the ServiceCenterPlus rules and classes added to it. Such a rule base is appropriate for evaluation purposes, but not necessarily for production.

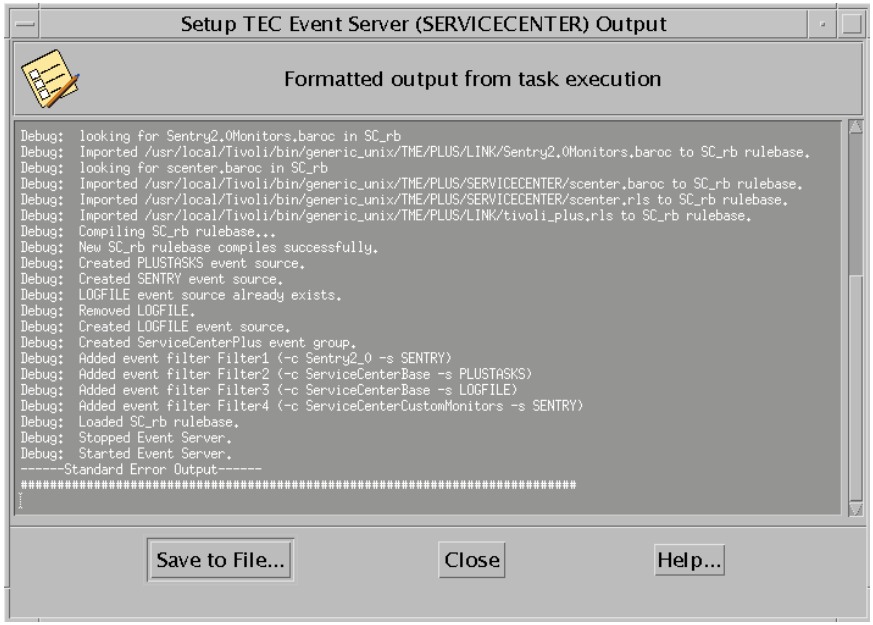
- If on the other hand you are installing ServiceCenterPlus on a production system, or in a test system that contains a copy of your production rules, you should either click the **Add to Existing Rule Base** button and specify an existing rule base for *scenter.rls* and *scenter.baroc* to be added to, or click the **Create New Rule Base** button and specify your existing rule base as the **Rule Base to Clone** as a starting point for the new rule base. Then click the **Set and Close** button to create or update the rule base.

Note: If you are uncertain how to proceed in this step, consult the person at your site responsible for TEC customization, a Tivoli consultant, or the Tivoli TEC documentation.

After you click the **Set and Close** button, a new rule base is created or the existing rule base is updated. As part of this process, the TEC Event Server is halted briefly and then restarted. A pop-up notification is displayed.

Do not be concerned when you see this message. If you want, you can customize the TEC Event Server offline to avoid disturbing time-sensitive TEC events.

The TEC Event Server then creates the verification rule base and produces an output file.



3 Wait for the output file to complete, and verify there are no errors.

4 Decide whether you want to save the output to a file, then click **Close** to close the screen.

The new rule base is now loaded.

5 Right-click on the **Application Launcher** icon to reveal the menu of choices. Then click on **Start TME <--> SC Events Processor**.

This starts the background process called *scevmon*. If you installed ServiceCenterPlus to the default directory, scevmon is executed out of */usr/SCPlus/bin*.

6 Check the file */usr/SCPlus/sctivoli.log* to verify that scevmon started correctly and was able to establish contact with the *scautod* listener in the ServiceCenter server.

This completes the installation and configuration of ServiceCenterPlus for Tivoli. To verify the end-to-end operation of the product, open a TEC console window that can see all events, and issue the following command from a terminal window on the TEC host:

```
wpostemsg -r HARMLESS hostname=test category=example SCpmOpen
ServiceCenter
```

This posts an SCpmOpen event of HARMLESS severity to the TEC.

One of the rules in *scenter.rls* forces HARMLESS and UNKNOWN serverities for SC events to WARNING, so the event will appear as a WARNING event. Shortly thereafter, if ServiceCenter is running and if *scautod* is listening at the SCAuto TCP port defined earlier, and if *scevmon* was started successfully, an *SCpmOpened* confirmation event appears in the TEC. The event contains the ServiceCenter problem number assigned to the ticket opened via the SCpmOpen request. If this happened, you have successfully implemented automated trouble ticketing with Tivoli. If not, review the */usr/SCPlus/sctivoli.log* for error messages and do the following:

1. Double-check the SCAuto port that you instructed ServiceCenterPlus to send events to. Does it match the value coded for the **scauto:** parameter in the *sc.ini* file in the RUN directory of the ServiceCenter server?
2. Verify that ServiceCenter is up and running.
3. Verify that the list of running processes for ServiceCenter includes at least one task called *scautod*.
4. Examine the *scautod.log* in the *logs* directory of the ServiceCenter server installation for any error messages.
5. Use *netstat -f inet | grep nnnnnn* to check for a listening process on the indicated SCAuto port on the ServiceCenter server host. Replace *nnnnnn* in the command above with the port number. For example:

   ```
   netstat -f inet | grep 12690
   ```

   would be used if your ServiceCenter SCAUTOD process is using the default port for scauto events. The correct output for this command would be:

   ```
   netstat -f inet | grep 31112

   scultra2.49470    scultra2.31112    32768   0 8192    0 ESTABLISHED
   scultra2.31112    scultra2.49470    8192    0 32768   0 ESTABLISHED
   ```

   This output assumes the following:

   - an established connection exists between *scautod* on the ServiceCenter host and *scevmon* on the TEC host
   - a port number of 31112
   - both processes running on host *scultra2*, :

3 TEC Configuration

CHAPTER

ServiceCenterPlus for Tivoli TEC relies upon the Tivoli TME 10 Enterprise Console (TEC) for much of its active event processing. To accommodate this processing, the TEC needs to be configured correctly. This can be done in an automated way using the pre-configured Tivoli task named *Setup TEC Event Server*, as shown in Chapter 2.

The TEC configuration can also be done in a manual fashion. This chapter describes the configuration that needs to be done in a step-by-step manner. This chapter explains Tivoli TEC concepts and procedures for ServiceCenter users unfamiliar with Tivoli.

The following topics are covered:

- TEC defined
- Configuring TEC for your environment
- Developing custom rules

TEC Defined

The TEC is a complex event-processing application composed of several components:

- Event server
- User-defined event consoles
- Event adapters

Event server

The Event Server is the central processor of the enterprise events. It uses static, defined *event classes* to categorize all possible events that it will process. It also uses static, defined *rules*, contained in specific rule-sets, to chart the course of processing actions for incoming events.

- *Rules* begin processing if they match incoming events, based upon the class of the incoming event.
- *Rule-sets* are imported into Rule Bases, which are then compiled into machine code for performance reasons.
- Compiled Rule Bases may be loaded into the Event Server, causing the execution of the Rule Base's rules against the incoming events (generated from the Event Adapters), and causing specific actions against the events to appear on the defined Event Consoles.

Event consoles

The Event Consoles provide graphical views of these events. Event Consoles are defined for specific Tivoli administrators, and they typically filter out whole classes of events so that individual Tivoli administrators are not overwhelmed with all enterprise events on their graphical consoles. For convenience, such filters are constructed in Event Groups, which allow both general and specific configuration of the events to be displayed to Event Consoles.

Event adapters

The Event Adapters convert management application data and log data into meaningful events to be processed. The TEC provides sample Event Adapters for operating system logfiles, network node managers, and other applications.

Event classes

A high-level program language called BAROC (BAsic Recorder of Objects in C) is used to define all types of events that may be processed. These definitions are called Event Classes. Specific instances of these classes are *events*. An Event Adapter has its own set of Event Classes in which it will generate events for the TEC.

For the SC Automate for Tivoli product, Peregrine Systems has defined a set of Event Classes related to Problem Management. These Event Classes are given in the *scenter.baroc* file, portions of which are shown below:

```
TEC_CLASS :
   SCenterPM ISA EVENT
   DEFINES {
         source: default= "ServiceCenter";
         sub_source: default= "SCAuto+Tivoli";
         evtype:STRING;
         evtime:STRING;
         evusrseq:STRING;
         evuser:STRING;
         evpswd:STRING;
         evsepchar:STRING,default= "^";
         assigneeEmail:STRING;
         number:STRING;
         category:STRING;
         actor:STRING;
         priorityCode:STRING;
         severityCode:STRING;
         assignment:STRING;
         referedTo:STRING;
         alertTime:STRING;
         action:STRING;
         StaTus:STRING;
         vendor:STRING;
         openTime:STRING;
         contactTime:STRING;
         backupStart:STRING;
         causeCode:STRING;
         logicalName:STRING;
         group:STRING;
```

```
        jobName:STRING;
        location:STRING;
        version:STRING;
        type:STRING;
        abendCode:STRING;
        model:STRING;
        resolution:STRING;
        keywords:STRING;
        referenceNo:STRING;
        id:STRING;
        downtimeStart:STRING;
        assigneeName:STRING;
        contactName:STRING;
        callerId:STRING;
        contactPhone:STRING;
        networkName:STRING;
        updateTime:STRING;
        updateAction:STRING;
        openGroup:STRING;
        openBy:STRING;
        closeTime:STRING;
        closedBy:STRING;
        resolutionCode:STRING;
        };
END
```

Rules

Another high-level program language, loosely based on the PROLOG language, is used to define all event processing actions as rules for the Event Server. Rules may be built using a graphical tool called the TME 10 Rule Builder, however, most rules continue to be developed and maintained as text files.

For the SC Automate for Tivoli product, Peregrine Systems has developed rule-sets that perform specific event processing based upon the prevailing operational scenario of Problem Management/Event Management. These rules are given in the *scenter.rls* file, portions of which are shown below:

```
/*************************************************************/
/*                                                           */
/* (C) COPYRIGHT Peregrine Systems, Inc. 1996, 1997, 1998    */
/* Portions (C) COPYRIGHT Tivoli Systems, Inc. 1996, 1997, 1998*/
/* All Rights Reserved                                       */
/* Licensed Material - Property of Peregrine Systems, Inc.   */
/*                                                           */
/*************************************************************/

/* "ensure_refno" ensures we have a referenceNo value in all SC TEC events */
/* "referenceNo" needs to be in all SC events created in the TEC. This */
/* slot value is propagated into the SC ticket and used as a correlator to */
/* later update or close the TEC event if the SC ticket is updated or closed. */
/* The slot "referenceNo" is the concatenation of event_server, event_handle, */
/* and date_reception. */

rule: ensure_refno: (
    event: _event of_class 'SCenterPM'
        where [ referenceNo: equals '',
            date_reception: _dr,
            server_handle: _sh,
            event_handle: _eh ],
    reception_action: set_refno: (
                sprintf( _refno, '%d%d%lu', [ _sh, _eh, _dr ] ),
            bo_set_slotval( _event, 'referenceNo', _refno )
        )
    ).
```

```
/* Force TEC severities having no SC equivalent to 'WARNING' for SC events */

rule: ensure_valid_SC_sev: (
  event: _event of_class within [ 'SCpmOpen', 'SCtestOpen' ]
    where [ priorityCode: equals '',
                    severity: outside [ 'FATAL', 'CRITICAL', 'WARNING', 'MINOR' ] ],
    reception_action: set_severity: (
                bo_set_slotval( _event, 'severity', 'WARNING' )
        )
    ).
```

A typical installation of SC Automate for Tivoli must copy certain Peregrine Systems rules and modify them for site-specific purposes.

Configuring TEC for Your Environment

Overview

To configure TEC for your environment:

1. Define Event Classes and the Event Adapters that will generate these types of events.
2. Develop rules that will apply your business logic to the events you intend to monitor and manage. Put your rules into Rule Sets, and compile these Rule Sets into Rule Bases.
3. Configure your Event Consoles and Event Groups.
4. Load your Rule Bases into your Event Server, and begin processing events.

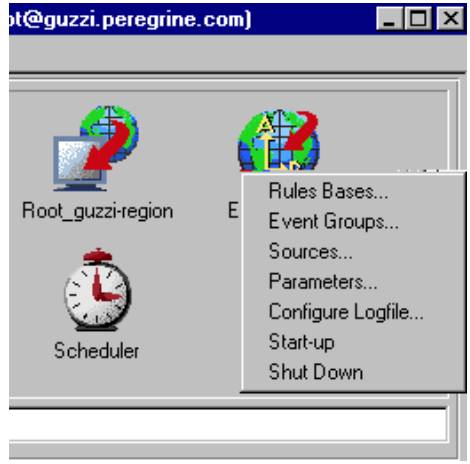
Note: Refer to the Tivoli TME documentation for complete coverage of these subjects.

Detailed configuration steps

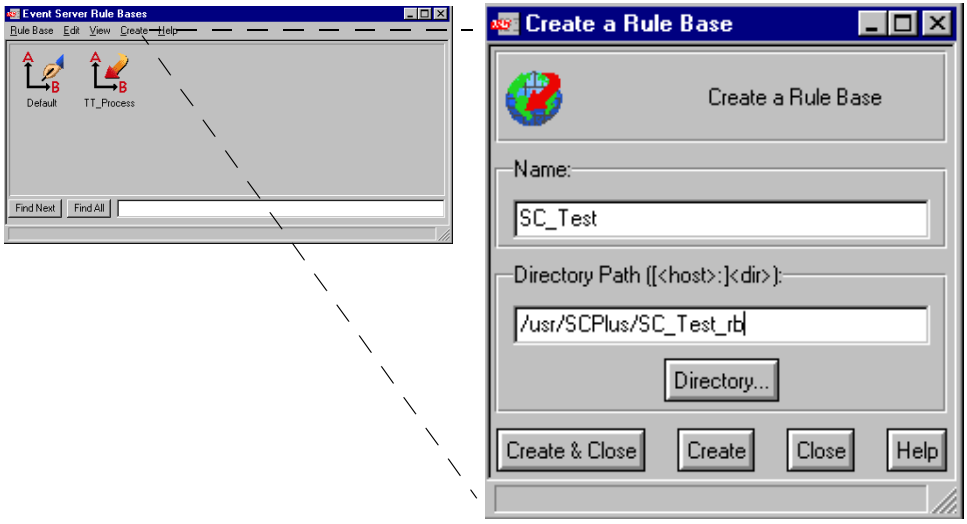
The process of defining Rule Bases, building event groups and event consoles are all shown in the following pages.

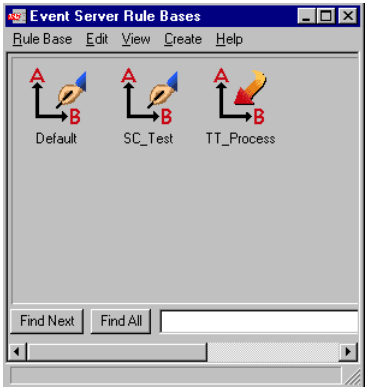
To create a new rule base:

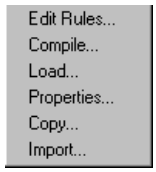
1. From the TME desktop, right mouse-click on the **Event Server** icon, then pull down to the **Rule Base** command.



2 Click on the **Create** menu and select the **Rule Base** option to create a new Rule Base.





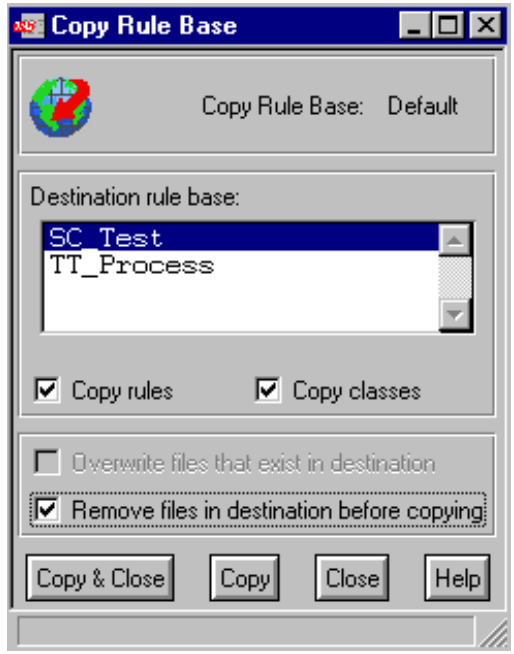


3 Clone the Default Rule Base to your newly created Rule Base:

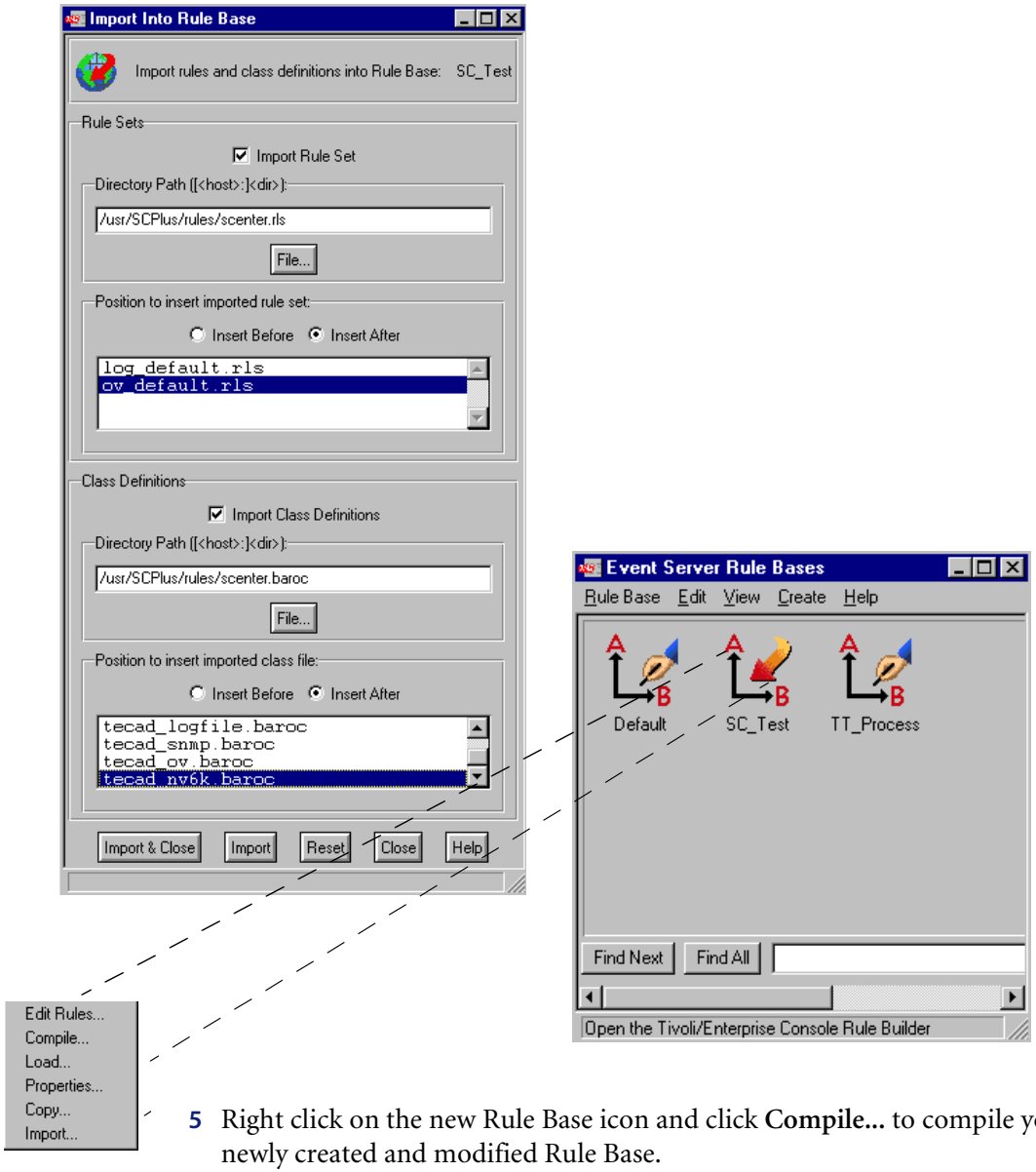
a. Right click on the *Default* Rule Base icon.

b. Click **Copy**.

c. From the dialog box that appears, click the check boxes and select an entry in the Destination rule base field to receive the *cloned* Rule Base.



