Peregrine ServiceCenter

Event Services

Release 6

PART NO: DSC-6-EN19

Peregrine SYSTEMS®

The information in this document is subject to change without notice and does not represent a commitment on the part of Peregrine Systems, Inc. Contact Peregrine Systems, Inc., Customer Support to verify the date of the latest version of this document. The names of companies and individuals used in the sample database and in examples in the manuals are fictitious and are intended to illustrate the use of the software. Any resemblance to actual companies or individuals, whether past or present, is purely coincidental. If you need technical support for this product, or would like to request documentation for a product for which you are licensed, contact Peregrine Systems, Inc. Customer Support by email at support@peregrine.com. If you have comments or suggestions about this documentation, contact Peregrine Systems, Inc. Technical Publications by email at doc_comments@peregrine.com. This edition of the document applies to version 6.0 of the licensed program.

040507

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

Contents

About this Guide

Increasingly, enterprise-wide network management tools depend on automation to detect activity on the network and to execute the appropriate procedures. These network incidents are often called alarms or alerts; ServiceCenter refers to them as *events*.

This manual introduces *Event Services* and explains:

- *Standard Event Operations* on page 25
- *Mapping and Filtering* on page 43
- *ServiceCenter/Network Discovery Integration* on page 73
- *Change Management Event Services* on page 79
- *Event Agent Operations* on page 87
- *SCemail* on page 99
- *Format Control Options* on page 115
- *Basic Troubleshooting* on page 127

Knowledge requirements

Readers of this guide need general knowledge of the following:

- How ServiceCenter works
- How the underlying database functions, and
- How the interface passes data into ServiceCenter.

Before you begin using this guide, become familiar with topics in ServiceCenter and third-party documentation as follows:

- For a working knowledge of ServiceCenter, search the *Administering ServiceCenter* online help topics.
- For an understanding of how to use Database Manager and view data in records, search the *Database Management* online help topics.
- To become familiar with ServiceCenter interfaces, search the *ServiceCenter* online help topics; for an understanding of external interfaces, refer to the appropriate product documentation.

Sample forms and examples

The sample forms and examples included in this guide are for illustration only, and may differ from those at your site.

Need further assistance?

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

Customer Support

For further information and assistance, contact Peregrine Systems Customer Support at support.peregrine.com.

If the KnowledgeBase does not contain an article that addresses your concerns, you can search for information by product; search discussion forums; and search for product downloads.

Documentation Web site

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

You can view PDF files, including release notes using Adobe Reader™, which is available at www.adobe.com.

Education Services Web site

Peregrine Systems offers classroom training anywhere in the world, as well as “at your desk” training via the Internet. For a complete listing of Peregrine’s training courses, see www.peregrine.com/education. You can also contact Peregrine Systems Education Services at +1 858.794.5009.

1 Introduction

CHAPTER

ServiceCenter Event Services provides a bi-directional interface between ServiceCenter and external systems. It is the preferred mechanism for interfacing ServiceCenter to external systems. It differs from the ServiceCenter System Event/DDE interface in that it supports more platforms, does not require an existing Windows client session to operate, and can be configured to run in the background.

Some of the products using ServiceCenter Event Services are:

- NetView Automated Problem Applications (NAPA)
- SCAuto for NetView OS/390 (which replaced NAPA)
- SCEmail (allowing ServiceCenter to send e-mail to the world)
- SCMail (UNIX) and SCMapi (Windows)—ServiceCenter two-way e-mail
- SCAutomate products—third-party products to tie ServiceCenter to products such as Tivoli, HP OpenView, and so on
- Custom-written SCAutomate applications
- Get-It (providing a real-time web interface to ServiceCenter)
- Connect-It (unique in that it uses Event Services inbound only, but for outbound operations can read ServiceCenter directly)

To accomplish the communication between products, you must establish a connection between ServiceCenter and the external system. The type of connection is dependent on the product and environment involved (although in most cases, some type of TCP/IP connection is used, often involving the **scenter** listener).

Once information comes into ServiceCenter, Event Services provides a series of standard applications to:

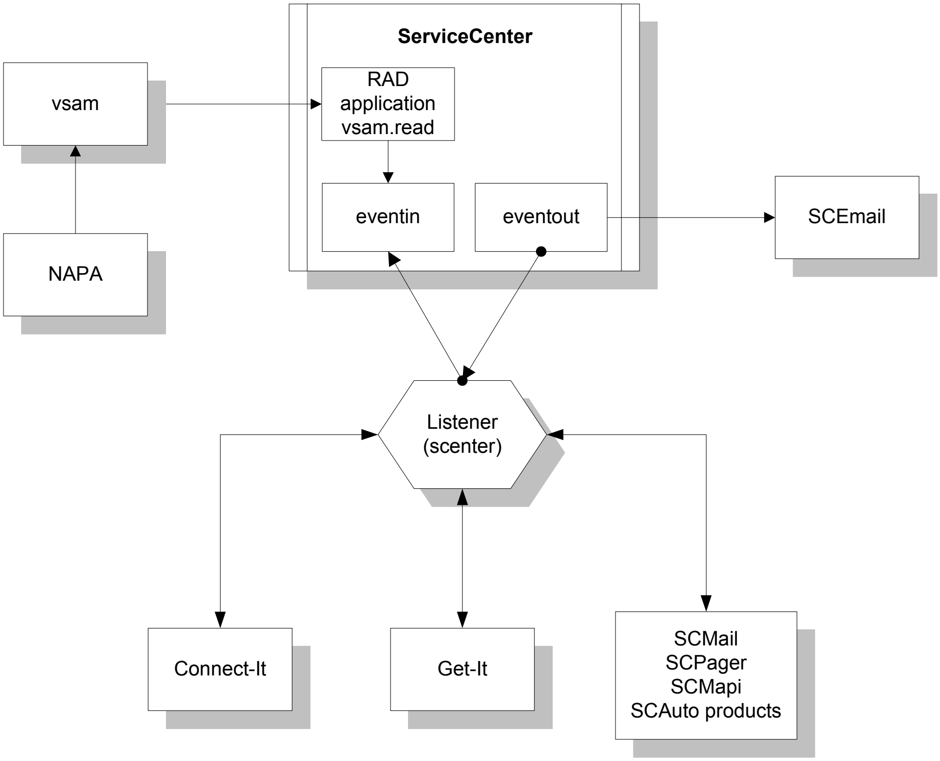
- Open, update, and close incidents.
- Open, update, and close calls.
- Add, update, and delete inventory items.
- Open, update, approve, and close changes and requests.

Similarly, standard applications are available for use within ServiceCenter to generate outbound information (such as e-mails). The standard applications come with predefined formats for the information. Through tailoring, you can change these formats and operations. The product is extensible; you can modify ServiceCenter Event Services to perform virtually any ServiceCenter operation on any table within the product.

How Event Services works

Events entering and exiting ServiceCenter are routed differently depending upon the external system with which ServiceCenter is communicating. For some products, information is routed in one direction only. For others, events flow in both directions. The following table shows the routing of events through external products currently supported.

File	Description
NAPA	Inbound events only. Information is routed from **vsam** (outside of ServiceCenter). An application within ServiceCenter, called **vsam.read**, reads a record from an existing VSAM data set and writes a corresponding **eventin** record. The capability to read VSAM data sets residing on an MVS machine is native to ServiceCenter. If the ServiceCenter server resides on a non-MVS platform (UNIX, Windows), you need to configure a connection between the ServiceCenter server and the MVS machine where the VSAM data set resides utilizing the SC3270 product.
SCEmail	Outbound events only. Information is routed from ServiceCenter to SCEmail by an **eventout** record. SCEmail, a special executable that runs on the same platform as the ServiceCenter server, is designed to connect directly to the server, and responds to and processes only e-mail **eventout** records.
Connect-It	Inbound events. Connect-It establishes a client connection to the ServiceCenter server through a listener (**scenter**) and information is routed bi-directionally through it. Connect-It is specifically designed so inbound information must go through Event Services. However, outbound information may be read directly from anywhere in the system.
Get-It	Inbound and outbound events. Get-It establishes a client connection to the ServiceCenter server through a listener (**scenter**) and information is routed bi-directionally through it. Get-It inbound and outbound information goes through Event Services (**eventin** for the inbound, **eventout** for the outbound).
SCMail SCPager SCMapi SCAuto products	Inbound and outbound events. The products establish a client connection to the ServiceCenter server through a listener (**scenter**) and information is routed bi-directionally through it.



Event Services files

There are five principle tables in ServiceCenter to define events and how they work. The names of these five files all are of the format **event***.

File	Description
eventregister	Defines the events that exist in the system. Event registration records also specify the eventmaps used to process events and defines the RAD application used for processing.
eventin	File used to move information into ServiceCenter from an external system. If a corresponding **input** eventregister record exists, external or internal applications can write records to the **eventin** file.
eventout	File used to move information from ServiceCenter into an external system. A particular type of an **eventout** record can be written only if a corresponding **output** eventregister record exists.

File	Description
eventmap	Defines how information is parsed. Eventmaps define individual fields and create condition statements for eventin and eventout records. Many eventmap records can exist for each eventregistration record.
eventfilter	Prevents duplicate events. Filters block incoming events based on defined criteria to prevent external systems from creating many eventin records for the same item in a short amount of time. Filters can block events by time frame, item, or location.

Event Services flowchart

This flowchart depicts a macro view of ServiceCenter Event Services.

Event Services (ES) uses its own scheduler called **event**. You start and stop the event scheduler like any other ServiceCenter scheduler and process events in background or asynchronously.

The event registration file contains all of the information ES needs to determine what to do with each event.