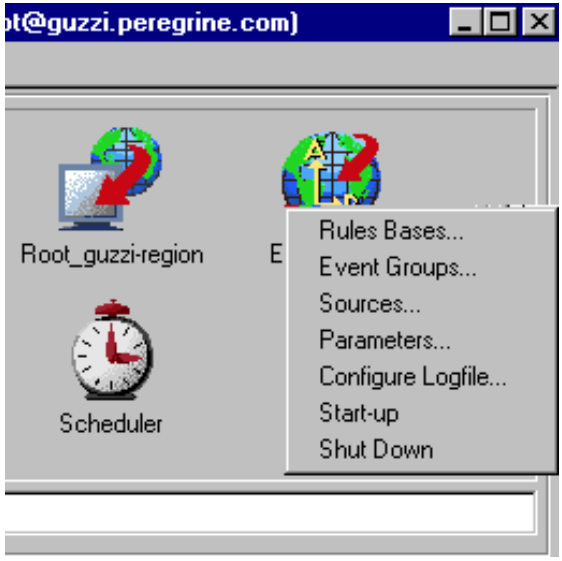
4 Import your newly developed rules, or the pre-defined rules in other Rule Sets.



5 Right click on the new Rule Base icon and click **Compile...** to compile your newly created and modified Rule Base.

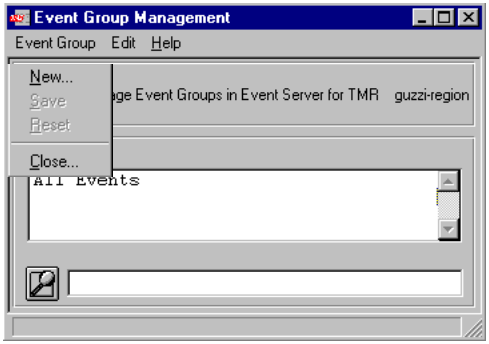
Note: If you wish to debug your rules, you must run the command line version of this command, *wcomprules*, and you must specify the *debug* option, **-t**. If your new Rule Base is named *My_RB*, the command line would be **wcomprules –t My_RB**. Refer to Tivoli TEC documentation for further details.

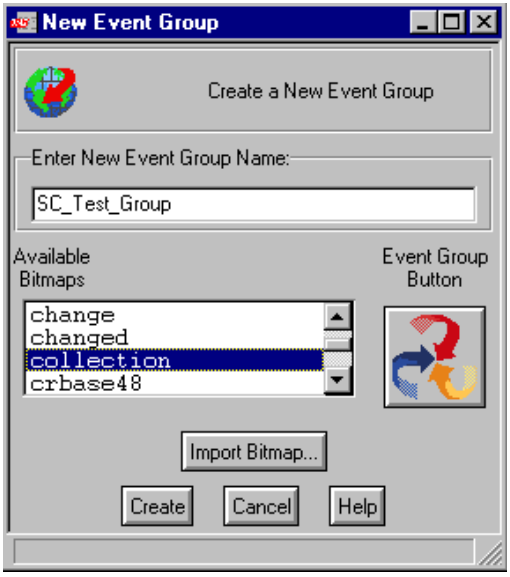
6 Click on Event Groups from the TEC Server icon at the TME desktop. This allows you to build the appropriate Event Groups for the TEC consoles you intend to support.



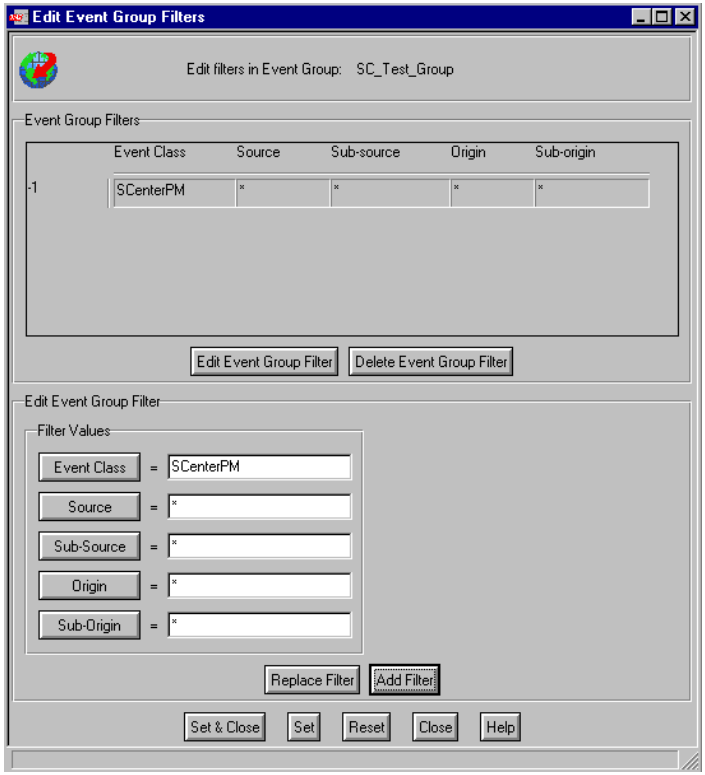
7 Click **New ...** from the **Event Group** menu and type a name for your new group.

8 Click **Create** to confirm the set up and generate the new group.





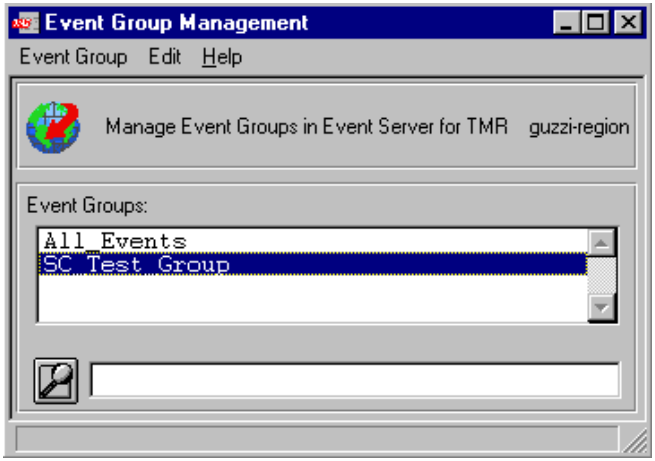
The Event Filter window is displayed.



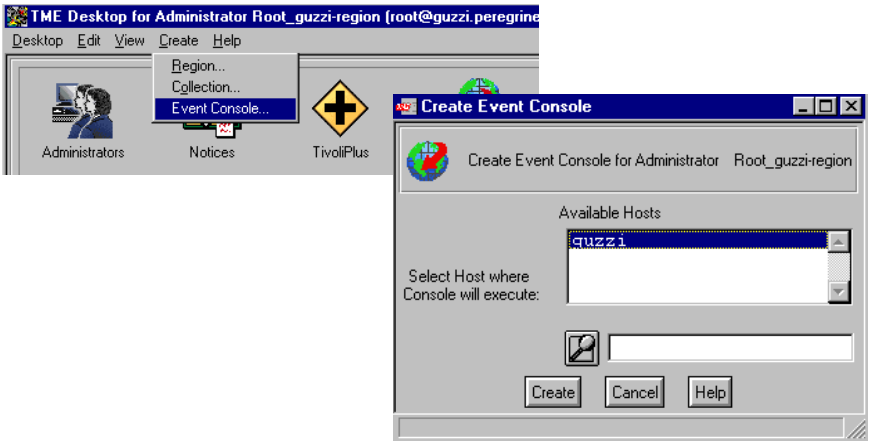
9 Setup the specific class and event types for this group.

10 Click **Set & Close** to confirm the filter setup.

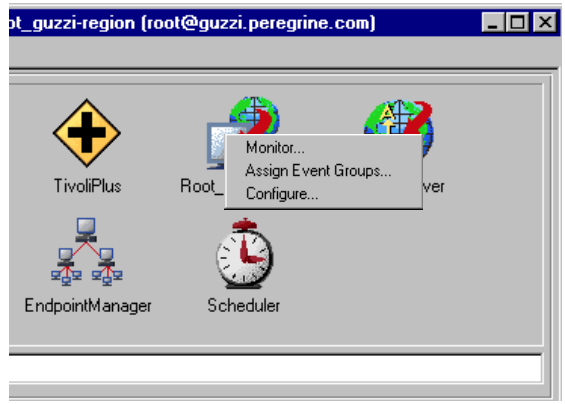
The new group appears in the *Event Group Management* window.



11 Create TEC consoles for all the Tivoli Administrators you intend to support.

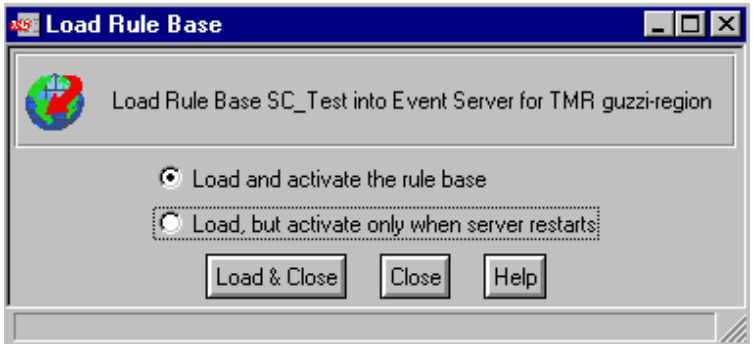


12 From the TME Desktop, right click on a TEC console icon and select **Assign Event Groups** from the pull down menu.



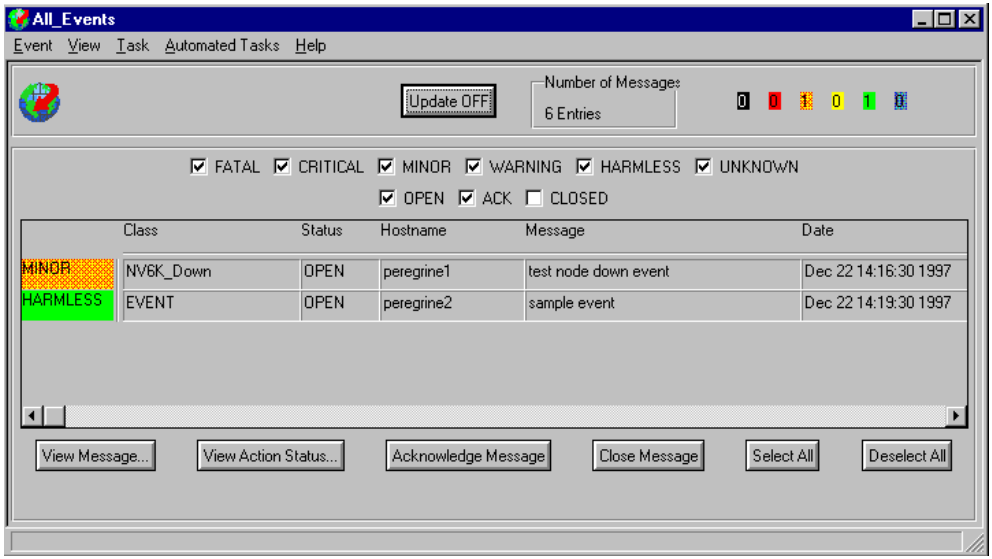
13 Assign specific Event Groups to specific TEC Event Consoles, and assign the authorization roles appropriate to the Tivoli administrators being assigned the Event Groups. This can be used to keep a junior administrator or read-only user from modifying events or executing tasks on events.

14 Click **Load and activate the rule base**, then click **Load & Close**. This loads your Rule Base and processes the events.



Event console

The following sample event console uses an event group named *All_Events*. This event group is assigned to the *Root_tivoli-region* event console. It is displayed below two sample events.



Developing Custom Rules

The easiest approach to developing TEC rules that reflect your business logic is to copy existing rules and modify them to meet your specifications. The following rule is taken from a *scenter.rls* file and shows how the ServiceCenter event is created anytime certain *SC* events are received by the TEC. This rule can be copied and modified for your purposes.

```
/* "create_SC_Event": What we do when an SC event is created in the TEC */
/* Fires when we see an 'SCpmOpen','SCpmUpdate', or 'SCpmClose' TEC event. */
/* These Peregrine Systems-defined events are requests to open, update, or close an
SC ticket. */
/* To make an SC ticket for a TEC event, create an SCpmOpen event from the TEC
event */

rule: create_SC_Event: (

   description:
    'Create ServiceCenter Event from Peregrine Systems-defined TEC Event',

   event: _ev1 of_class within [ 'SCpmOpen',  'SCtestOpen',
                        'SCpmUpdate','SCtestUpdate',
                        'SCpmClose', 'SCtestClose' ] ,

   reception_action: run_sceventin: (
   (exec_program(_ev1, '/usr/SCPlus/lib/sceventin.sh', '', [], 'YES'))
         )
   ).
```

ServiceCenter rule summary

This section describes the scripts that are run from within TEC rules to accommodate the TEC <--> SC event integration.

sceventin.sh

This shell script should be launched to open a problem ticket. It creates a *pmo* input event in ServiceCenter.

scpmusev.sh

This shell script should be called to update ServiceCenter when the severity is changed on an event that opened a problem ticket. It generates a ServiceCenter *pmu* input event.

scpmuack.s

This shell script should be called to update ServiceCenter when the event that opened a problem ticket is acknowledged in the TEC. It generates a ServiceCenter *pmu* input event.

scpmclose.sh

This shell script should be called to close a ticket in ServiceCenter when the initiating event is closed in TEC. It generates a ServiceCenter *pmc* input event.

The ServiceCenter *Output Event Rule Commands* that are provided with the SCPlus for Tivoli module are as follows:

sctmeack.sh

This shell script should be launched to acknowledge a TEC event. The script is usually caused by a ServiceCenter output event *pmu*.

sctmesev.sh

This shell script should be launched to update the severity of a TEC event. The script is usually caused by a ServiceCenter output event *pmu*.

sctmeclose.sh

This shell script should be launched to close an event in the TEC usually caused by a *pmc* output event from ServiceCenter.

4 ServiceCenter Configuration

CHAPTER

This chapter describes the process of configuring ServiceCenter for use with ServiceCenterPlus for Tivoli TEC. The requires configuration of:

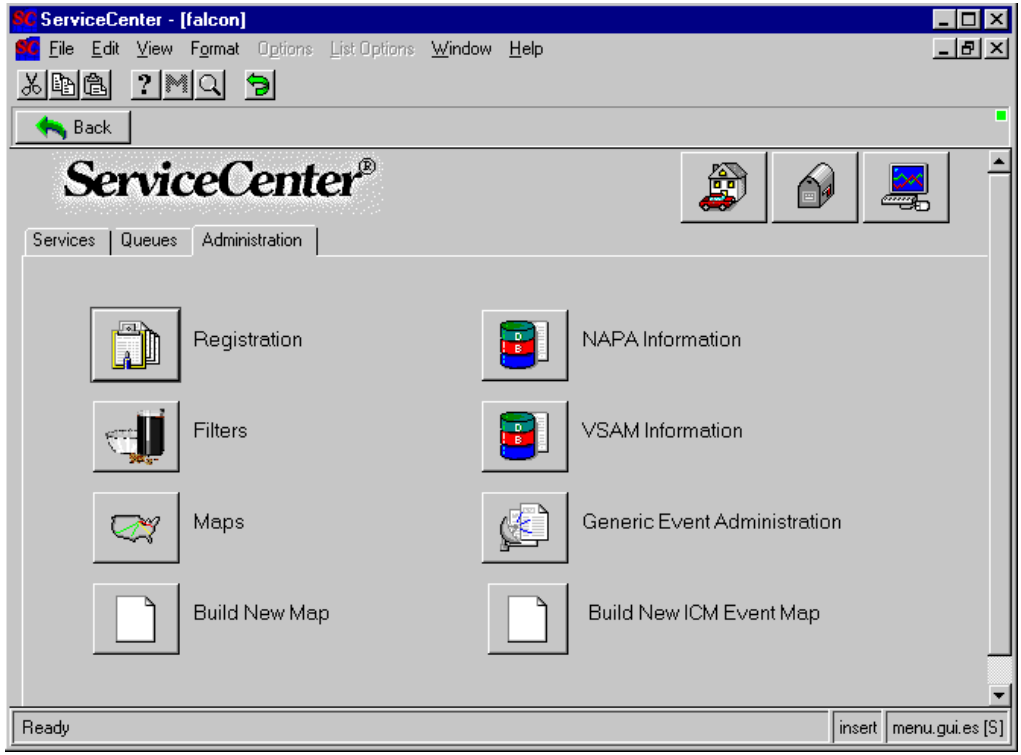
- ServiceCenter Event Services
- Events
- Tivoli to ServiceCenter mappings
- Event filtering
- ServiceCenter Problem Management

ServiceCenter Event Services

To access Event Services:

1 Click **Event Services** from the ServiceCenter main menu.

The Event Services menu is displayed.