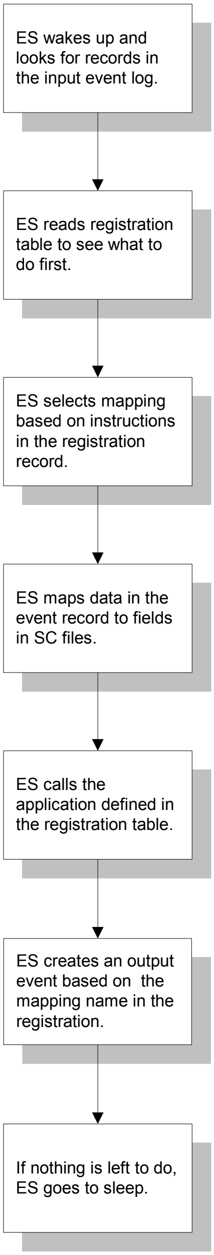
Mapping records contain instructions to move data from the eventin record to fields in ServiceCenter files.

Based on instructions in the mapping records, a data structure is built.

A multi-purpose call routine is issued to the application named in the registration record, along with any necessary variables.

When the application has completed, an output event is created and added to the queue, if instructed by the registration.

If Event Services has nothing left to do, it sleeps for an interval, then reawakens to look for more work.


ES wakes up and looks for records in the input event log.
ES reads registration table to see what to do first.
ES selects mapping based on instructions in the registration record.
ES maps data in the event record to fields in SC files.
ES calls the application defined in the registration table.
ES creates an output event based on the mapping name in the registration.
If nothing is left to do, ES goes to sleep.

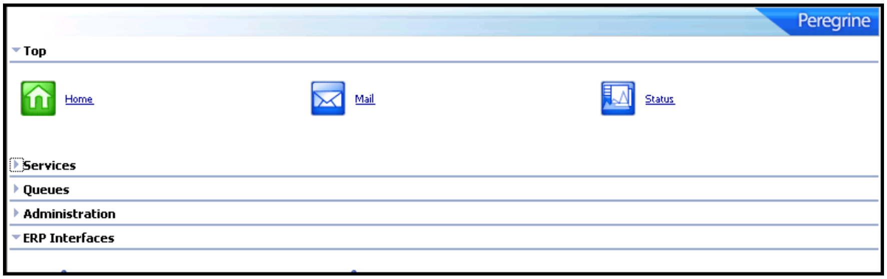
Accessing Event Services

You must be a ServiceCenter system administrator to work in Event Services.

Graphical User Interface client

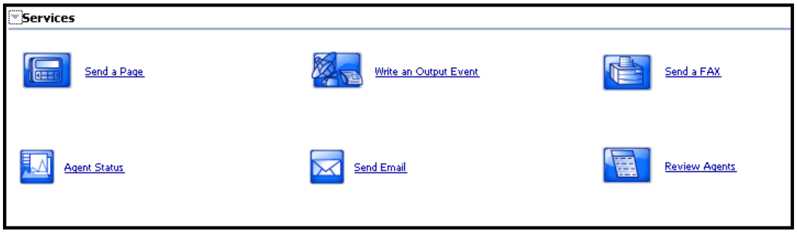
To access Event Services

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** to open the Event Services menu.



This menu controls all of the applications, parameters and filters that SCAuto for NetView OS/390 and ServiceCenter Automate (SCAuto) use.

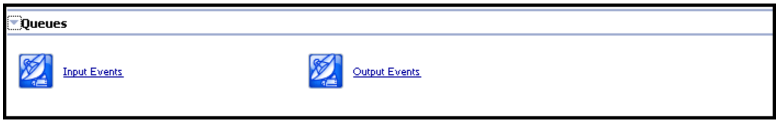
Services menu



Button	Action
Send a Page	Initiates a paging event and opens the page transmission form, pager.info.g.
Send a Fax	Initiates the fax event and opens the fax transmission form, send.fax.g.
Send Email	Initiates the e-mail event and opens the e-mail transmission form, send.email.g.

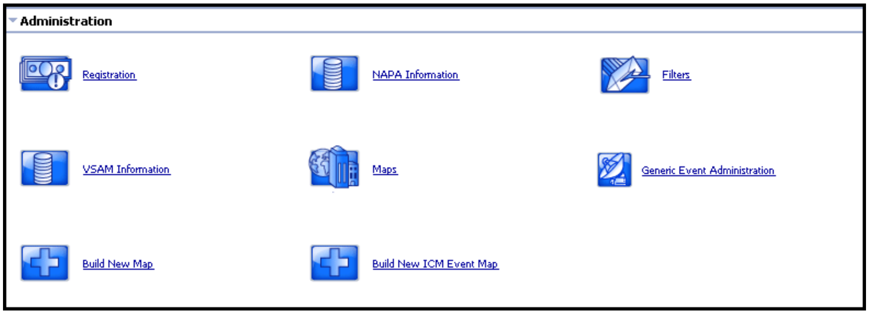
Button	Action
Write an Output Event	Initiates event create script, prompting you for selection of type of external event: Incident, Inventory or Generic (message).
Agent Status	Displays a status list of all SCAutomate agents, including last expiration and idle time, and provides Start and Stop controls for each agent.
Review Agents	Opens the Event Scheduler screen, displaying details for scheduled system events.

Queues menu



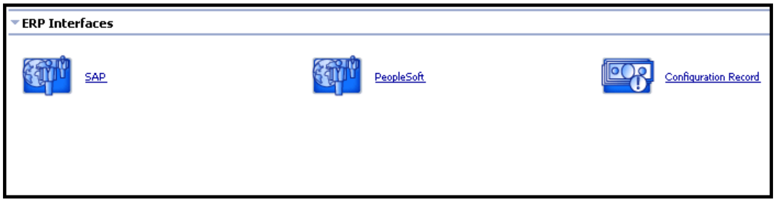
Button	Action
Input Events	Opens the **eventin** file for review. This file contains all events awaiting action by ServiceCenter and those that have been processed but not deleted.
Output Events	Opens the **eventout** file for review. This file contains all ServiceCenter events awaiting action by an external application and those that have been processed but not deleted.

Administration menu



Button	Action
Registration	Accesses Event Services registration records. Each registration record provides the information ServiceCenter requires to process and event type.
Filters	Allows maintenance of event filters. Although general in scope so that you can use filters for any purpose, the primary focus is on incident filtering.
Maps	Allows maintenance of existing eventmaps. Eventmaps define the relationship between data passed into and out of ServiceCenter in flat, delimited form, and fields in ServiceCenter files.
Build New Map	Accesses an application which helps to quickly define a new eventmap for a ServiceCenter file.
NAPA Information	Allows maintenance of the NAPA schedulers.
VSAM Information	Allows maintenance of the vsaminfo file.
Generic Event Administration	Opens the Generic Events Menu of controls, which include editing of GOE configuration records, mass exporting of records, and exporting of database dictionary structures.
Build New ICM Event Map	Begins the new inventory script, beginning by prompting you for the ICM file upon which to build the eventmap.

ERP Interfaces menu



Button	Action
SAP	Opens the SAP R/3 Interface menu for establishing an exchange of SAP HR contact and support records.
PeopleSoft	Opens the PeopleSoft Interface menu for establishing an exchange of PeopleSoft contact and support records.
Configuration Record	Accesses the SAP HR configuration file, where the HR and Materials Management (MM) interfaces are specified, and default server names are identified.

2 CHAPTER

Standard Event Operations

Events take many forms and occur at various times throughout the operation of the system. See *Common Events* on page 135 for some of the more commonly-used events.

The primary operations of ServiceCenter Event Services include:

- *Event registration* on page 26
- *Input events* on page 33
- *Output events* on page 40
- *Generic Event Administration* on page 42

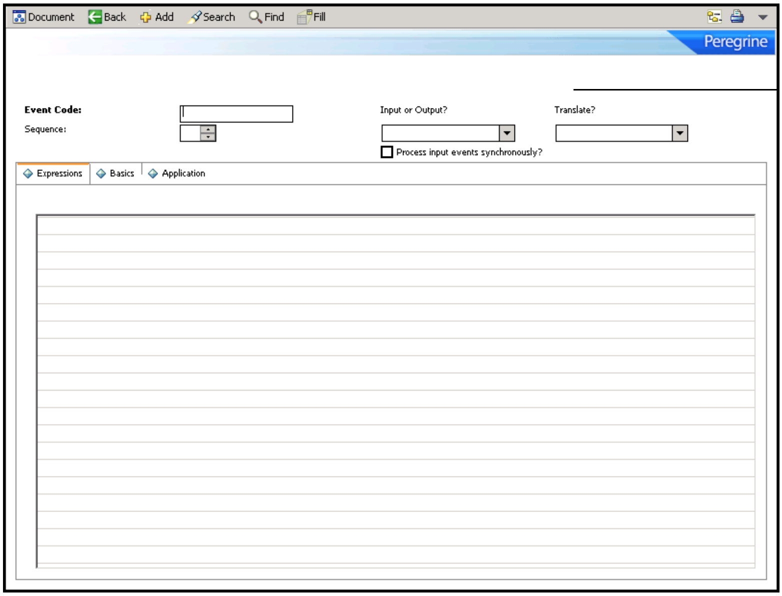
Event registration

All events are registered in the **eventregister** file. The **eventregister** file includes a unique event code as well as a sequence number, so a single event can execute a series of applications. In addition, it contains initialization statements, mapping information and instructions for calling the ServiceCenter application.

Reviewing event registration

To review event registration

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Registration** to open the Event Registration log.



This area contains the Header fields.

The form opens on the Expressions tab.

Event Registration fields

The encoded file names recorded in the eventregister file are for your reference only.