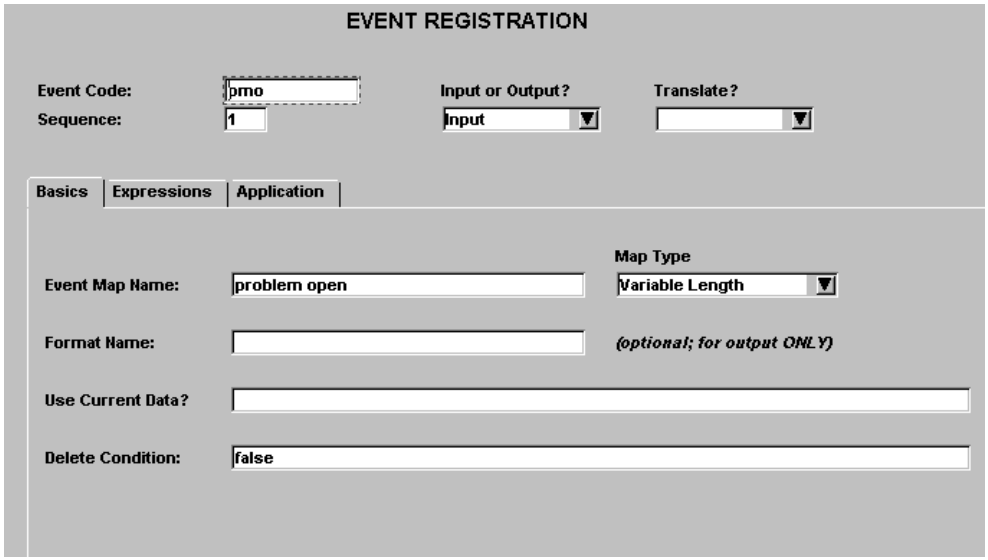


2 Click on the Administration tab.

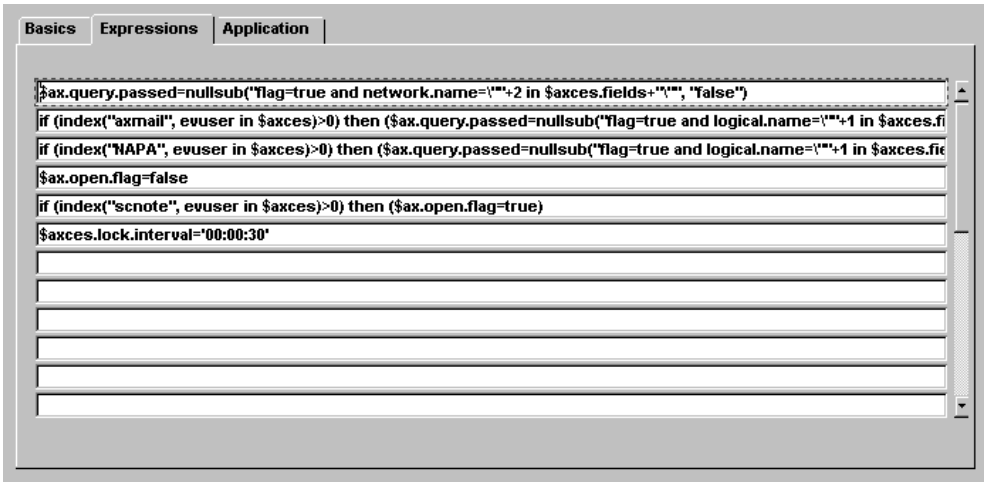
3 Click **Registration**, **Filters** or **Maps** to access these Event Services functions.

Event registration

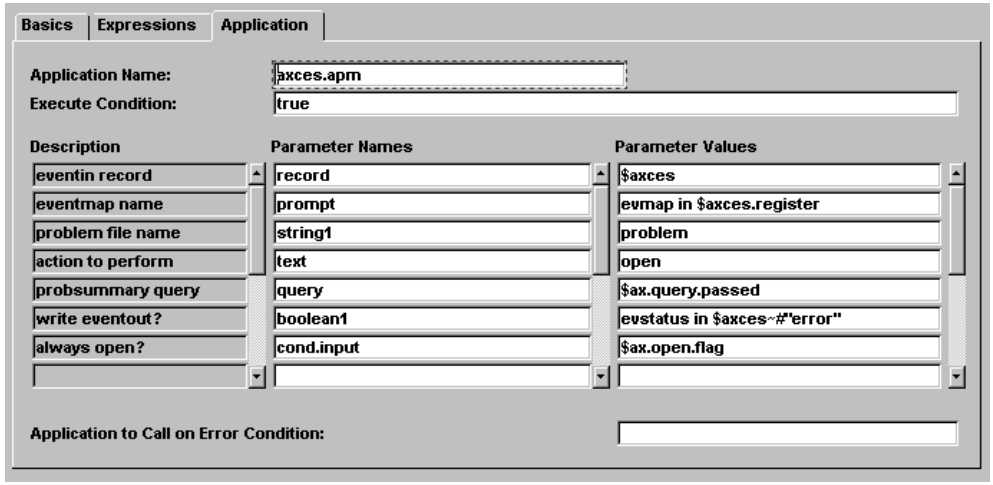
Event Registration forms establish the definitions for events processed by the system.



The *Expressions* tab displays the processing logic associated with the event type, e.g. *pmo*.

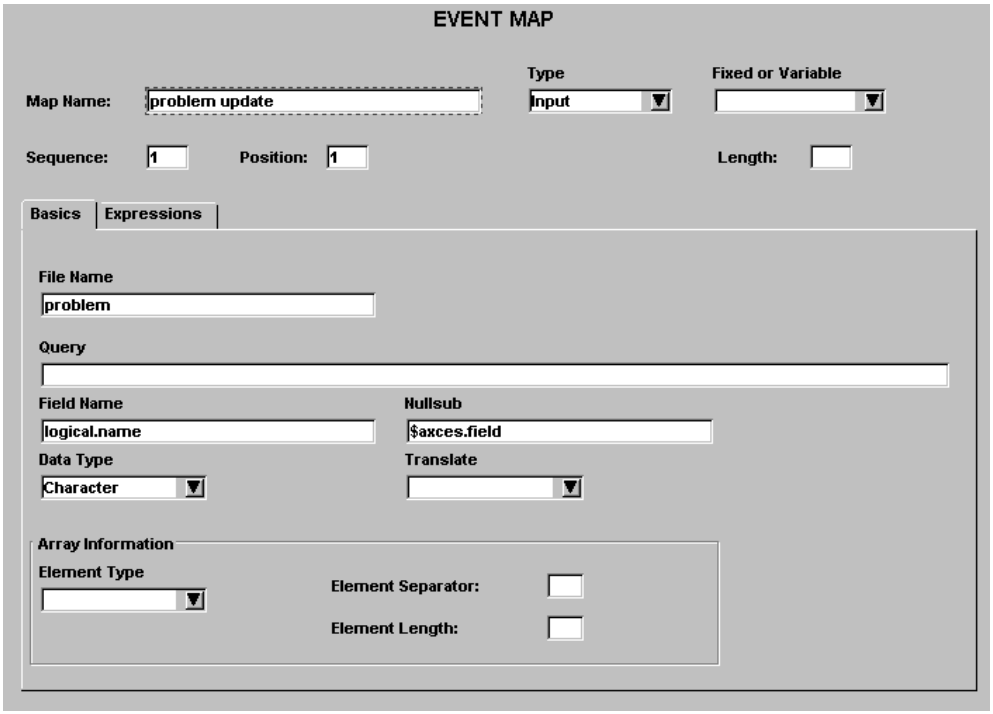


The *Applications* tab displays the RAD associated with the event, as well as the parameters used when running the application upon processing of the event.

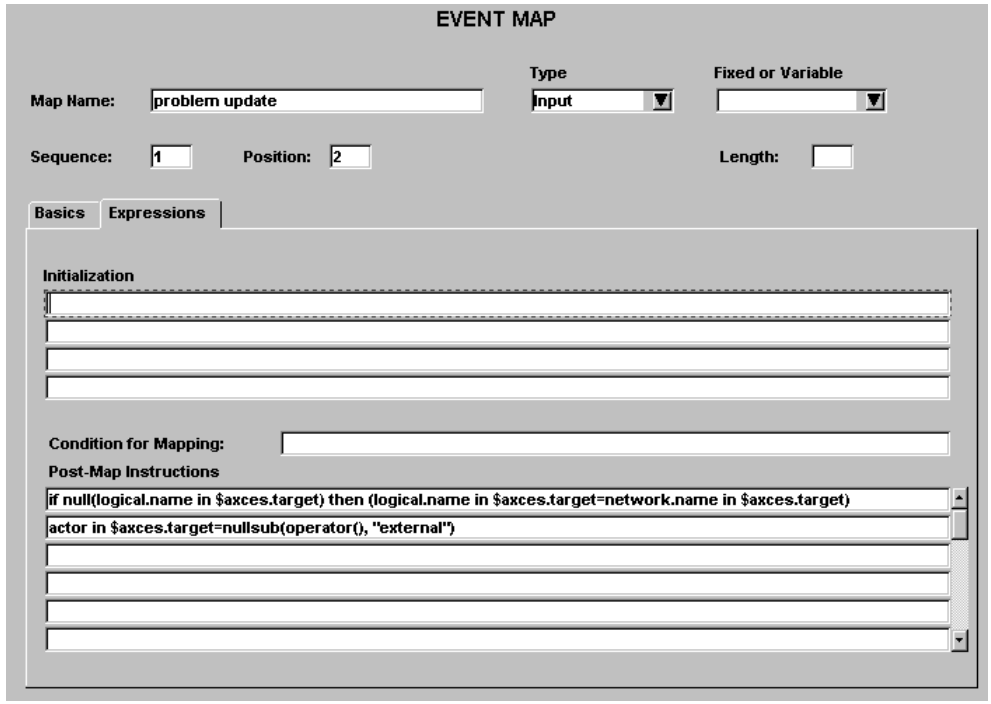


Event maps

Event Maps are the guides to recording and processing events, including the data type and the name of the file where event data is stored.



The *Expressions* tab reveals additional instructions for event processing.



All Event Services features are addressed in greater detail in the *Event Services User's Guide*.

Events

Event examples

The following examples provide likely Problem Management event scenarios and the communications which result between the Tivoli Plus module and ServiceCenter Event Services. These scenarios are examples supported by the sample rules and commands included in the ServiceCenter Plus module. Many combinations are available, and you can even tailor or create your own scripts for different interactions.

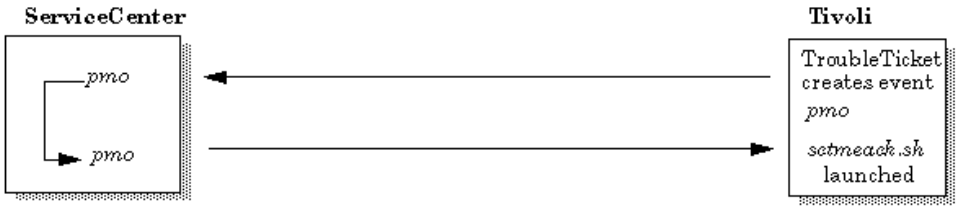
Tivoli-generated/SC managed

This scenario assumes that the problem ticket is opened by Tivoli and is completely managed by ServiceCenter HelpDesk.

Problem opened

Tivoli agents monitor various actions and conditions across the enterprise system. The Tivoli Plus module can be set in such a way to generate a problem ticket when certain data is received from an agent. This ticket event initiates the class (event) that, passing through SCAutomate, creates a *problem open* (*pmo*) event in ServiceCenter.

On the ServiceCenter side, once the *pmo* is created, another event is sent back to Tivoli (*ProblemOpened*), acknowledging the first event. Tivoli receives this message acknowledging the initiating event by launching the shell script *sctmeack.sh*, that records the status of the class (*event*) as problem opened. All future steps/changes that affect this problem ticket will originate on the ServiceCenter side.



Problem updated

A change or update to an existing problem/Trouble Ticket (for example, a change in problem severity) initially takes the form of a *problem updated* event (*pmu*) sent from ServiceCenter.

This message is received by Tivoli, where the Severity of the initiating event is changed by launching the *sctmesev.sh* shell script.

Problem closed

Closing an existing problem/Trouble Ticket from ServiceCenter takes the form of a *problem closed* event (*pmc*) on the SC side.

This event is received by Tivoli, where the class initiating event is closed and the shell script *sctmeclose.sh* is launched.

SC-generated/Tivoli managed

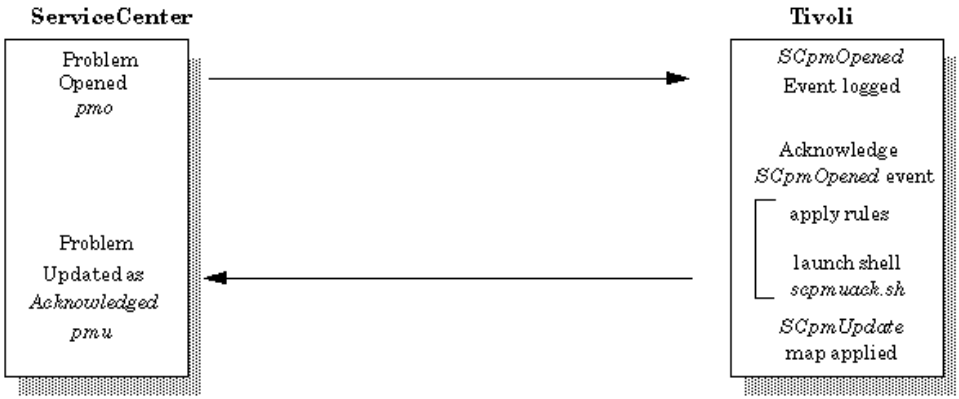
This scenario assumes that the ticket is opened by ServiceCenter and completely managed by Tivoli.

Problem opened

When a problem ticket is opened in ServiceCenter, a *pmo* event is created. When this event is transmitted to Tivoli, it is read as a class *SCpmOpened*. This event/class is logged using the *pmo* map.

Event *SCpmOpened* is acknowledged. According to the rules applied to the system, the shell *scpmuack.sh* is launched and the *SCpmUpdate* class is generated. The class is sent to ServiceCenter, indicating the problem opened (*pmo — SCpmOpened*) event was received by Tivoli.

ServiceCenter receives this message in the form of a *problem update* event (*pmu*), adding/updating the acknowledged data to the initial problem ticket.



Problem updated

In the event that the severity of the problem is changed, Tivoli applies the system rules and launches the shell script *scpmusev.sh*, passing a message to ServiceCenter through the *SCpmUpdate* map.

ServiceCenter receives this message in the form of a *problem update* event (*pmu*), that updates the severity of the initial problem ticket.

Problem closed

When a problem ticket is closed in Tivoli, the system rules are applied and the *scpmclose.sh* shell script is launched. A message is applied to the *SCpmClose* map and transmitted to ServiceCenter.

ServiceCenter receives this message in the form of a *problem closed* event (*pmc*), and closes the initial problem ticket.

ServiceCenter and Tivoli co-manage

This scenario assumes that tickets can be initiated from either system and sent to the other. Once a ticket is received and acknowledged, both systems will manage the ticket together by sending updates and closes to the other as appropriate.

Problem opened

A ticket event initiating the class (event) *SCpmOpen*, passes through SCAutomate, and creates a *problem open* event (*pmo*) in ServiceCenter.

-or-

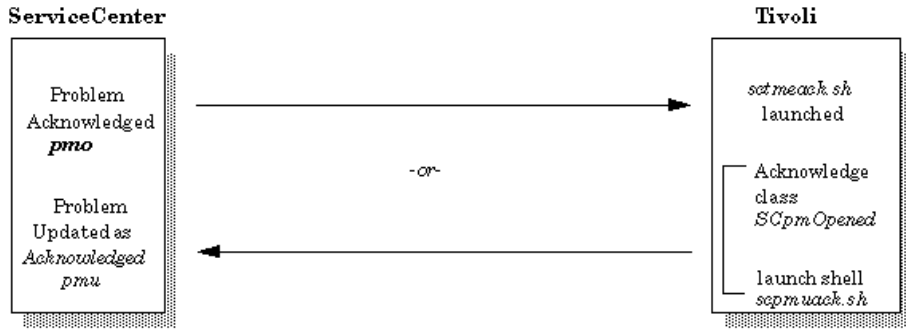
A problem can be opened in ServiceCenter and can pass a *pmo* event to Tivoli that is read as a class *SCpmOpened*. This event/class is logged using the *pmo* map. The *sctmeclose.sh* shell script is launched and a *SCpmOpen* class is created.

Problem updated

If desired, ServiceCenter can acknowledge the originating event by generating an output *pmo* event. This will then be mapped to perform an ACK of the originating event.

-or-

When Tivoli acknowledges the class *SCpmOpened*, the shell script *scpmuack.sh* is launched and the *SCpmUpdate* class is generated. The class is sent to ServiceCenter in the form of a *problem update* event (*pmu*), and ServiceCenter updates the initial problem ticket as acknowledged by Tivoli.

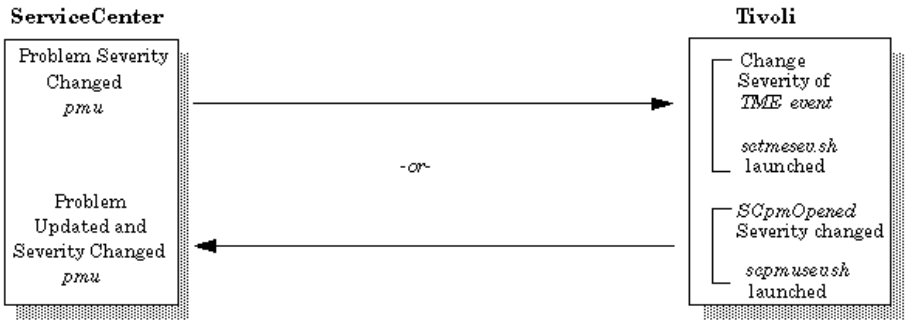


Similarly, when the severity of a ticket is changed, a notice of this alteration is sent to the originating system, and that system updates the ticket.

In ServiceCenter, a *pmu* event is created and then transmitted to Tivoli. When Tivoli receives this event, it is read as a change in ticket's severity and the shell script *sctmesev.sh* is launched.

-or-

In Tivoli, the severity of the class *SCpmOpened* is changed and the shell script *scmusev.sh* is launched, which sends a *pmu* event to ServiceCenter. ServiceCenter receives this event and updates the problem ticket.



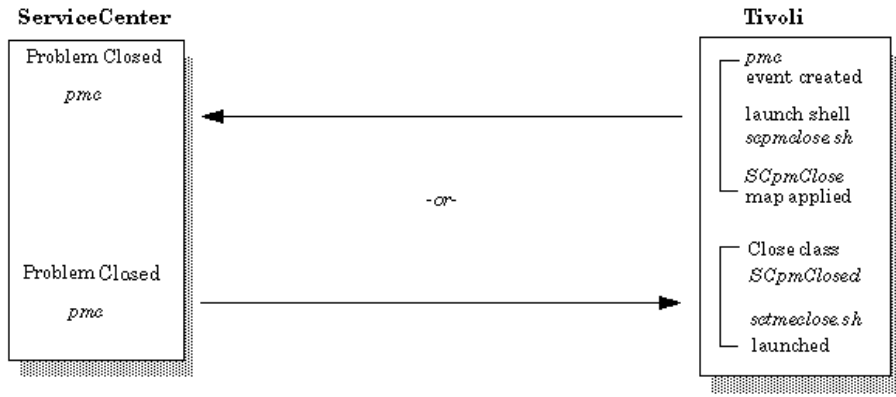
Problem closed

Closing a problem ticket affects both sides of the exchange.

When a ticket is closed on Tivoli, the shell script *scpmclose.sh* is launched and a message is sent to ServiceCenter through the *SCpmClose* map. ServiceCenter receives this message as a *problem close* event (*pmc*) and closes the initial ticket.

-or-

When a ticket is closed in ServiceCenter, a *problem close* event is created and passed to Tivoli, where the shell script *sctmeclose.sh* is launched.



Tivoli to ServiceCenter Mapping

Event mapping

Please review the following sections in the *Event Services Guide*:

- Event maps
- Input events
- Output events

The open interface provided utilizes mapping that relates ServiceCenter Event Service events to TEC events. The Plus module maps correlate to ServiceCenter maps as follows:

Tivoli input map names correspond to the TEC class events they are mapping (e.g. *SCpmOpen* class would map using *../maps/eventin/SCpmOpen*).The *pmo* input map in Event Services is matched with any map in *..maps/eventin* that specifies *pmo* as the *SCENTER_EVENT:* in their map (e.g. *maps/eventin/SCpmOpen*).

Tivoli output map names correspond to the ServiceCenter event type. The *pmo* output map in Event Services is matched to the *..maps/eventout/pmo* map in ServiceCenter Plus for Tivoli.

Note: The maps are positional, so it is helpful to print or display the ServiceCenter map fields when building your ServiceCenter/Tivoli maps.

It is recommended that the standard maps not be modified and that the new events be created for all customized implementations. As well as product installation verification, the standard events are utilized by other SCAutomate adapters and the standard ServiceCenter applications expect standard events. As long as you are mapping your TEC classes to standard ServiceCenter events, no application expertise should be required.

Customization of RAD applications and the creation of new ServiceCenter events may be done by Peregrine Systems Professional Services or your staff. A working knowledge of ServiceCenter Applications and Event Services is required for any major modifications.

Eventin maps

The eventin mapping files relate *event slot variables* which are specified in the TEC class *to ServiceCenter eventin maps*. You need to browse the *.baroc file* of the classes (events) you wish to use, or any reference material specifying the slot variables available for a particular event. All events contain the base slot variables for class EVENT. The base slot variables may be reviewed in the *TEC Event Integration Facility Guide* or *TEC Rule Builders Guide*. A DEFAULT map is provided which is used if no map is defined for the TEC event class. This map, unless modified, creates a *pmo* event (Problem Management Open) in ServiceCenter. It utilizes the EVENT base slot variables.

The mapping syntax is easily applied with slight variations between input and output mapping. The map fields syntax is as follows:

- The form SCENTER_EVENT: specifies the ServiceCenter event type to create.
- The fields represent environment variables set by the TEC.
- The fields must be in the order specified in the corresponding Event Services map.
- If environment variable is not set, or is invalid, a missing field indicator is substituted.
- The form (sepchar:....,....,...,....) is a list of variables to be placed in one field with a separator (separator character ^ is reserved).
- To concatenate with no sepchar use form (:.....,.....,.....,....)
- The form [literal] is used to place a constant in a field.
- The # in the first character is treated as a comment.
- The form N/A is used to indicate a missing field.

Sample eventin map

See the corresponding ServiceCenter *pmo* Eventin figure for map name correlation.

```
# Mapping for ServiceCener input Standard event pmo
#   (Problem Management Open)
# Refer to README for syntax
#
# TEC slot for SC
# PMO event pos'l parm
# --------------------
SCENTER_EVENT:pmo
hostname
origin
referenceNo
causeCode
msg
N/A
N/A
(@origin,sub_origin)
[TivoliEvent]
category
adapter_host
objid
version
model
serialNo
vendor
location
administrator
N/A
resolution
administrator
priorityCode
abendCode
(_:source,sub_source)
```

Eventout maps

The eventout mapping files relate *event slot variables* that are specified in the TEC class to *ServiceCenter eventout maps*. You need to browse the *.baroc file* of the classes (events) you wish to use, or any reference material specifying the slot variables available for a particular event. If an output map is not provided, the ServiceCenter event is dropped and no TEC event is created. The maps provided will map current ServiceCenter standard events to TEC classes that are also provided as part of the PLUS module. Slight or no customization should be required in the supplied output maps and classes.

The mapping syntax is easily applied with slight variations between input and output mapping. The map fields syntax is as follows:

- The mapping files are named after the type of the ServiceCenter event (e.g. a *pmo* event will use a file called *pmo* in this directory for mapping).

  The files consist of lines of the following (white space optional):

  slot name: value

  where the slot names are names of TEC event slots to be assigned values.
- A blank line, or a line starting with a # is ignored.
- TEC_CLASS is a special slot name that specifies the TEC event class to be created. This keyword may be specified with or without TEC_TASK (default None).
  - TEC_TASK is a special slot name that specifies a task to be executed
- The full path name should be specified. This special name can be specified with or without TEC_CLASS (default None).
- The values are expanded before being assigned to the slots, using a shell-like syntax:
  - $n expands to the n'th field in the ServiceCenter event.
  - $type expands to the event type string (somewhat redundant).
  - $user expands to the user that created the event.
  - $time expands to the event time string.

Sample eventout map

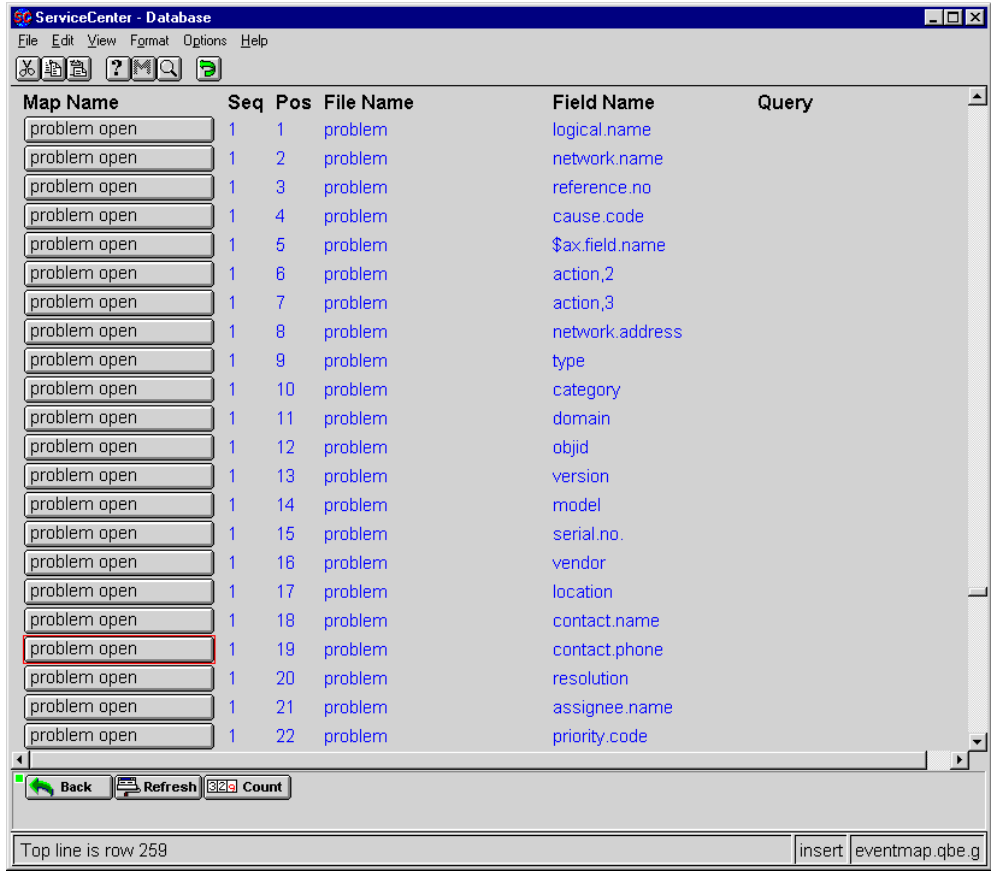
See the corresponding ServiceCenter *pmo* Eventout figure for map name correlation.

```
TEC_CLASS:SCpmOpened
msg:Problem $2 opened by $user.
hostname: $18
assigneeEmail:$1
number:$2
category:$3
openTime:$4
openBy:$5
actor:  $6
priorityCode:$7
severityCode:$8
assignment:$9
referedTo:$10
alertTime:$11
StaTus:$12
vendor:$13
referenceNo:$14
contactTime:$15
backupStart:$16
causeCode:$17
logicalName:$18
group: $19
jobName:$20
location:$21
version:$22
type:   $23
abendCode:$24
model:$25
action:$26
keywords:$27
referenceNo:$28
id:      $29
downtimeStart:$30
assigneeName:$31
contactName:$32
callerId:$33
```

contactPhone:$34
networkName:$35
openGroup:$36
resolution:$37
updateAction:$38
source: ServiceCenter

ServiceCenter event maps

To view the current ServiceCenter maps, select **Event Map** from the Event Services main menu.



Note: The ServiceCenter eventout *pmo* map has 38 fields, though only 28 are shown in the figure above.

Event Filtering

Filtering is required for ServiceCenter output events so that only those relevant to Tivoli are processed. For example; not all problem tickets are opened relevant to Tivoli TEC managed resources. Therefore, a simple shell script is provided which can be customized for your particular operational specifications. Event selection may be based on *evtype, evuser*, and *evfields* contents. The shell script is invoked on all outgoing events from ServiceCenter Event Services eventout file.

To block an event, a non zero return code is returned from the *../bin/evfilter.sh* script. The supplied filter example blocks all non-implemented event types (pager, fax, email, etc.). More filtering could be provided to look at category, assignment, etc. The filter should be adjusted to your operational requirements as a step in your installation.

Filtering on the input side can be accomplished using the TEC rules or ServiceCenter Event Services filtering discussed in the *Event Services Guide*.

evfilter.sh

The script shown below is a sample *eventout* filter script. It allows simple and complex filtering of events prior to creating an event in Tivoli. The four filtering parameters are the *evtype*, *evuser*, *evsepchar* and *evfields*. The default filter blocks the events listed.

An *exit 0* allows the event to be forwarded to Tivoli; an *exit 1* blocks the event.

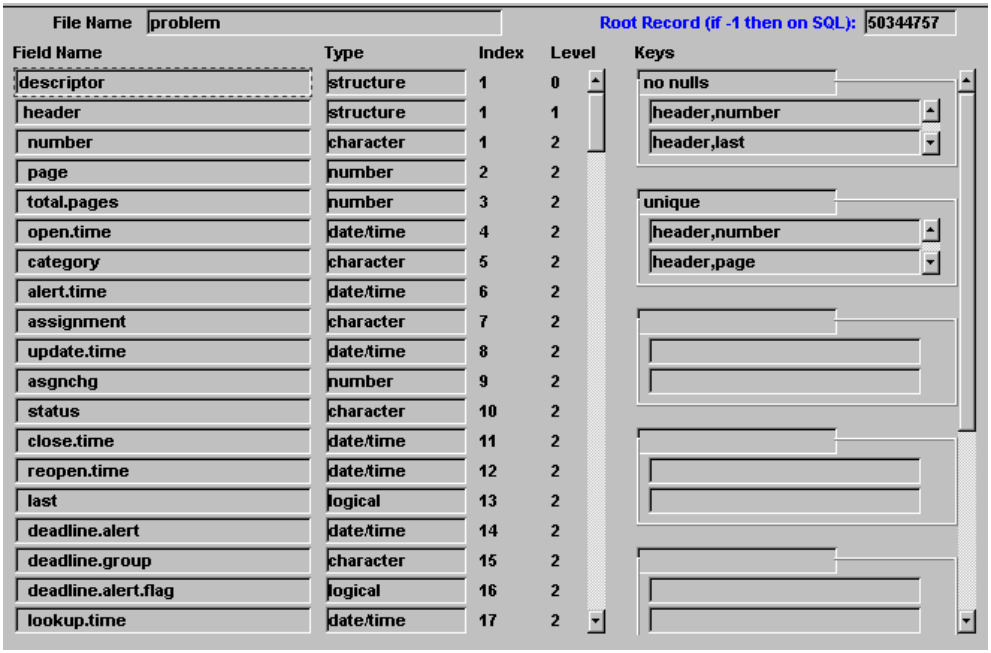
```
#
#
#############################################################
##############
EVTYPE=$1
EVUSER=$2
EVSEP=$3
EVFIELDS=$4
if [ -n "$EVTYPE" ]; then
  if [ $EVTYPE = "pager" -o  $EVTYPE = "email" ]; then
  exit 1
  fi
  if [ $EVTYPE = "fax" -o $EVTYPE = "submit" ]; then
```

```
  exit 1
  fi
  if [ $EVTYPE = "icma" -o $EVTYPE = "icmu" -o $EVTYPE = "icmd" ]; then
  exit 1
  fi
  if [ $EVTYPE = "cm3rin" -o $EVTYPE = "cm3rout" -o  $EVTYPE = "cm3ack" ]; then
  exit 1
  fi
  if [ $EVTYPE = "cm3tin" -o $EVTYPE = "cm3tout" ]; then
  exit 1
  fi
fi
exit 0
```

ServiceCenter Problem Management

Another area you may want to customize and configure to meet your needs in ServiceCenter is the definition of problem ticket formats and management. Based on your needs, you will likely need to create additional problem categories that reflect the automatic problem ticket processing supported by the SCPlus for Tivoli product. You can use the problem category named *example* as a guide or template to build new automated ticket categories and their associated forms and Database Dictionary definitions.

The following figure shows a portion of the Database Dictionary structure for the problem file. This can be used as reference for custom categories based on the problem file. Refer to the *Database Dictionary* section of the *Base Utilities Guide*.



Event data is channeled and presented in ServiceCenter according to controls defined in the *Format Control* records associated with specific problem category formats (display forms). Refer to the *Format Control Guide* for complete information on Format Control in ServiceCenter.

Additionally, the *Forms Designer* tool is used to create custom forms for added problem categories and customizations to accommodate the automated problem ticket generation from Tivoli (TEC) events. Refer to the *Forms Designer Guide* for more details on form (format) development and customization screens.

CHAPTER 5 Inventory Integration

This chapter describes inventory integration, which consists of three parts:

- Acquiring inventory data from Tivoli
- Providing a mapping mechanism to convert data acquired from Tivoli into ServiceCenter mapped data
- Sending mapped data to ServiceCenter

Acquiring Inventory Data from Tivoli

The current method of querying the Tivoli Inventory database relies on the Tivoli Query Libraries. This ensures that if Tivoli is able to access the data, so can the Plus Module. The method involves creating database views (if they are not part of the default query library), creating a Tivoli Query using the database view, and using the Query name in the mapping javascript to get the data out. Most of the time, the default database views created by the Tivoli Inventory are more than adequate in providing a very full set of data. This method not only allows for an easy Query creation using the Tivoli GUI Query builder, it also allows room for designing various ways to optimize data retrieval. For example, it allows the user to tailor the Query to retrieve all rows in a table and to group them by Endpoint in memory, resulting in only one database access, or design very specific database queries to get exactly the data requested. In the new Plus Module is a new task called *Install Default Inventory Queries*. By executing this task on the Policy Region of the Manage Node that does the integration, you will create the thirteen default queries used in the default operation of the integration.

The following is an example of how these queries are created in the shell script for the task:

```
wcrtquery -d "Query to be used by SC+Plus Hardware Inventory" \
 -r inventory -v INVENTORYDATA \
 -c TME_OBJECT_ID \
 -c TME_OBJECT_LABEL \
 -c HARDWARE_SYSTEM_ID \
 -c COMPUTER_ARCHITECTURE \
 -c COMPUTER_MODEL \
 -c PHYSICAL_MEMORY_KB \
 -c PAGING_SPACE_KB \
 -c COMPUTER_SCANTIME \
 -c PROCESSOR_MODEL \
 -c PROCESSOR_SPEED \
 -c BOOTED_OS_NAME \
 -c BOOTED_OS_VERSION \
 -x SC_Queries SCHardware
```

After these queries are created, they can be used by the mapping Javascripts. Internally, these queries are actually being used in a call to *wrunquery*. Therefore, the requirement for this Plus Module is Tivoli Inventory 3.2 and later because of the use of these newer Tivoli command line interfaces.

Mapping Data

The mapping of Tivoli Inventory data into ServiceCenter data is a flexible yet powerful feature. It is done using an active scripting language (Javascript) referencing passive data formats from Tivoli Queries and ServiceCenter static event map files. In addition to the standard Javascript methods, the scripting is also extended to allow for creation of Tivoli Query objects, ServiceCenter Event Objects, and a method for executing command line programs.

The simple process in a typical mapping script is as follows:

1. Create a Tivoli Query (TivoliQuery) object based on the Tivoli Query, and execute it to populate the object with data.
2. Create a ServiceCenter Event (SCEvent) object based on a static map file.
3. Assign variables in the SCEvent object using data accessed with the TivoliQuery object.
4. Generate and write the event to the *scevents* file.

Ultimately, you should have an Iterator query returning Endpoint names from the INVENTORYDATA view (used in a Tivoli Query). You will loop with the Endpoint values and assign values as you *iterate* through the rest of the TivoliQuery objects to SCEvent objects.

TivoliQuery object

The TivoliQuery object provides an object oriented view of a database table. Normally, the object is initiated given the equivalent Tivoli Query name, executed to fill it with data rows, and iterated, extracting the rows out sequentially.

Class Method	Description
TivoliQuery TivoliQuery(String queryName)	Object constructor passing query name of Tivoli Query. Example, to create a new TivoliQuery object *testQuery* from a Tivoli Query *TivoliQueryName*: testQuery = new TivoliQuery("TivoliQueryName");
void execute(void)	Executes the query and populate object with rows from the Tivoli Query. Example: testQuery.execute();

Class Method	Description
boolean hasMoreValues(void)	Returns true if object contains more values, false otherwise. This is a method used in iterating the object for testing to see if the object has exhausted all its values. Example: *while(testQuery.hasMoreValues()) // loops until testQuery has no more values* *{* *.... process values ...* *testQuery.next(); // see explanation below* *}*
void next(void)	Move the current row pointed to in the query to the next row. This is another utility method for using in an iteration for moving the object value to its next value. Example: *while(testQuery.hasMoreValues()) // see explanation above* *{* *.... process values ...* *testQuery.next(); // move the object to point to its next value* *}*
void reset(void)	Reset the current row pointer to the beginning. This method resets the object's value pointer to return the first one. This is generally used to reset the object after its values have been exhausted, to be used again. Example: *while(testQuery.hasMoreValues())* *{* *.... process values ...* *testQuery.next();* *}* *testQuery.reset(); // resets object so that it may be used later again* *// starting at its first value.*

Class Method	Description
String getCurrentValue(String fieldName)	Given the query field name, returns the corresponding value. The field names are identical to the names given in the original Tivoli Query. If a field name is given that does not exist in the Tivoli Query, the string value FIELD_NAME_NOT_FOUND is returned instead. If you call this method and there are no more rows/values from this object, you will get NO_MORE_VALUES returned as a String instead. Example: *while(testQuery.hasMoreValues())* *{* *var testVar = testQuery.getCurrentvalue("TestFieldName");* *testQuery.next();* *}* *testQuery.reset();*
void setWhereClause(String whereClause)	Modify the query and add the where clause. This method is used to add a where clause to the Tivoli Query on the fly. Example: testQuery.setWhereClause("CONFIG_CHANGE_TIME '" + timeVar + "'");
int getNumRows(void)	Gets the number of row values contained in a TivoliQuery object. Note: a newly created TivoliQuery object will be empty until it has called the execute() method and retrieved the database rows. Example: *testQuery = new TivoliQuery("Test");* *testQuery.execute();* *var numRows = testQuery.getNumRows(); // variable will contain // the number of rows*

SCEvent object

The SCEvent is the ServiceCenter event object that can generate the appropriate formats understood by the event monitoring process *scevmon*. This object provides methods of setting, getting its field values by name, and generating the appropriately formatted event string to a file.