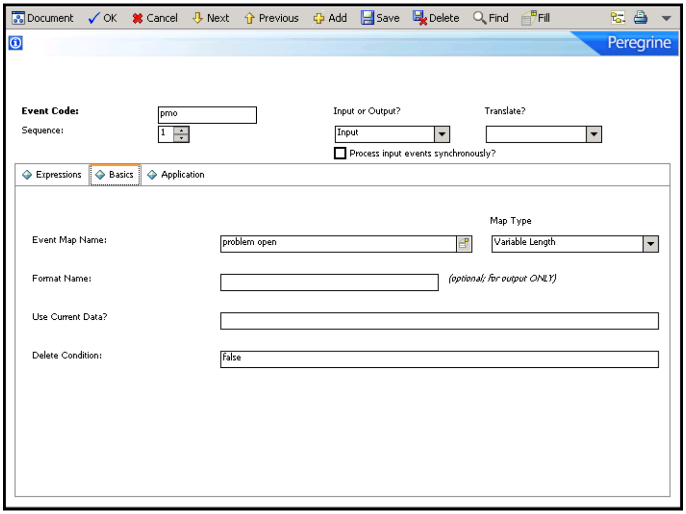
Header

Field	Description
Event Code (evtype)	Unique code that identifies this registration.
Sequence (evseq)	Number used to order the sequence of RAD applications to be executed for a single device type.
Input or Output (evftype)	Flag to identify whether this registration is for an input or an output transaction; only input or output is acceptable.
Translate (evtranslate)	Indicates whether to translate to upper (uc) or lower (lc) case; default is no translation.
Process input events synchronously? (synch.process)	When selected (true), prompts the system to process the event as soon as the record is added to the database, rather than waiting for the event background scheduler to wake up and process all events in the eventin queue.

Expressions tab

Field	Description
Expressions (evinit)	Array of statements that execute at run time to initialize variables or initiate action based on the contents of the data passed in the eventin ($axces) and the eventregister ($axces.register) records, and/or on global variables available at run time; the global variable $axces.fields represents an array of the fields passed in the evfield field of the eventin record.

Basics tab



Event Code: pmo

Sequence: 1

Input or Output? Input

Translate?

□ Process input events synchronously?

Event Map Name: problem open

Map Type: Variable Length

Format Name: (optional; for output ONLY)

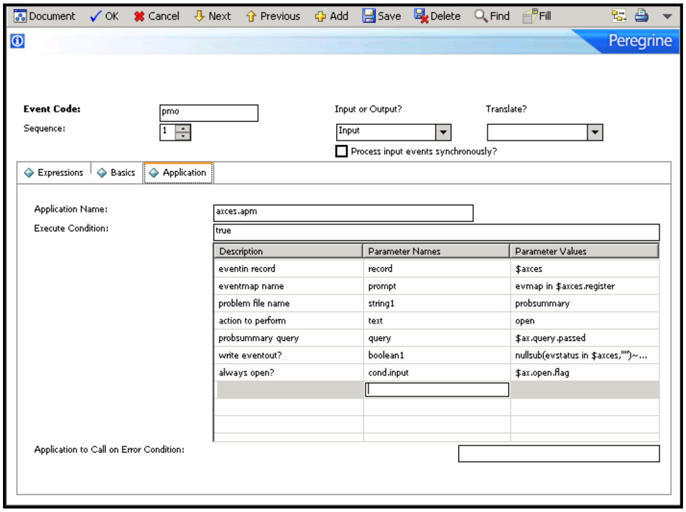
Use Current Data?

Delete Condition: false

Field	Description
Event Map Name (evmap)	Name of the event map to use.
Map Type (evmaptype)	Determines the length of the map. Use this field for incoming events only. With **Variable Length**, the Event Map Name is not used, and all the incoming data has no fixed length. With **Fixed Length**, the Event Map Name is used, and the length is determined by the mapping definitions.
Format Name (evformat)	Used only for output events, the name of the format that displays the record.

Field	Description
Use Current Data? (evnullsub)	If the condition is **true**, this always substitutes the current value in the target data when the external event passes a null value. For example, if an **icmu** event does not pass a value for **vendor** and the inventory item being updated has **Peregrine** in the **vendor** field, the result of mapping keeps **Peregrine** as the vendor. The event map allows specification of **evnullsub** on a field-by-field basis and overrides this default when set in an individual map record.
Delete Condition (evdelete)	A condition whose result determines whether to delete an **eventin** record after it is successfully processed.

Application tab



Event Code:	pmo	Input or Output?	Translate?
Sequence:	1	Input	
		□ Process input events synchronously?	