Class Method	Description
SCEvent SCEvent(String evType, String mapFile)	Object constructor takes the evtype *ICMserver* and the path to the static map file that defines the slot names in the ServiceCenter Event Registration Event Map. In the future, this map file can be automatically generated. Example: icmaEvent = new SCEvent("ICMserver","maps\\inventory\\ICMserver.map");
void setEvField(String fieldName, String fieldValue)	Sets the corresponding field name to the corresponding field value. The fieldName that is specified as the argument must exist in the map file that the object is instantiated with. All field values are treated as String, so you might want to convert it to String before passing it as the argument. Make sure that the rules in ServiceCenter Event Registration maps are met such that required fields are always set with a valid value. Example: *icmaEvent.setEvField("type", "server");*
String getEvField(String fieldName)	Gets the field value corresponding to the specified fieldName argument. Returns a null String if the fieldName is not found. This method is rarely used because once the object is set with the field values it is seldom needed to extract its field values back out. Example: var fieldValue = icmaEvent.getEvField("type");
void writeEventString(String eventFilePath)	Given the file path to *scevents* will generate the proper event string and write it to *scevents*. This is the last step in the life cycle of an SCEvent object. From this point, the event monitor *scevmon* will pick up the generated event string and forward it to ServiceCenter. Example: icmaEvent.writeEventString("scevents");

SCDiscover object

SCDiscovery is the main object that the Javascript parsing engine is embedded in. It does not need to be instantiated and only has one relevant static method.

Class Method	Description
void run(String commandLine)	Executes a specified command line and returns a string of the output.

Sending Data to ServiceCenter

The mapped and formatted data string consisting of an SCAutomate Event Header and data in ^ (hat) delimited format is appended to the *scevents* file in the $SCPLUSHOME. This allows the *scevmon* event monitor to pick up the event when it is ready and forward it to the ServiceCenter *eventin* file. The event types accepted are the default *icma* and *prgma* events as well as the newer *ICMserver* and *ICMworkstation* types. There is no restriction as to the type of event being forwarded as long as the type is defined in the targeted ServiceCenter. A ServiceCenter *unload* file is provided named *icmNew.unl* on the root directory of the Plus Module CD. This file will create the relevent *ICMserver* and *ICMworkstation* format control, as well as RAD applications to support it.

System Architecture

The SCPlus Tivoli Inventory integration consists of a Java 1.1.6 application embedded with a Javascript Interpreter (FESI version 1.1.1) as well as the event monitor *scevmon* that comes with the product. The output of the Tivoli Inventory data is translated into ServiceCenter-compatible events and placed in the *events* queue file to be picked up by the event monitor *scevmon*. By default, the generated event types are *icmServer* and *icmWorkstation*. These are not event types that are shipped with ServiceCenter before version 3. Therefore they do not require the loading of an unload file *icmNew.unl* from the CD ROM.

The system is a *pull* type of integration, meaning it is not driven by any events in the Tivoli TME 10 environment. Rather, the operator issues an *add* or *update* command from the Launch icon of the Plus Module to initiate the process. However, it is feasible to set up a scheduled Tivoli Task to do periodic synchronization of the inventory data.

The files installed for this implementation. Their function is as follows:

Note: The files are installed into the directory where the plus module was installed (*/usr/SCPlus* by default). Therefore, they are shown as relative to this path.

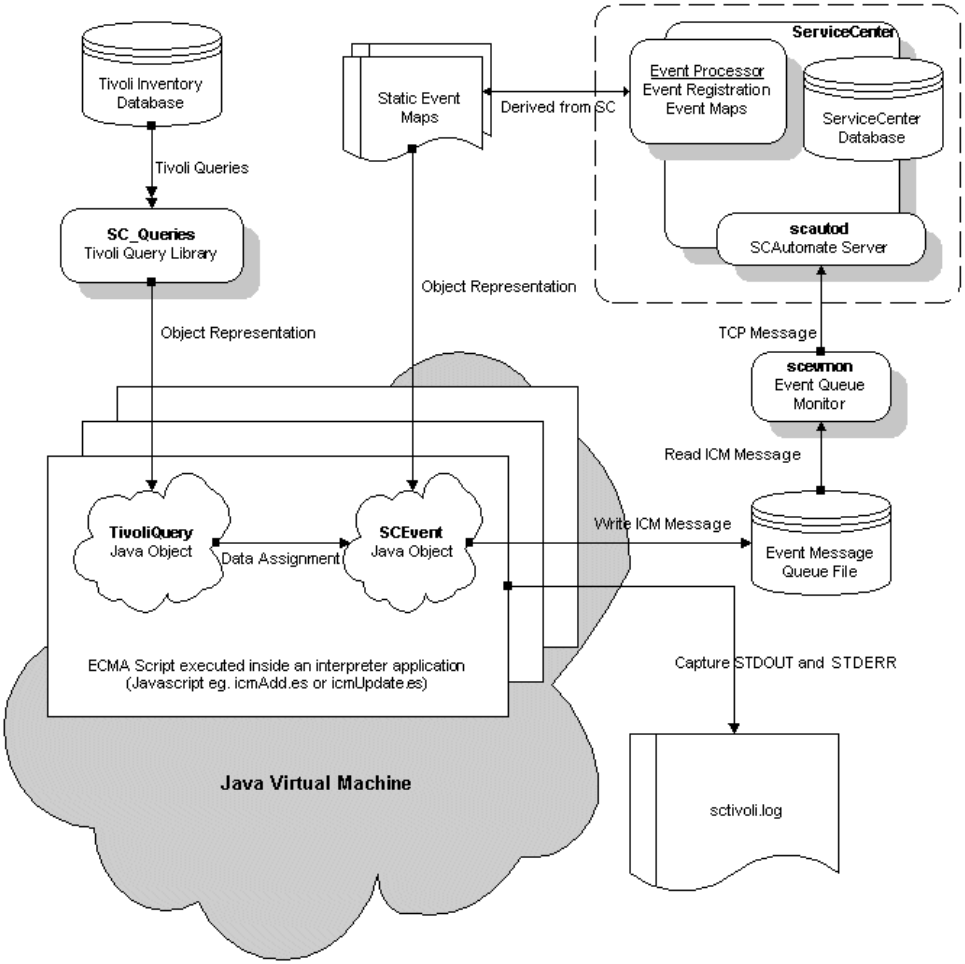
Files	Description
sctivoli.log	This is the log file that will show the progress of the integration. By default, standard output and standard error output is captured to this file. Therefore, to output from the mapping Javascript to be logged by this file, use the write() and writeln() methods in your script.
scevents	This file contains event messages that are to be, and have been sent to ServiceCenter. The accompanying *syncfile* file (see below) keeps track of the last event sent as well as the last event retrieved from ServiceCenter. The messages are stored in native '^' delimited format. They are in chronological order, with the latest at the bottom.
syncfile	This file contains an identical entry of the last event message read from the *scevents* file as well as a sequence number. The identical entry is used to compare with the *scevents* file to see which message was last read. The sequence number is used to mark the last sequence number of the event message retrieved from the ServiceCenter *eventout* file.

Files	Description
bin/scevmon(.exe)	This executable is a persistent process that reads the *scevents* file and updates the *syncfile*. It communicates with the ServiceCenter *scautod* SCAutomate daemon process. If this process is taken down, there will be no messages going to or coming from ServiceCenter.
lib/scdiscover.add.sh	This shell script gets invoked from the Launch icon menu for *Add new nodes to ServiceCenter Inventory*. This script, in turn, invokes the jre 1.1.6 Java Runtime Environment to execute a Java Application that drives the integration process. This script can also be executed by itself or scheduled as a task.
lib/scdiscover.upd.sh	This shell script gets invoked from the Launch icon menu for *Update nodes in ServiceCenter Inventory*. This script, in turn, invokes the jre 1.1.6 Java Runtime Environment to execute a Java Application that drives the integration process. This script can also be executed by itself or scheduled as a task.
maps/inventory/icmAdd.es	This is the Javascript (ECMA script) that configures the logic and maps to use for generating inventory event messages from Tivoli Inventory data. By default, it references the *ICMServer.map* and the *ICMWorkstation.map* static map files to generate the *ICMServer* and *ICMWorkstation* event types. This is a script that will treat all inventory data as new and will generate events from them.
maps/inventory/icmUpdate.es	This is the Javascript (ECMA script) that configures the logic and maps to use for generating inventory event messages from Tivoli Inventory data. By default, it references the *ICMServer.map* and the *ICMWorkstation.map* static map files to generate the *ICMServer* and *ICMWorkstation* event types. This is a script that will incrementally create event messages only for the nodes that have been scanned.
maps/inventory/ICMServer.map	This is a static map file that corresponds positionally (top down) to the Event Registration map for the *ICMServer* event type. To get this event type in versions preceeding ServiceCenter 3, you must load the unload file *icmNew.unl* from the CD ROM.
maps/inventory/ ICMWorkstation.map	This is a static map file that corresponds positionally (top down) to the Event Registration map for the *ICMWorkstation* event type. To get this event type in versions preceeding ServiceCenter 3, you must load the unload file *icmNew.unl* from the CD ROM.
jre1.1.6/bin/jre(.exe)	This Java Runtime Environment is used to execute the SCDiscover Java application.
jre1.1.6/lib/SCDiscover.jar	This Java Jar (Java Archive) file that contains all the classes needed to implement the TivoliQuery, SCEvent, and SCDiscover objects.

Files	Description
jre1.1.6/lib/fesi.jar	The Java Jar file contains implementation for an embeddable Javascript/ECMA-262 interpreter used by the application.
jre1.1.6/lib/classes.zip	These are platform specific classes that archive for JRE1.1.6.

System Diagram

The following diagram illustrates the overall system and how it functions with Tivoli Inventory and ServiceCenter.



Installation

To install ServiceCenterPlus for Tivoli:

1. Install the ServiceCenter for Tivoli Plus Module 1.2.a onto the managed node via the Tivoli Desktop.
2. Execute the second step installation task within the Plus Module.
3. Generate the default query library (SC_Queries) and all its queries by executing the *Install ServiceCenter Default Query Libraries* task. If you are connecting with a ServiceCenter version less than 3.0, you will want to load the unload file *icmNew.unl* from the CD ROM to get the *icmServer* and *icmWorkstation* event types. This unload file is for ServiceCenter application level A9802 (A9901 and later include these enhancements).

Operation

After installing, you may want to initially gather all the inventory data. You may do this by executing the menu option *Add new nodes to ServiceCenter Inventory* on the Launch icon of the Plus Module. This will execute the script *scdiscover.add.sh* and send *icm* event messages to the events queue file. If you start the event monitor now, by choosing the menu option *Start TME <- SC event processor* on the Launch icon of the Plus Module, the event monitor "scevmon" will pick up the event messages from the queue file and start forwarding them to ServiceCenter to be processed.

It is suggested that after you have initially populated the ServiceCenter database, you will want to periodically *update* it by choosing the menu option *Update nodes in ServiceCenter Inventory*. This task will execute the shell script *scdiscover.upd.sh* and query the Tivoli Inventory database for modified items to be updated (see section titled *System Configuration* on page 105 for details on how to modify the basic configuration). Alternatively, you may create a scheduled task that executes the same shell script to periodically pick up modified nodes.

System Configuration

The system is configured by customizing:

- Tivoli tables and views
- Tivoli queries
- ECMA script
- Static event maps

The default behavior when you *add new nodes* or execute the *scdiscover.add.sh* script is for the system to iterate through the INVENTORYDATA view for all the endpoints and create *icmServer* messages when it is a TMF_Managed_Node and *icmWorkstation* messages. Otherwise, the default behavior when you *update nodes* or execute the *scdiscover.upd.sh* script is for the system to do the following:

1. Determine the last maximum scantime (maximum of COMPUTER_SCANTIME in the INVENTORYDATA view). If none is found, use the arbitrary date 1900-01-01 12:00:00.
2. Query for the current maximum of COMPUTER_SCANTIME in the INVENTORYDATA view and save that to be used in step 1 for next time.
3. Using the last maximum scantime (from step 1), set the where clause for the INVENTORYDATA view to only return endpoints that have COMPUTER_SCANTIME greater than this. This is used as the Iterator for looping the rest of the subqueries for data, thereby effectively only querying for data on endpoints that have been scanned after we last updated the ServiceCenter database.
4. Using the last maximum scantime (from step 1) set up the where clauses for the rest of the subqueries (whenever there is available a CONFIG_CHANGE_TIME field) to only return data that have CONFIG_CHANGE_TIME greater than this.
5. Create and send event messages only when data has been detected as changed (from step 4).

Customizing Tivoli Tables, Views, and Queries

You may choose to create new tables or views in the Tivoli Inventory database to track additional information not available in the default system. The source of this new information may use the customizable *useradd.mif* facility provided by Tivoli Inventory.

To create new tables or views:

1. Create the new tables or views using the appropriate SQL database clients.
2. Customize the Inventory Profile and *useradd.mif* files to populate the tables during a scan.
3. Create a Tivoli Query in the *SC_Queries* query library, specifying the attributes of the table that you wish to bring over to ServiceCenter.
4. Modify the ECMA script that does the integration (*icmAdd.es* or *icmUpdate.es*) to create TivoliQuery objects from the newly created Tivoli Query. The data is now available to you in the form of a TivoliQuery object.
5. Programmatically assign the TivoliQuery object's retrieved data to the SCEvent object in the script. If you are expanding the event type in ServiceCenter to accomodate these new data items, you will also have to modify the *Static Event Maps* that the SCEvent objects are built upon to reflect the changes.

Customizing ECMA Scripts

By default, the system interprets the *icmAdd.es* script for adding new inventory, and the *icmUpdate.es* script for updating it. The scripts are standard ECMA-262 conformant, and is compatible with most Javascript constructs. Information on ECMA-262 can be found in the CD ROM in the file *ecma-262.pdf*. Javascript syntax and function library can also be located at the netscape web site:

http://developer.netscape.com/docs/manuals/js/core/jsguide/index.htm

The basic functional format of the ECMA script is described in the following sections.

Note: Whenever there is a *write()* or *writeln()* method being called in the scripts, output is being captured to the *sctivoli.log* log file.

Declaring packages that will be used later

This section establishes aliases to packages in Java as well as object classes that may be used as shortcuts later in the script. They take the form of:

```
TivoliQuery = Packages.TivoliQuery.TivoliQuery;
SCEvent = Packages.SCEvent.SCEvent;
SCDiscover = Packages.SCDiscover.Main;
```

This means that later on, the access *Packages.SCEvent.SCEvent*, you only need to specify *SCEvent*.

Creating TivoliQuery objects

TivoliQuery objects are objected-oriented representations of Tivoli Queries. They contain structure information as well as data (after calling execute() method). They are the basis of loops and data extraction. For the main loop, an Iterator query object is constructed, as well as the subquery objects that return the data to be transferred. To create an Iterator TivoliQuery object based on the *SCIterator* Tivoli Query and execute it to fill it with data:

```
Iterator = new TivoliQuery("SCIterator");
Iterator.execute();
```

To create a TivoliQuery object *hardwareQuery* from its equivalent Tivoli Query *SCHardware* and execute it to fill it with data:

```
hardwareQuery = new TivoliQuery("SCHardware");
hardwareQuery.execute();
```

Loops

Loops of all kinds are being used in the ECMA script to iterate through rows of data and pick and choose the ones needed to create events from. Typically, it is a *while* loop, using the *hasMoreValues()* object method to test for continuation, the *next()* object method to advance to the next data row of values, and the *reset()* object method to reset the object to point back at the first entry. So it has this form:

```
while(queryObject.hasMoreValues())
{
   ....
   .... process data ....
   ....
   queryObject.next();
}
queryObject.reset();
```

Creating SCEvent objects

SCEvent objects derive their internal structure from Static Event Maps that document event registration maps on ServiceCenter. They are the place holders of formatted/converted inventory data to be sent to ServiceCenter. To construct a new SCEvent object of event type *ICMserver* based on the static map *maps\inventory\ICMserver.map* and set its *type* field to *server*:

```
icmaEvent = new SCEvent("ICMserver",
"maps\\inventory\\ICMserver.map");icmaEvent.setEvField("type",
"server");
```

Getting data out of TivoliQuery objects and into SCEvent objects

After you have created and filled TivoliQuery objects with data, retrieve data from them so that you can assign them to SCEvent objects. For example, to get the COMPUTER_ARCHITECTURE from the hardwareQuery TivoliQuery object, enter the command:

```
hardwareQuery.getCurrentValue("COMPUTER_ARCHITECTURE");
```

And, to assign it to the *vendor* field of the SCEvent object, you can enter the command (in one line):

```
icmaEvent.setEvField("vendor",
hardwareQuery.getCurrentValue("COMPUTER_ARCHITECTURE"));
```

Writing the SCEvent object out to the queue file

After you have gotten the data in the format that you want, and you have satisfied all the requirements of the event type (e.g., required fields in your SCEvent object), write it out to the queue file to be forwarded to ServiceCenter by the *scevmon* process:

```
icmaEvent.writeEventString("scevents");
```

Customizing Static Event Maps

Static Event Maps are being used to create SCEvent objects so that each field in the event can be named.

Static Event Maps define positionally what ServiceCenter is expecting in a field delimited format from any SCAutomate adapter products. It is an *ascii* file created mirroring the position of each field of the event type you are integrating on each line. Therefore, if the first position in the event map for the *ICMserver* event type is *logical.name*, then the first line in its equivalent static map file will have the line *logical.name*, and so forth.

6 Operation

CHAPTER

This chapter introduces the various utilities associated with the ServiceCenter Plus for Tivoli TEC module. Configuration of each piece of the application is accomplished through a pop-up screen, which appears when you click on an icon. Input fields and parameters for each piece are identified and explained in the following material. A selected subscriber should be set prior to execution of any job, and the *Define Target ServiceCenter Server* task must be run once on any subscriber containing a ServiceCenter image.