Application Name: axces.apm

Execute Condition: true

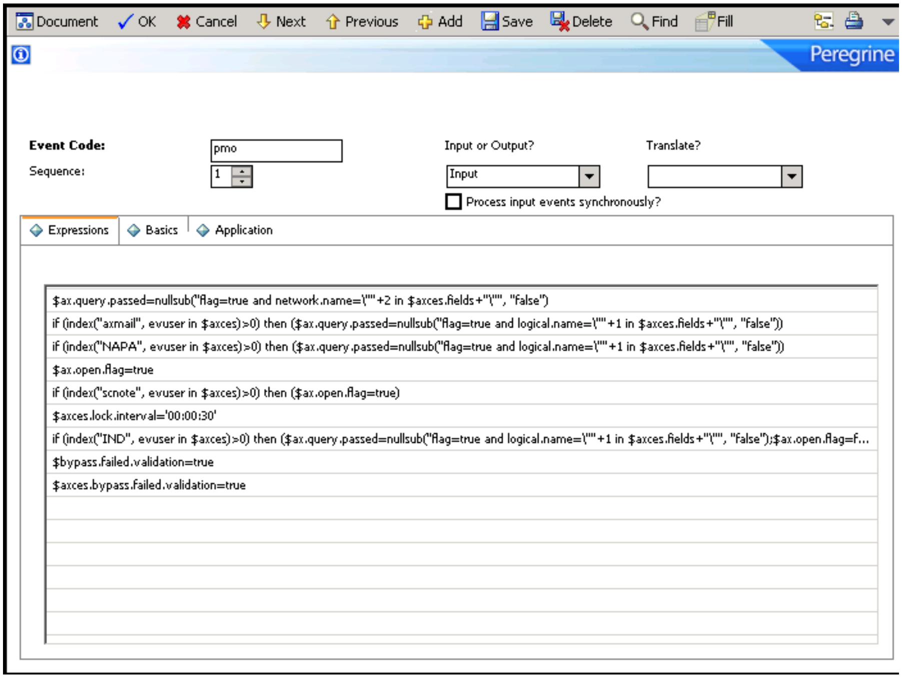
Description	Parameter Names	Parameter Values
eventin record	record	$axces
eventmap name	prompt	evmap in $axces.register
problem file name	string1	probsummary
action to perform	text	open
probsummary query	query	$ax.query.passed
write eventout?	boolean1	nullsub(evstatus in $axces,"")~...
always open?	cond.input	$ax.open.flag

Application to Call on Error Condition:

Field	Description
Application Name (evappl)	Name of the RAD application to execute.
Execute Condition (evcondition)	A condition that, if true, allows the RAD application to be executed.
Description (comments)	Array used to describe the elements in the Parameter Names and Parameter Values fields.

Field	Description
Parameter Names (names)	Array of parameter field names that pass to the RAD application; these names must exist in the application file.
Parameter Values (values)	Array of variables or literals that correspond to the list of parameter names passed in the names field; the data types must match.
Application to Call on Error Condition (evgoto)	Name of a RAD application to call after execution of the primary application if the primary application fails due to an error condition; parameters may not be passed as local variables.

Registration is necessary for all input events that external applications process. In the following example, event code pmo identifies opening an incident.

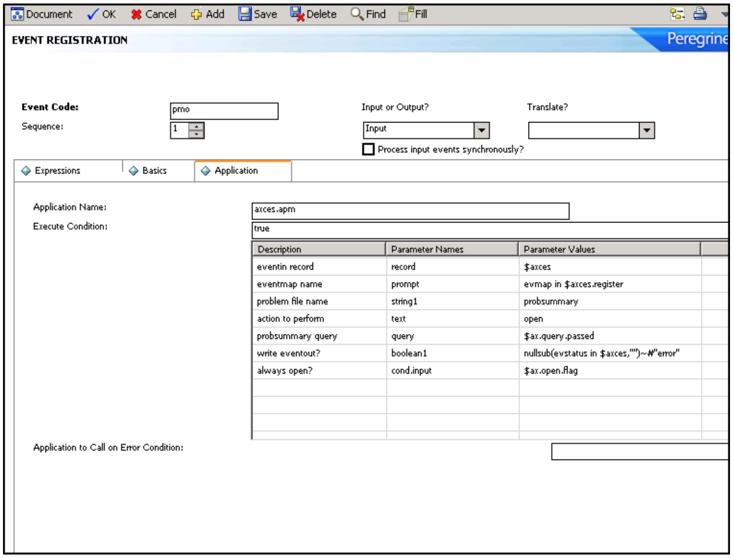


Event Code:	pmo	Input or Output?	Translate?
Sequence:	1	Input	
		☐ Process input events synchronously?	

```
$ax.query.passed=nullsub("flag=true and network.name=\""+2 in $axces.fields+"\"", "false")
if (index("axmail", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false"))
if (index("NAPA", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false"))
$ax.open.flag=true
if (index("scnote", evuser in $axces)>0) then ($ax.open.flag=true)
$axces.lock.interval='00:00:30'
if (index("IND", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false");$ax.open.flag=f...
$bypass.failed.validation=true
$axces.bypass.failed.validation=true
```

When a pmo event occurs, the system calls application axces.apm if the condition evaluates to true. The parameters are passed by name and value, just as they are in the operator record. The Event Map Name identifies the map to use.

The expression statements in the previous example set up different queries depending on the source of data. IPAS events depend on network.name, so the query uses network.name to select open incidents for update. The SCAuto mail incident event uses logical.name.



EVENT REGISTRATION

Event Code: pmo

Sequence: 1

Input or Output? Input

Translate?

□ Process input events synchronously?

Application Name: axces.apm

Execute Condition: true

Description	Parameter Names	Parameter Values
eventin record	record	$axces
eventmap name	prompt	evmap in $axces.register
problem file name	string1	probsummary
action to perform	text	open
probsummary query	query	$ax.query.passed
write eventout?	boolean1	nullsub(evstatus in $axces,"")~#"error"
always open?	cond.input	$ax.open.flag

Application to Call on Error Condition:

This registration record instructs Event Services to select a record from the probsummary file (based on the query in $ax.query.passed), then map data from the eventin record ($axces), based on the incident open (evmap in $axces.register) map record, then open an incident.

In most standard Event Services input applications, the first two parameters passed are the event record and the name of the event map. An exception in standard ServiceCenter SCAuto applications is email, which passes the mail record and the delimiter character. See *Appendix B, Common Events* for a list of commonly-used events.

Global variables

The following global variables are available when defining registration events:

Variable	Description
$axces	Represents the **eventin** record.
$axces.fields	Represents the **evlist** field in the **eventin** record.
$axces.register	Represents the event registration record.
$axces.lock.interval	An interval of time (for example, '00:02:00' for two minutes) after which a retry occurs if the attempt to update a problem is denied due to a lock.
$axces.debug	If set to **true**, the **evlist** array in the **eventin** record is not removed before attempting to update the record. If the size of the record exceeds 32 KB, an error is issued, the **eventin** record is NOT updated, and the event reprocesses (since the evtime field is not removed). **Use this feature with discretion.**
$axces.bypass.failed.validation	Used in events calling the application **axces.apm**. the default is **true**. When set to **true**, the application ignores any failed **formatctrl** validations. When set to **false**, the event status is "**error-fc**".

Note: The two additional standard events, **page** and **fax**, are not controlled through the registration table.

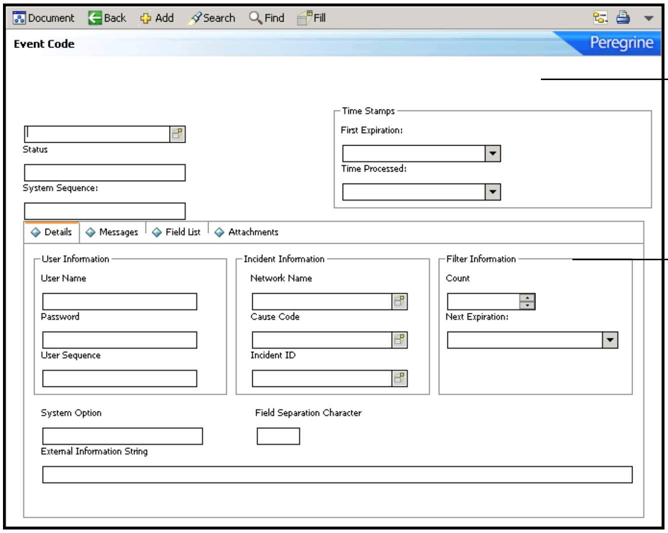
- A **fax** event is a report that uses the FAX config record's name as its printer name. The report writes to the **eventout** file and the external SCAuto application directs it as required.
- A **page** event is normally called as a Format Control subroutine based on conditions at problem open time.

Input events

The input event log file is called **eventin**. It contains a record for every event detected but not filtered by SCAuto external applications. The record must contain the event code, a unique system ID and a time stamp. Data passes to ServiceCenter in a character string using a delimiter character to separate fields.

To review input events

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Queues** > **Input Events** to open the **eventin** (**event.in.g**) form.



This area contains the Header fields.

The form opens on the Details tab.

2 Click **Search** to display a QBE list of all input events.

3 Double-click on an event to display the record.

Input fields

The following tables contain the event fields found on the form and their corresponding properties. The encoded field names recorded in the **eventin** file are for your reference only.