The following topics are covered:

- ServiceCenter configuration and administration
  - Define target ServiceCenter server
  - Start ServiceCenter server
  - Shutdown ServiceCenter server
  - ServiceCenter server locks
  - ServiceCenter Server shared memory
  - Add ServiceCenter operator
  - Update ServiceCenter operator
  - Delete ServiceCenter operator
  - ServiceCenter license report

- ServiceCenter server semaphores
- ServiceCenter system bulletins
- ServiceCenter publish/subscribe information
- ServiceCenter server status

- Starting and stopping a ServiceCenterPlus for Tivoli session
  - Start TME <--> SC event processor
  - Stop TME <--> SC event processor
  - ServiceCenter client
  - Add new node to ServiceCenter inventory
  - Update nodes in ServiceCenter inventory
  - Archive processed events

ServiceCenter Configuration

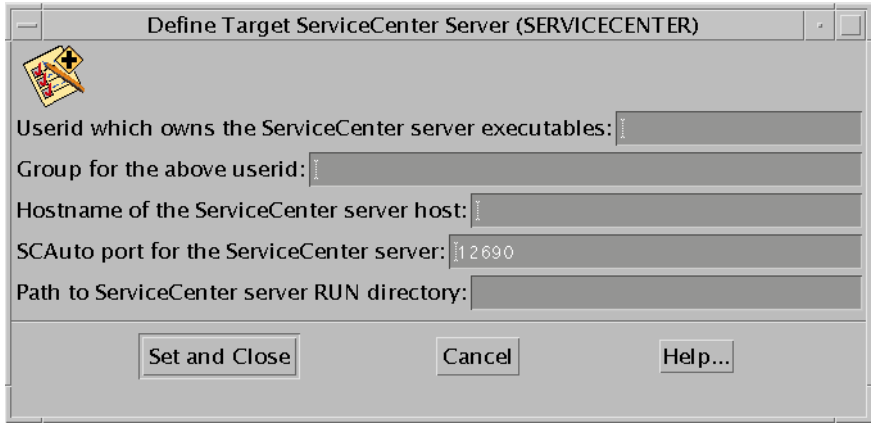
Define target ServiceCenter server

The Define Target ServiceCenter Server screen allows you to provide the login ID and ServiceCenter installation path information for any selected subscriber containing a ServiceCenter image. The task must be run once on the subscriber prior to the execution of other ServiceCenter tasks (ScStart, ScStatus, etc.). The *Define Target ServiceCenter Server* task would only have to be rerun on a subsciber if the user or install paths are changed. The direction for the SC Client must be set up here before the client can be executed.

To define the target ServiceCenter server:

1 Locate and activate the **Define Target ServiceCenter Server** icon.

The Define Target ServiceCenter Server configuration screen is displayed.



2 Complete the fields in this screen as follows:

Userid which owns...

Identifies the specific userid and operator profile to be used when accessing the ServiceCenter server on the selected subscriber. Use the name you specified in the set up procedure.

Group for the above userid

Specifies the Tivoli User group to which the user above belongs.

Hostname of the ServiceCenter ...

Specifies the hostname of the server where ServiceCenter is currently running.

SCAuto port...

Identifies the specific port used for connection to the current running instance of ServiceCenter on the above named host.

Path to ServiceCenter Server ...

Specifies the pathname where the ServiceCenter RUN directory and binaries can be reached on the selected subscriber system, *e.g. .../sc/RUN* (on Windows NT, a drive letter may be specified as well (e.g., *c:/sc/RUN*).

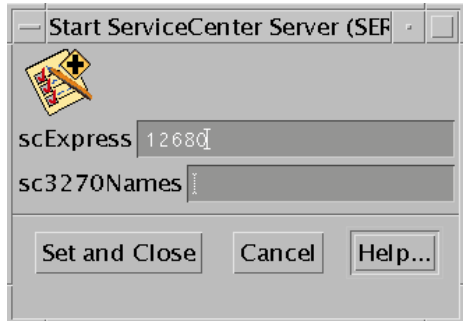
3 Click **Set and Close** to confirm the new configuration and exit the screen.

Start ServiceCenter server

This task starts the ServiceCenter server and related processes on the selected subscriber.

1 Locate and activate the **Start ServiceCenter Server** icon.

The Start ServiceCenter Server configuration screen is displayed.



2 Complete the fields in this screen as follows:

scExpress (service name or port)

ServiceCenter Express ports are usually specified as *12680*. Leave this field blank if you are not using this type of client.

sc3270Names (APPC or TCP/IP connections)

Supply the port name or number associated with this type of port, if you plan to use this type of client. Leave this field blank if you are not using this type of client.

Note: Both fields are optional depending on whether your system currently does or will support these types of clients. If you do not intend to use these types of SC clients, leave these fields blank.

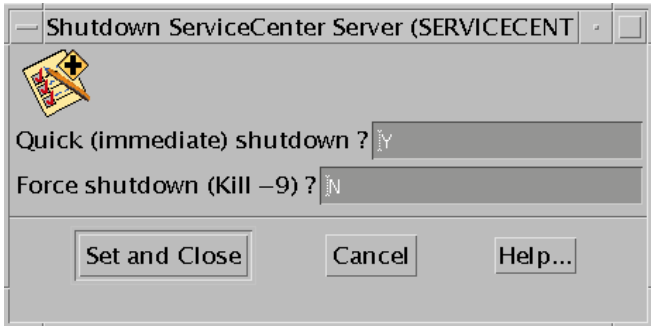
3 Click **Set and Close** to confirm configuration and exit the screen.

Shutdown ServiceCenter server

This task will stop ServiceCenter server and related processes on the selected subscriber.

1 Locate and activate the **Shutdown ServiceCenter Server** icon.

The Shutdown ServiceCenter Server configuration screen is displayed.



2 Complete the fields in this screen as follows:

scQuick

(Yes {default}| No)

scForce

(Yes|No {*default*})

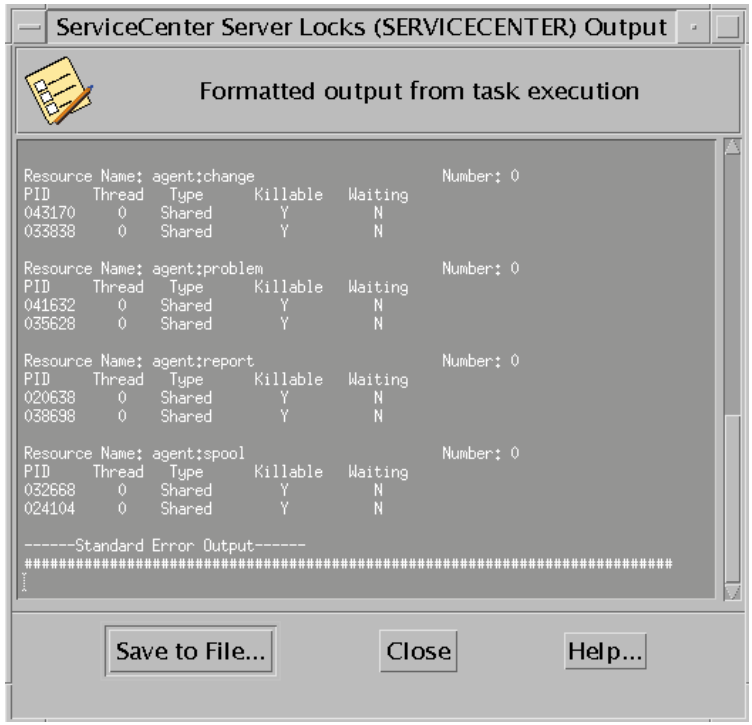
3 Click on **Set and Close** to confirm configuration and exit the screen.

ServiceCenter server locks

This task will display all current ServiceCenter locks on the selected subscriber.

1 Locate and activate the **ServiceCenter Server Locks** icon.

The ServiceCenter Server Locks configuration output text is displayed.



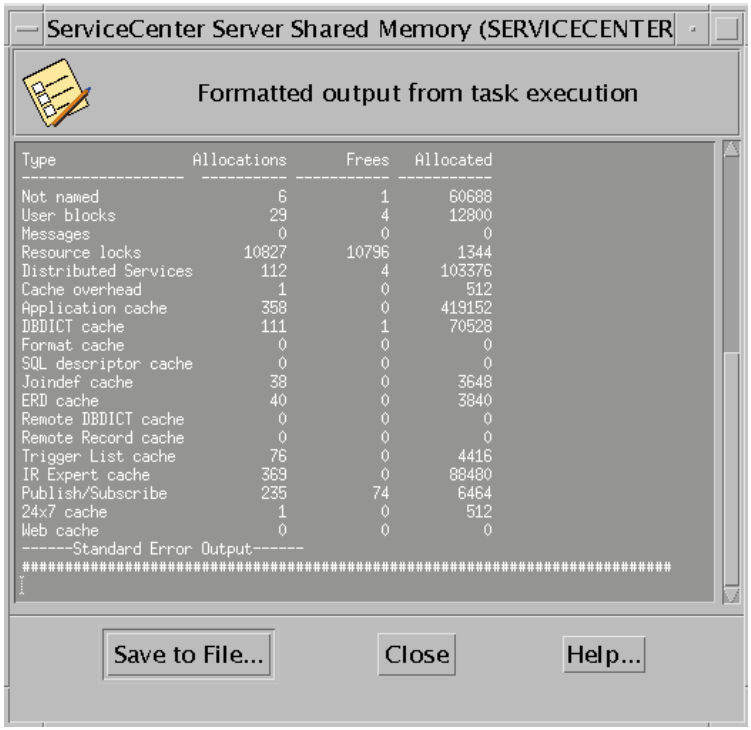
2 Click **Save to File** to output data, or **Close** to exit the screen.

ServiceCenter server shared memory

Use this task to display all ServiceCenter shared memory usage on the selected subscriber server.

1 Locate and activate the **ServiceCenter Server Shared Memory** icon.

The ServiceCenter Server Shared Memory configuration output text is displayed.



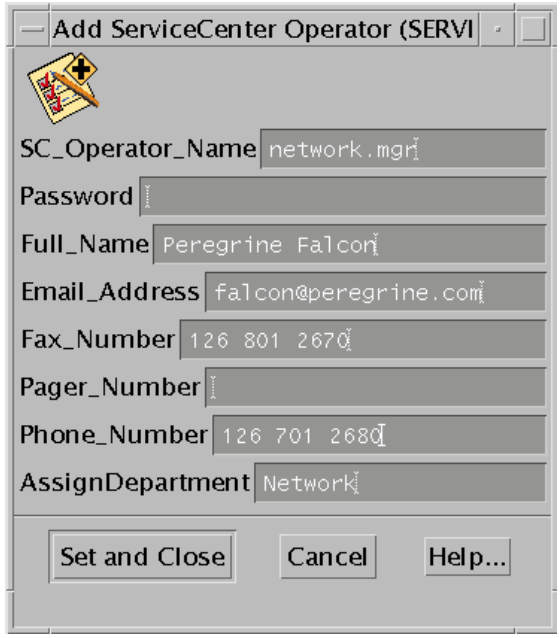
2 Click **Save to File...** to save output, or **Close** to exit the screen.

Add ServiceCenter operator

The Add ServiceCenter Operator task allows you to create an operator record in ServiceCenter. The supplied information is merged with a DEFAULT operator record in ServiceCenter to obtain capabilities and other information to complete the record.

1 Locate and activate the **Add ServiceCenter Operator** icon.

The Add ServiceCenter Operator configuration screen is displayed.



2 Complete the fields in this screen as follows:

SC_Operator_Name

ServiceCenter operator name (e.g., *network.mgr or system.mgr*).

Password

Specific password identifying the user.

Full_Name

Operator's long form name.

Email_Address

Email address for user/operator.

Fax_Number

Fax number for ServiceCenter or operator contact.

Pager_Number

Pager number for SCAutomate interface.

Phone_Number

Ground line telephone number for operator contact.

AssignDepartment

Group/dept. operator belongs to.

3 Click **Set and Close** to confirm output.

An Output screen is displayed that indicates the SC operator has been created.

Update ServiceCenter operator

This task will update a ServiceCenter Operator record in ServiceCenter.

1 Locate and activate the **Update ServiceCenter Operator** icon.

The Update ServiceCenter Operator configuration screen is displayed.

2 Complete the fields in this screen as follows:

SCName

ServiceCenter operator name (e.g., *system.mgr*, *network.mgr*).

Password

Specific password identifying the user.

FullName

Operator's long form name.

EmailAdd

Email address for user/operator.

FaxNumber

Fax number for ServiceCenter or operator contact.

PagerNumber

Pager number for SCAutomate interface.

PhoneNumber

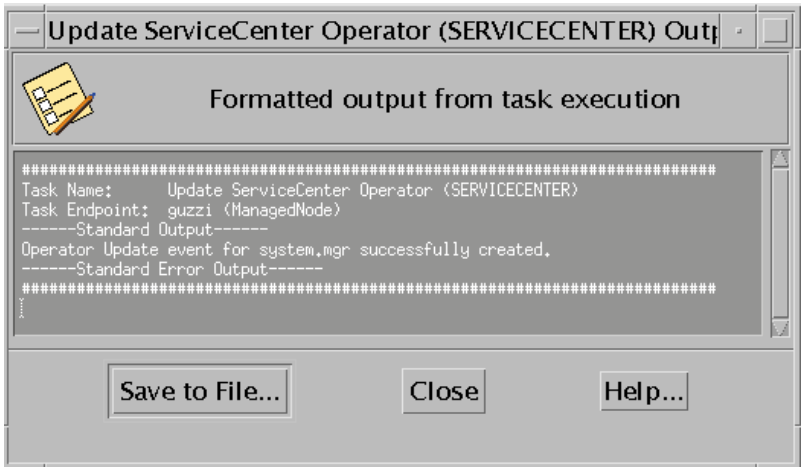
Ground line telephone number for operator contact.

AssignDepartment

Group/dept. to which operator belongs.

3 Click **Set and Close** to save configuration.

An output screen is displayed that shows the text produced during the update to the operator record.

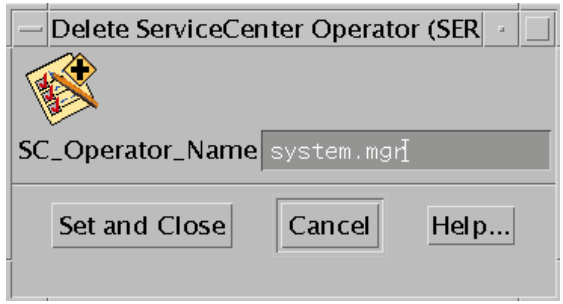


Delete ServiceCenter operator

Use this task to delete a ServiceCenter Operator record.

1 Locate and activate the **Delete ServiceCenter Operator** icon.

The Delete ServiceCenter Operator configuration screen is displayed.



2 In the **SC Operator Name** field, type the ServiceCenter operator name (e.g., *system.mgr*).

3 Click **Set and Close** to confirm output.

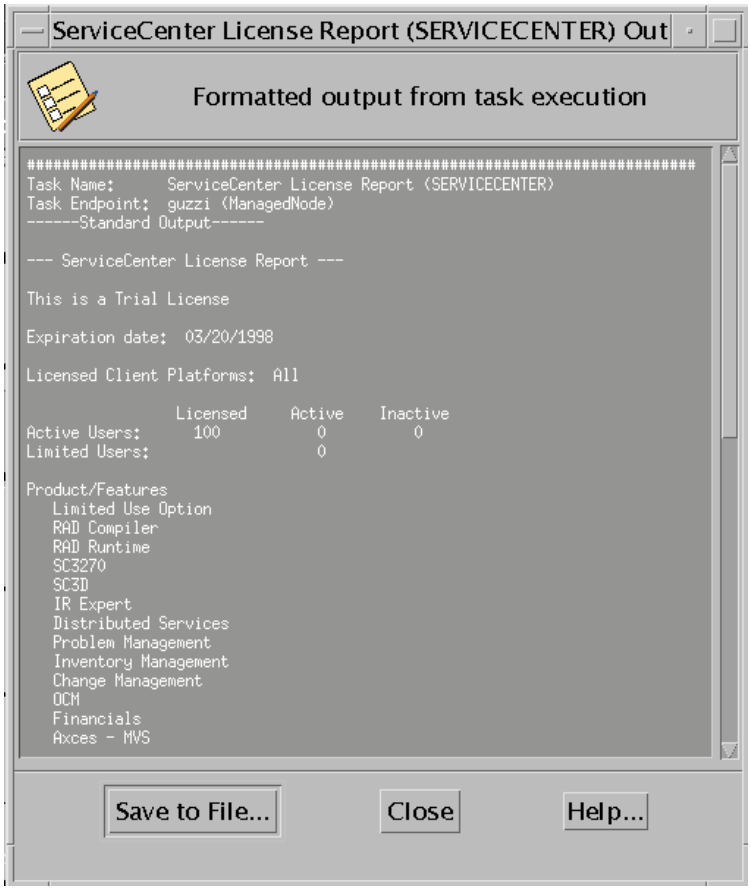
An output screen is displayed that shows the text produced during the deletion of the operator and the update to the operator record.

ServiceCenter license report

This task will list ServiceCenter License information on the selected subscriber.

1 Locate and activate the **ServiceCenter License Report** icon.

The ServiceCenter License Report output text is displayed, showing the various components covered by the current license agreement.



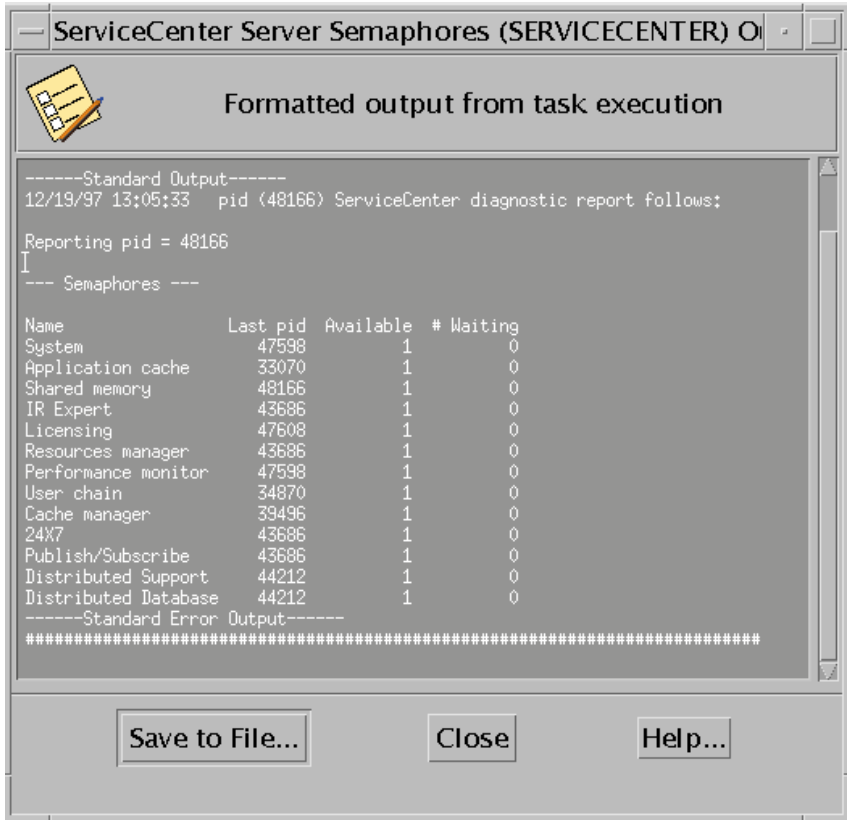
2 Click **Save to File...** to record the output data, or click **Close** to confirm configuration and exit the screen.

ServiceCenter server semaphores

This task displays current ServiceCenter semaphore usage on selected subscriber.

1 Locate and activate the **ServiceCenter Server Semaphores** icon.

The ServiceCenter Server Semaphores configuration output text is displayed.



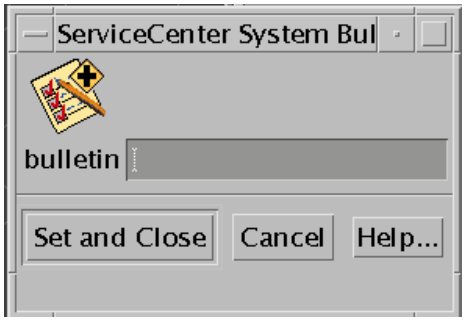
2 Click **Save to File...** to save the output, otherwise click **Close** to confirm configuration and exit the screen.

ServiceCenter system bulletin

Use this task to create a systems bulletin in ServiceCenter.

1 Locate and activate the **ServiceCenter System Bulletin** icon.

The ServiceCenter System Bulletin configuration screen is displayed.



2 In the **bulletin** field, type a text string of the bulletin you wish to post to ServiceCenter.

3 Click **Set and Close** to confirm configuration and exit the screen.

ServiceCenter publish/subscribe Information

This task displays current published information in ServiceCenter for selected subscriber.

1 Locate and activate the **ServiceCenter Publish/Subscribe Info** icon.

The ServiceCenter Publish/Subscribe Info configuration output text is displayed.

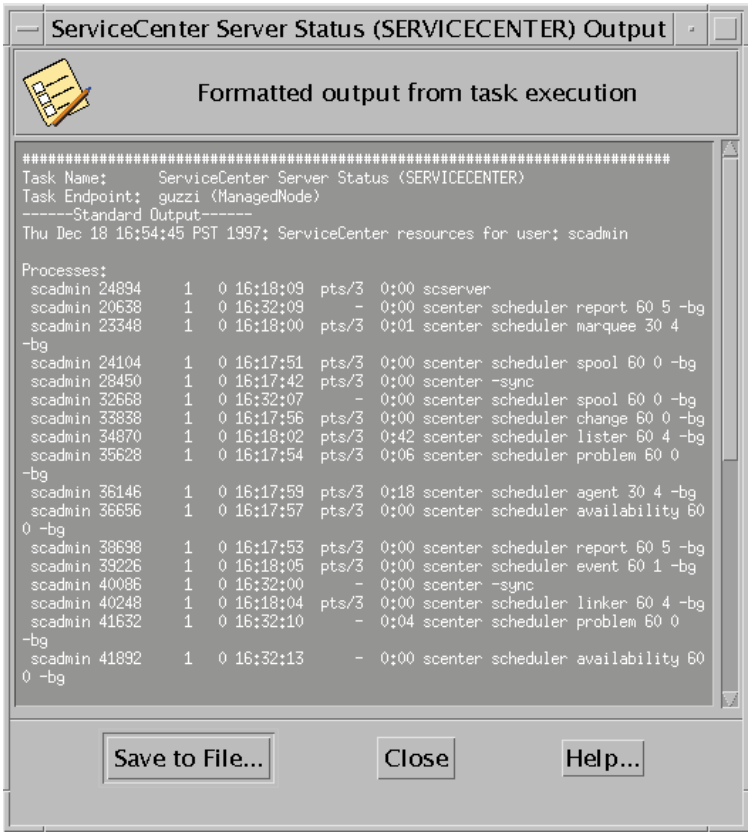
2 Click **Save to File...** to record the configuration, or click **Close** to exit the screen.

ServiceCenter server status

This task displays current ServiceCenter server status information for selected subscriber.

1 Locate and activate the **ServiceCenter Server Status** icon.

The ServiceCenter Server Status configuration output text is displayed.



2 Click **Save to File...** to record configuration data, or click **Close** to exit the screen.

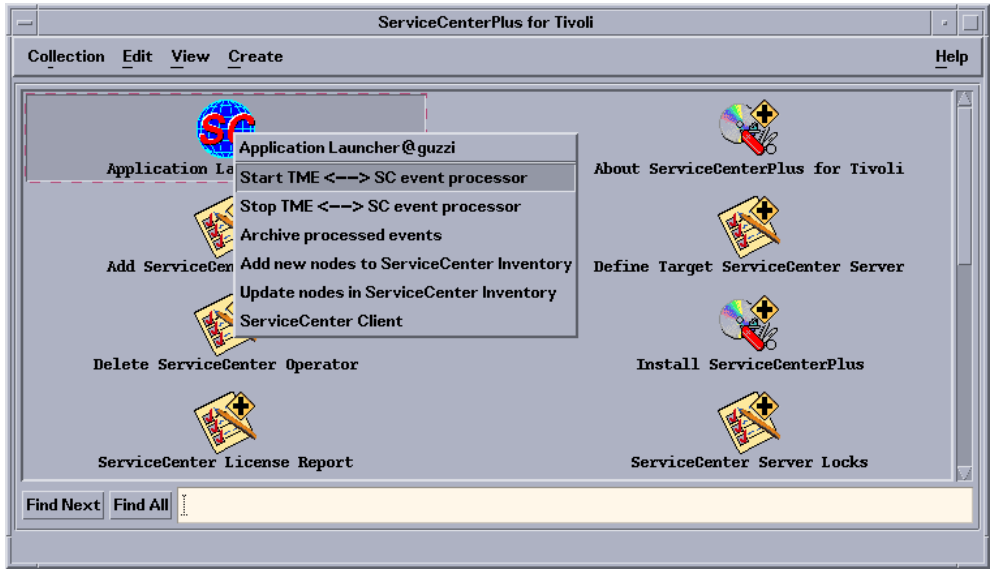
Starting and Stopping a ServiceCenterPlus for Tivoli Session

Once all the pieces of the module are configured, you can begin a session with ServiceCenter.

To start a ServiceCenterPlus for Tivoli session:

1 On the ServiceCenterPlus for Tivoli screen, right-click the **Application Launch** icon.

The application launch menu is displayed, listing the available ServiceCenter options.



2 Select **Start TME <--> SC event processor**.

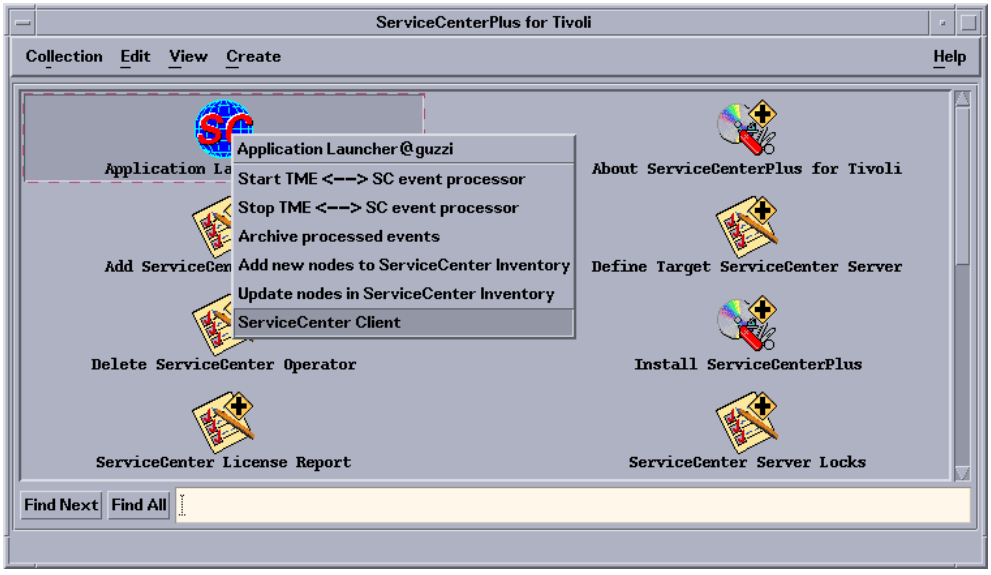
This starts an event monitor interface with ServiceCenter. SCAutomate is now started and events will be exchanged between your TME and ServiceCenter Applications.

3 To stop the session, select **Stop TME <--> SC event processor** and the application closes.

Launching a ServiceCenter client

Note: Before the client can be executed, the client connection to the server for the SC Client must be set up using either **Define Target ServiceCenter Server** or the initial **Install ServiceCenterPlus**. Be sure to configure this path before trying to startup a ServiceCenter Client.

1 On the ServiceCenterPlus for Tivoli screen menu, right-click the ServiceCenter **Application Launch** icon and keep the button depressed.

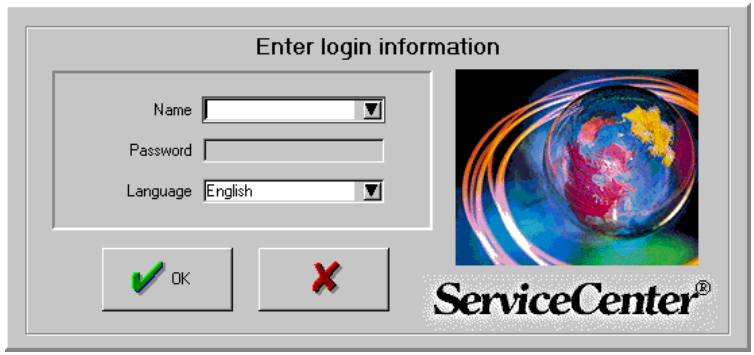


2 Select **ServiceCenter Client** and then release the button.

A message screen is displayed, indicating whether the client was launched successfully.

3 Click **Dismiss** to confirm the startup and clear the message screen.

The ServiceCenter login screen is displayed.

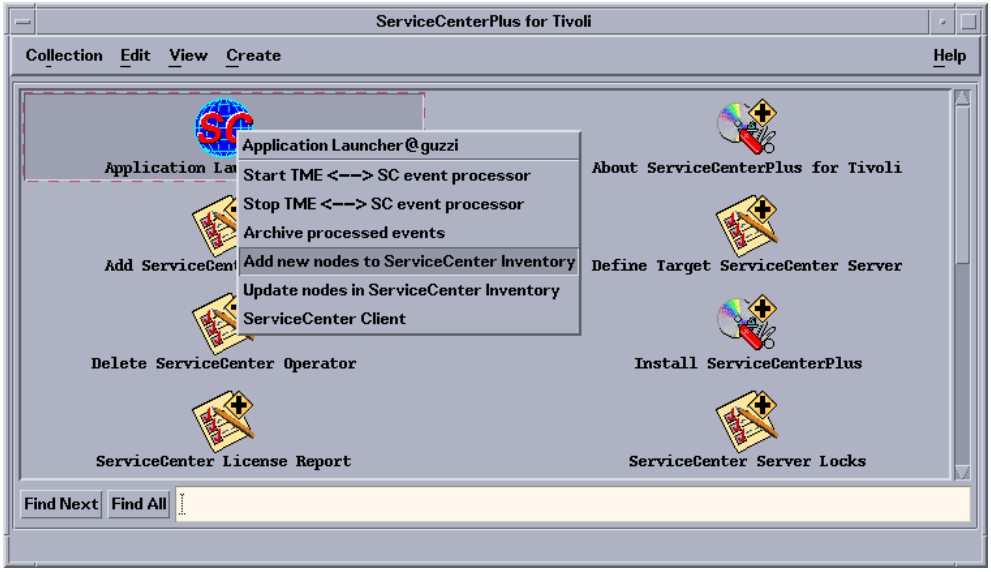


4 Login to ServiceCenter.

ServiceCenter inventory nodes

ServiceCenter inventory is the key to maintaining an interactive network management tool. Item records in the inventory database can be added, deleted, and updated using the Application Launch icon menu.

1 Right-click the **Application Launch** icon.



2 Access inventory in one of the following ways:

- To access in the Add/Delete mode, select **Add new nodes to ServiceCenter Inventory**.
- To access in the Update mode, select **Update nodes in ServiceCenter Inventory**.

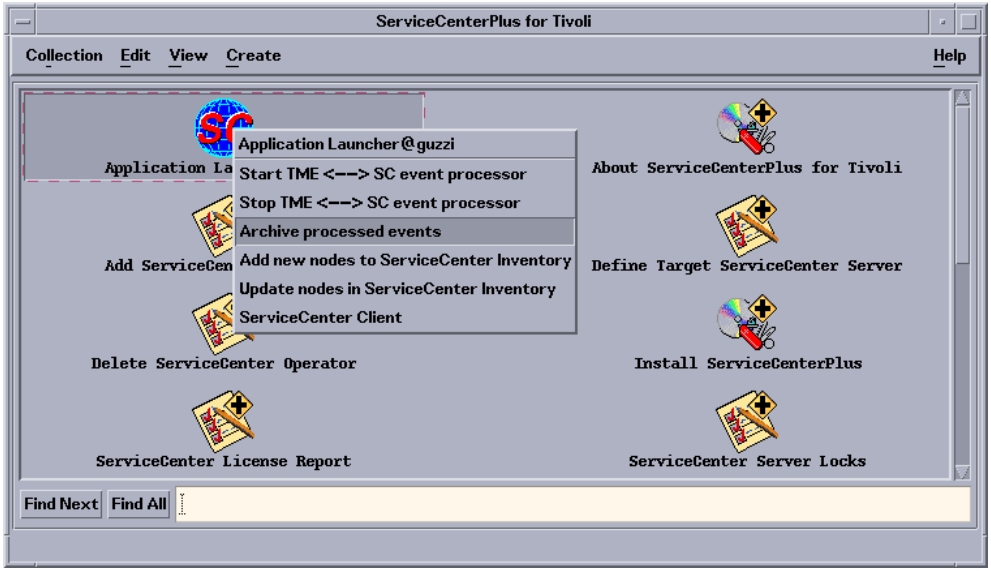
ServiceCenter processed event archive

The ServiceCenter Archive Processed Events function organizes the SCAutomate *SCEvents* queue, producing a list of only active system events. This is accomplished as follows:

- SC Archive reads if any events are present in the queue.
- A *sync* file looks to see which records have been processed in relation to the queue events.
- SC Archive removes all processed records and saves them to a timestamped file in the SCPlus directory.
- SC Archive reconfigures a new *events* file with only non-processed event records.
- SC Archive reads if any events are present in the queue.
- A *sync* file looks to see which records have been processed in relation to the queue events.
- SC Archive removes all processed records and saves them to a timestamped file in the SCPlus directory.
- Then it reconfigures a new events file with only non-processed event records.

To activate the Archive utility:

1 Right-click the **Application Launch** icon and keep the button depressed.



2 Select **Stop TME <--> SC event processor** and release the button.

3 Click **Archive processed events.**

4 Select **Start TME <--> SC event processor.**

7 Customization

CHAPTER

This chapter provides reference material and guidelines to customize ServiceCenterPlus for Tivoli TEC.

The following topics are covered:

- Customization
- Monitoring ServiceCenter
- Basic troubleshooting

Customization

Once the ServiceCenterPlus for Tivoli product is installed and configured, it provides a certain level of *out of the box* functionality. However, this functionality needs to be tailored to specific business processes, such as the processes for managing enterprise systems events, problems and problem resolution, and enterprise inventory configurations. Such tailoring and custom configuration is to be expected of any true enterprise application, and in this case, your ServiceCenterPlus for Tivoli module is building a bridge between two such enterprise applications.

To tailor this product, you must first identify the operational scenario appropriate to your organization. To assist you, sample scenarios are provided in this chapter.

As an example scenario, Peregrine Systems recommends an operational scenario in which specific classes of TEC events are forwarded to ServiceCenter, and are then not operated on from within the TEC. This allows for complete management of the problem (indicated by the original TEC event) from ServiceCenter's Problem Management application.

Once you have decided on an operational scenario, you may need to configure both Tivoli and ServiceCenter components to support your chosen scenario:

- Tivoli TEC rules can be created (or *cloned*), which cause specific classes of events to cause trouble tickets.
- TEC event to SC event conversion maps can be modified to allow specific TEC event slot values to go into specific ServiceCenter problem ticket fields.
- SC event to TEC event conversion maps can be modified to allow enhanced TEC rule processing of ServiceCenter output events related to problem tickets.
- SC event processing logic can be modified to reflect the operational scenario (e.g., always open a new ticket on incoming events from a specific node, or always update/re-open tickets on specific nodes, or either of these based on time constraints, etc.).
- SC problem ticket forms and database fields can be modified to incorporate the information and processing logic associated with your Problem Management scenario.

Modifying TEC rules

The following rule is a template that maps a NetView 6000 event, received at the TEC, into a class of event that is forwarded to ServiceCenter to create a problem ticket:

```
/*************************************************************/
/*                                                           */
/* (C) COPYRIGHT Peregrine Systems, Inc. 1996, 1997          */
/* Portions (C) COPYRIGHT Tivoli Systems, Inc. 1996, 1997    */
/* All Rights Reserved                                       */
/* Licensed Material - Property of Peregrine Systems, Inc.   */
/*                                                           */
/*************************************************************/

/* "map_NV_to_SC" finds NV6K_Events and creates SC TEC events that will */
/* be propagated into SC tickets.                                       */
/* Note: the NV6K_Event's category slot refers to the OpenView category */
/* not the SC category.  It is therefore put into the SCpmOpen's        */
/* causeCode slot.  The SCpmOpen's category is given the name of the    */
/* problem ticket format, in this case 'example'.  At this time, this   */
/* the only SCpmOpen slot that is filled with NV6K_Event specific data. */

rule: map_NV_to_SC: (
     event: _event of_class 'NV6K_Event'
          where [ source:_source,
               sub_source: _sub_source,
origin: _origin,
               sub_origin: _sub_origin,
hostname: _hostname,
               adapter_host: _adapter_host,
date:   _date,
               status: _status,
severity: _severity,
msg:    _msg,
               msg_catalog: _msg_catalog,
msg_index: _msg_index,
               repeat_count: _repeat_count,
               category:_category
],
```

```
    reception_action: gen_SC_evt: (
        generate_event( SCpmOpen,
[ source=_source,
            sub_source=_sub_source,
origin=_origin,
            sub_origin=_sub_origin,
hostname=_hostname,
            adapter_host=_adapter_host,
date=_date,
            status=_status,
severity=_severity,
msg=_msg,
            msg_catalog=_msg_catalog,
msg_index=_msg_index,
            repeat_count=_repeat_count,
            category='example',
            causeCode=_category ]
)
  )
).
```

This rule takes the data from the slots of an NV6K_Event and places them into slots of an SCpmOpen event. This allows the standard SCPlus for Tivoli rules to process this event as a TEC event that will cause a TroubleTicket. You can substitute the class of events and the slots associated for the NV6K_Event and use this template to build rules that cause your desired events to create TroubleTickets.

Event classes are defined in *baroc* files. The NV6K_Event shown previously is defined in the default NetView 6000 baroc file named *tecad_nv6k.baroc.* The SCpmOpen event class is defined in the *scenter.baroc* file.

Event maps

In the *maps* subdirectory of the location where SCPlus has been installed, there are several subdirectories. The *eventin* subdirectory contains maps that convert TEC events into the valid SC events. The README file in this directory explains the format of these files. These maps are also described in Chapter 4.

The *eventout* subdirectory contains maps which convert SC events into TEC events. The README file in this directory also describes the format of these files, as well as how to modify them for your purposes. This is also given expanded treatment in Chapter 4.

Modifying SC Event Services

In the ServiceCenter Event Services administration there are events registration, event maps, and event filters. Each one of these can be used to modify the way SC Event service processes incoming and outgoing SC events.The key events you may want to modify are those associated with the TEC Event conversions (*pmo* - for opening a problem ticket, *pmu* - for updating a ticket, *pmc* - for closing a ticket).

Sample Scenarios

Sample TEC-created and TEC-managed problem scenario

Any TEC event can be used to produce a ServiceCenter event.

- To create a problem ticket, state a rule to launch the command **/usr/SCPlus/lib/sceventin.sh** when the event is received. This shell will create an input event in ServiceCenter using a provided map for your event in **/usr/SCPlus/maps/eventin** or the DEFAULT eventin map. Refer to the maps supplied in **/usr/SCPlus/maps/eventin**.
- To update ServiceCenter when the event is acknowledged, add a rule to the event to launch **/usr/SCPlus/lib/scpmuack.sh** when the event is acknowledged. This will create a problem update(*pmu*) event in ServiceCenter. Therefore, the event being acknowledged must have been used to open a problem or it must contain a problem number slot.
- To update ServiceCenter when an events severity is changed, add rules to launch **/usr/SCPlus/lib/scpmusev.sh** when the severity is raised or lowered. This also creates a problem update(*pmu*) and must be the event which opened the ticket or it must have a problem number slot.
- To close the problem from Tivoli, state a rule that when an event is closed, launch **/usr/SCPlus/lib/scpmclose.sh**. This will create a *pmc* event to close the ticket in ServiceCenter when the TEC event is closed. The event closed must be an event that created a problem ticket or contain a problem number slot.

Sample ServiceCenter-created and TEC-managed problem scenario

- To notify the TEC when a problem ticket is opened in ServiceCenter, provide a **/usr/SCPlus/maps/eventout/pmo** map to create an *SCpmOpened* event in the TEC. This event can be defined with rules to launch the same functions as mentioned previously.
- To update ServiceCenter when the event is acknowledged, add a rule to the event to launch **/usr/scauto/tivoli/bin/scpmuack.sh** when event is acknowledged.

- To update ServiceCenter when an events severity is changed, add rules to launch **/usr/SCPlus/lib/scpmusev.sh** when the severity is raised or lowered.
- To close the problem from Tivoli, state a rule that when the event is closed, launch **/usr/SCPlus/lib/scpmclose.sh**. This will create a *pmc* event to close the ticket in ServiceCenter when the TEC event is closed.

Sample ServiceCenter-created and ServiceCenter-managed problem scenario

To notify the TEC when a problem ticket is opened in ServiceCenter you must provide an **/usr/SCPlus/maps/eventout/pmo** map to create an *SCpmOpened* event in the TEC. This event is managed by ServiceCenter output events.

Status

Two methods are available to inform the TEC of the status of the problem:

- The first method is the creation of events in the TEC whenever events occur in ServiceCenter. The TEC console can then acknowledge and close the events as they are reviewed. This may or may not cause activity in ServiceCenter based on the rules specified for these new events. The required maps are supplied as part of the Plus module but may be modified as described in the *Tivoli Event Mapping* section.
  - To update the TEC when a ServiceCenter problem update has occurred you must provide a **../eventout/pmu** map to create a *SCpmUpdated* event in the TEC. Each problem update will create a new SCpmUpdated event.
  - To notify the TEC when the problem is closed, you must provide a **../eventout/pmc** map to create a *SCpmClosed* event in the TEC.
- The second method using ServiceCenter output events is to launch commands on TEC when the ServiceCenter output events are created. The commands provided allow you to acknowledge, update the severity, and then close the initial *SCpmOpened* event.
  - When the *SCpmOpened* event is received, your rule must launch the **/usr/SCPlus/lib/scpmuack.sh**. This updates the ServiceCenter ticket that the event has created in TEC, but more importantly it provides the event handle to be used in update and close.

- To update the TEC when a ServiceCenter problem update has occurred, you must provide an **../eventout/pmu** map to launch **sctmesev.sh**, which will update the severity of the problem. Other fields in the update are not reviewed using this method and this may or may not be relevant in your choice of method.
- To notify the TEC when the problem is closed, you must provide a **../eventout/pmc** map to launch **sctmeclosed.sh**, which will close the *SCpmOpened* event in the TEC.

Monitoring ServiceCenter

Tivoli/Sentry and Tivoli/Logfile monitors may be employed to improve the performance and availability of your help desk applications. Some initial SCAutomate monitors are configured in the Plus module, but with the number of platforms and operational configurations possible in ServiceCenter, monitoring specifications may be required.

A suggested methodology would be to review your production environment and note the critical resources required to maintain satisfactory performance and availability of your applications. The environment might include the monitoring of applications other than ServiceCenter (e.g., Oracle, Sybase, inet daemon, etc.). Once the critical resources are identified, select the appropriate monitor and then configure and deploy it to the necessary platforms. Where possible, proactive monitoring and automatic recovery could greatly enhance your availability (e.g., daemon process restart autotasks).

Included is a starter list of messages which may be monitored in *sc.log* by the LOGFILE monitor (**/usr/SCPlus/lib/logfile.msgs**).They may be used to create your initial *.fmt* file(s).

Also you may want to provide Tivoli/Sentry Monitors to monitor the following on your ServiceCenter platform(s):

- File Size
  - ../sc/sc.db1-n Critical > 1.5 Gigabytes
  - ../sc/sc.db1-n Warning > 1.0 Gigabytes
  - ../sc.log Warning > 200K
- Processes
  - scserver
  - listeners (sc3270,express)
  - schedulers (report, event, alert, scauto server, etc..)
- Swapspace
  - Critical < 10M
  - Warning < 20M
  - Severe < 15M

- Pageouts
  - Critical > 50K
- Freespace
  - Critical < 300M

8 Troubleshooting

CHAPTER

This chapter provides tips for troubleshooting the system.

If the verification test fails during connection between the SCAutomate Base (scautod) and the SCAutomate Tivoli (scevmon) please check the following common failures:

- Is scautod running on ServiceCenter Server Platform? Use ServiceCenter *status* command to check. If it is **not running** start it from the status option *start schedulers* selecting *scauto.startup*. If the start fails the scautod daemon logs to the *sc.log* file, so be sure to specify and check the log for any error messages.Check the TCP/IP specifications for your platform, a service name other than that specified for ServiceCenter server is required, and the *scauto:* keyword should be specified in the *sc.ini* file and match this name. If it is **running** check the service name specified in *the sc.ini* file on the ServiceCenter platform for the *scauto:* keyword. If the keyword is not specified, scautod defaults to the services name *scauto*. This will be useful in the next step.

- Are the TCP/IP specifications correct on the SCAutomate Tivoli platform? Check the host and service specification, are they specified and do they **match** the *host* and *service* name of *scautod?* Specifications may be made in local files (**UNIX /etc/hosts, /etc/services**) or a name server may be used. Check with your network administrator for specification. Also, check your specification in the **/usr/SCPlus/bin/sc.ini** configuration file and make sure they all match.
- Is there connectivity between SCAutomate Base (scautod) and the SCAutomate Tivoli platform? *Ping* the SCAutomate Base (scautod) platform from the SCAutomate Tivoli platform. If this fails and all TCP/IP specifications are correct contact your network administrator for assistance.

If you have an event in the eventin file but no RAD application is invoked (e.g. *pmo* event and no problem opened) check the following:

- Is the Event Scheduler running on the ServiceCenter server platform? Use ServiceCenter *status* command to check. If it is **not running** start it from the status option *start schedulers* selecting *event.startup*. If the start fails review error messages and retry. If errors persist call Peregrine Systems Customer Support. If it is **running** and not processing, stop the Event Scheduler and build a new *schedule record* and restart.
- Review the section titled *Basic Trouble Shooting* in the *Event Services Guide* in regard to schedule record specifications.

No event created in eventout file but RAD application has been invoked (e.g. a problem was opened and no *pmo* output event created) check the following:

- Refer to *Using Format Control to Write Eventout Records* section in the *Event Services Guide*. The problem is associated with the application generating the output event(e.g. Problem Management).
- Review the *Basic Trouble Shooting*, and relevant sections of *Event Services Guide* for output event generation (e.g.*Writing Eventout Records form Problem Management*).

If you have an event in the eventout file but the external SCAutomate Application (*scevmon*) is not invoked (e.g. *pmo* event and no TEC event created) check the following:

- Is there connectivity to the SCAutomate Base server? Review *connection between the SCAutomate Base (scautod) and the SCAutomate Application (scevmon)* in the *troubleshooting* section.
- Is SCAutomate application (*scevmon*) active? If it is **not running** check the SCAutomate *logfile (***/usr/SCPlus/sctivoli.log***)* for errors. Correct errors and, restart using appropriate command with the **-debug** option. The debug option will allow you to determine if the operation was attempted and provides more information on the problem. If restart fails, review the SCAutomate *logfile.* Also check that a corresponding eventout map was provided, without an eventout map the event is discarded. Iff it is **running**, check the SCAutomate application *logfile* for errors. Correct errors and, restart using appropriate command with the debug option if available. Recheck the *logfile* and try commands (**wpostemsg**) manually as shown with the **-debug** option.

If you don't have an event in the *eventin* file but the external SCAutomate Application event has occurred (e.g. TEC event to launch TroubleTicket.sh to open a problem ticket and no *pmo* event created) check the following:

- Is there connectivity to the SCAutomate Base server? Review *Connection between the SCAutomate Base (scautod) and the SCAutomate Application (scevmon)* in the *troubleshooting* section.
- Is SCAutomate *scevmon* active? If it is **not running** check the SCAutomate *logfile (sctivoli.log)* for errors. Correct errors and, restart using appropriate command with the **debug** option. The debug option will allow you to determine if it is an SCAutomate application failure and provides more information on the problem. If restart fails using debug option review the SCAutomate application *logfile* for any new error message information. If it is **running**, check the SCAutomate application *logfile* for errors. Also, check your evfilter.sh and ServiceCenter filter records to ensure that you are not filtering the event. Correct errors and, restart using appropriate command with the **debug** option if available. Recheck the *logfile* and correct errors and retry.

If you need assistance in the resolution of your problem, please have the following available before calling Peregrine Systems for support:

- Core files
- Error logs produced by service program (e.g., Tivoli) and SCAutomate application with debug option
- Filter, map files, class, and rule information
- ServiceCenter log and Scheduler log(s) if specified
- ServiceCenter msglog for affected applications or services
- ServiceCenter print of agent status (Event Services menu)
- Unloaded Event records relevant to errors
- Unloaded ServiceCenter filter specifications if relevant

Index

A

add operator 119
add ServiceCenter operator 119
admin 31, 34
AssignDepartment 120, 122

B

baroc 47
bulletin 126

C

classes
  baroc 47
  scenter.baroc 41, 138
commands
  evfilter.sh 86
  scpmuack.sh 62, 74, 77
  scpmusev.sh 62, 75
  sctmeclose.sh 63, 73, 76, 79
  sctmesev.sh 62, 73, 78
  wcomprules 55
configuration
  add ServiceCenter operator 119
  AssignDepartment 120, 122
  bulletin 126
  delete ServiceCenter operator 123
  EmailAdd 120, 121
  FaxNumber 120, 122
  FullName 119, 121
  PagerNumber 120, 122
  password 119, 121
  PhoneNumber 120, 122
  SC_Client 130
  SC_ShMem 118
  SCName 119, 121, 123
  scpath 115, 116
  scuser 115, 116
  update ServiceCenter operator 121
contacting Peregrine Systems 8

D

delete operator 123
delete ServiceCenter operator 123

E

EmailAdd 120, 121
events
  files
    scevents 17
  pmc 63, 73, 75, 79, 140, 141, 142
  pmo 62, 72, 74, 76, 77, 80, 81, 82, 83, 84, 85, 146, 147
  pmu 62, 73, 74, 77, 78, 140, 141
  problem closed 63, 73, 75, 79, 140, 141, 142
  problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
  problem opened 72, 74, 76
  problem updated 62, 73, 74, 77, 78, 140, 141
  referenceNo 17
  scevents 17
  scevmon 17, 145, 147
  SCpmClose 75, 79

SCpmClosed 141
SCpmOpen 76
SCpmOpened 74, 76, 77, 78, 140, 141, 142
SCpmUpdate 74, 75, 77
SCpmUpdated 141
Slot
referenceNo 17
wpostemsg 147
evfields 86
evsepchar 86
evtype 86
evuser 86

F

FaxNumber 120, 122
fields
AssignDepartment 120, 122
bulletin 126
EmailAdd 120, 121
FaxNumber 120, 122
FullName 119, 121
PagerNumber 120, 122
password 119, 121
PhoneNumber 120, 122
SCName 119, 121, 123
scpath 115, 116
scuser 115, 116
files
scevents 17
filters
evfields 86
evsepchar 86
evtype 86
evuser 86
FullName 119, 121

I

install
new rule base 42
TEC event server 41
install ServiceCenterPlus task 40

L

launching SCAuto 129
logs
scautod.log 44

N

new rule base 42

P

PagerNumber 120, 122
parameters
evfields 86
evsepchar 86
evtype 86
evuser 86
scauto 44
password 119, 121
Peregrine Systems, contacting 8
PhoneNumber 120, 122
pmc 63, 73, 75, 79
pmo 62, 68, 72, 74, 76, 77, 80, 81, 82, 83, 84, 85, 146, 147
pmu 62, 73, 74, 78
problem closed 63, 73, 75, 79, 140, 141, 142
problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
problem opened 72, 74, 76
problem updated 62, 73, 74, 77, 78, 140, 141
processes
sceventin.sh 62, 140
scpmclose.sh 62

R

referenceNo 17
roles
admin 31, 34
setting up roles 31
user 31, 34
rules
bases
scenter.baroc 42
scenter.rls 42
scenter.baroc 47
scenter.rls 41, 49

S

SC_Client 130
SC_ShMem 118

SCAuto
events
pmo 62, 80, 81, 82, 83, 84, 85, 146, 147
scauto 44
SCAuto startup
application launch 129
scautod 15, 145, 147
scautod.log 44
SCAutomate
events
monitor
scevmon 17, 145, 147
pmc 63, 73, 75, 79, 140, 141, 142
pmo 72, 74, 76, 77
pmu 62, 73, 74, 77, 78, 140, 141
problem closed 63, 73, 75, 79, 140, 141, 142
problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
problem opened 72, 74, 76
problem updated 62, 73, 74, 77, 78, 140, 141
scevmon 17, 145, 147
SCpmClose 16, 75, 79
SCpmClosed 141
SCpmOpen 16, 76
SCpmOpened 74, 76, 77, 78, 140, 141, 142
SCpmUpdate 16, 74, 75, 77
SCpmUpdated 17, 141
pmo 72, 74, 76, 77
problem closed 63, 73, 75, 79, 140, 141, 142
problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
problem opened 72, 74, 76
problem updated 62, 73, 74, 77, 78, 140, 141
processes
sceventin.sh 62, 140
scpmclose.sh 62
rules
sceventin.sh 62, 140
scpmclose.sh 62
scautod 15, 145, 147
scautod.log 44
sceventin.sh 62, 140
scevmon 17, 145, 147
SCpmClose 16, 75, 79
scpmclose.sh 62
SCpmClosed 141
SCpmOpen 16, 76
SCpmOpened 44, 74, 76, 77, 78, 140, 141, 142
SCpmUpdate 16, 74, 75, 77
SCpmUpdated 17, 141
scenter.baroc 41, 42, 47, 138
scenter.rls 41, 42, 49
sceventin.sh 62, 140
scevents 17
scevmon 17, 145, 147
SCName 119, 121, 123
scpath 115, 116
SCpmClose 16, 75, 79
scpmclose.sh 62
SCpmClosed 141
SCpmOpen 16, 76
SCpmOpened 44, 74, 76, 77, 78, 140, 141, 142
SCpmUpdate 16, 74, 75, 77
SCpmUpdated 17, 141
scpmusev.sh 62, 75
sctmeack.sh 62, 72
sctmeclose.sh 63, 73, 76, 79
sctmesev.sh 62, 73, 78
scuser 115, 116
ServicecCenter operator-update 121
ServiceCenter
baroc 47
classes
baroc 47
scenter.baroc 41, 138
events
pmo 72, 74, 76, 77
problem closed 63, 73, 75, 79, 140, 141, 142
problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
problem opened 72, 74, 76
problem updated 62, 73, 74, 77, 78, 140, 141
SCpmClose 75, 79
SCpmClosed 141
SCpmOpen 76

SCpmOpened 74, 76, 77, 78, 140, 141, 142
SCpmUpdate 74, 75, 77
SCpmUpdated 141
rules
scenter.baroc 47
scenter.rls 41, 49
scautod 15, 145, 147
scenter.baroc 41, 47, 138
scenter.rls 41, 49
ServiceCenter operator- add 119
ServiceCenter operator- delete 123
ServiceCenter server status task 40
setup TEC event server 45
support, contacting 8

T

tasks
install ServiceCenterPlus 40
ServiceCenter server status 40
setup TEC event server 45
TEC
events
pmo 72, 74, 76, 77
problem closed 63, 73, 75, 79, 140, 141, 142
problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
problem opened 72, 74, 76
problem updated 62, 73, 74, 77, 78, 140, 141
SCpmOpen 76
eventsSCpmClose 75, 79
eventsSCpmClosed 141
wpostemsg 147
TEC event server 41
technical support, contacting 8
Tivoli
events 74, 75, 77, 141
pmo 72, 74, 76, 77
problem open 62, 68, 72, 74, 76, 80, 81, 82, 83, 84, 85, 146, 147
problem opened 72, 74, 76
problem updated 62, 63, 73, 74, 75, 77, 78, 79, 140, 141, 142
SCpmClose 16, 75, 79
SCpmClosed 141
SCpmOpen 16, 76
SCpmOpened 44, 74, 76, 77, 78, 140, 141, 142
SCpmUpdate 16
SCpmUpdated 17
wpostemsg 147
evfilter.sh 86
filters
evfields 86
evsepchar 86
evtype 86
evuser 86
parameters
evfields 86
evsepchar 86
evtype 86
evuser 86
problem opened 72, 74, 76
SCpmClose 16
SCpmOpen 16
SCpmUpdate 16
SCpmUpdated 17
setup TEC event server 45
shells
evfilter.sh 86
sceventin.sh 62
scpmclose.sh 62
scpmuack.sh 62, 74, 77
scpmusev.sh 62, 75
sctmeack.sh 62, 72
sctmeclose.sh 63, 73, 76, 79
sctmesev.sh 62, 73, 78
tasks
setup TEC event server 45
wpostemsg 147
TME10
commands
scpmuack.sh 62
sctmeclose.sh 63, 73, 76, 79
sctmexev.sh 62, 73, 78

U

update operator 121

update ServiceCenter operator 121
user 31, 34
user roles
admin 31, 34
setting user roles 31
user 31, 34

W

wcomprules 55
wpostemsg 147