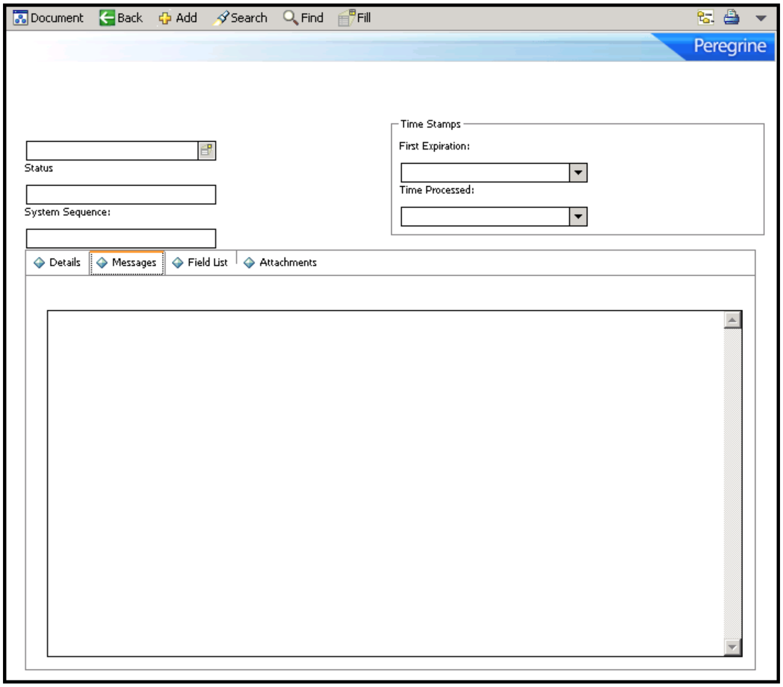
Header

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>Event Code (evtype)</td><td>The registration name for the event (required).</td></tr>
<tr><td>Status (evstatus)</td><td>The result of the Event Manager action. If events are not deleted after processing, ServiceCenter automatically assigns one of the following statuses to each:
<table>
<tr><th>Status</th><th>Description</th></tr>
<tr><td>added</td><td>An inventory item has been added to the database; the device’s name is in the <b>Network Name</b> field.</td></tr>
<tr><td>closed</td><td>An incident has been closed; the Incident Number is in the <b>Incident ID</b> field.</td></tr>
<tr><td>deleted</td><td>An inventory item has been marked for deletion in the database; the device’s name is in the <b>Network Name</b> field.</td></tr>
<tr><td>error</td><td>An error occurred while processing the event. This status is assigned by</td></tr>
<tr><td>locked</td><td>The record to be updated or deleted was locked.</td></tr>
<tr><td>filtered</td><td>An incident was filtered, and is waiting for the filter condition to be satisfied.</td></tr>
<tr><td>mailed</td><td>Electronic mail has been sent.</td></tr>
<tr><td>opened</td><td>An incident has been opened; the Incident Number is in the <b>Incident ID</b> field.</td></tr>
<tr><td>processed</td><td>A software inventory item or change has been successfully processed.</td></tr>
</table>
</td></tr>
<tr><td>System Sequence (evsysseq)</td><td>System-assigned sequence number, for event tracking (system provided).</td></tr>
<tr><td>First Expiration (evtime)</td><td>The time the event occurred (required).</td></tr>
<tr><td>Time Processed (evtimestamp)</td><td>The system time translation of the actual time that ServiceCenter processed the event.</td></tr>
</table>

Details tab

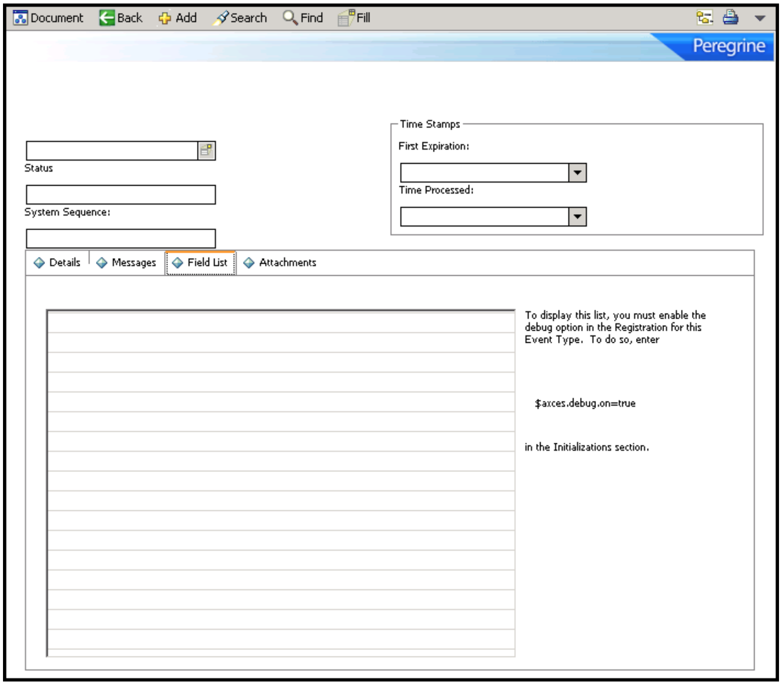
Field	Description
User Name (evuser)	The event user name; if passed, it is the operator name (optional).
Password (evpswd)	The event user's password (optional).
User Sequence (evusrseq)	User-assigned sequence number, used to trace an event through the ServiceCenter system (for example, an external reference number; optional).
Network Name (evnetnm)	Used in filtering, the unique network name of a device (system defined by Event Services).
Cause Code (evcode)	Used in filtering, an event code sent to Event Manager (system defined by Event Services).
Incident ID (evid)	Problem character ID; used in filtering (system defined by Event Services).
Count (evcount)	Used in filtering, the number of events for a particular transaction (system defined by Event Services).
Next Expiration (evexpire)	Used in filtering, the time when an incident is opened (system assigned by Event Services).
System Option (evsysopt)	Code to identify system options (optional).
Field Separation Character (evsepchar)	Character used to separate fields in the **evfields** field (substitutes ^ if null).
External Information String (evfields)	Data describing the event, with fields separated by the **evsepchar** character; specific positions in the **evfields** field are reserved for application dependent data. For example: `falcon^max@peregrine.com^falcon;susie;root^ Re:meeting this afternoon^Tuesday, 23 January 2004 16:41:07` In this example, max@peregrine sends falcon an e-mail, with carbon copies to falcon, susie, and root. The subject is Meeting this afternoon, and the text follows the subject. **Note:** The first line of text always includes the date and time the message was sent. Each of the data fields is separated by a separation character, or delimiter, that is defined in the registration file. If no delimiter is defined, ^ is the default.

Messages tab



Field	Description
Messages (evmsg)	Any messages generated during event processing.

Field List tab



To display this list, you must enable the debug option in the Registration for this Event Type. To do so, enter

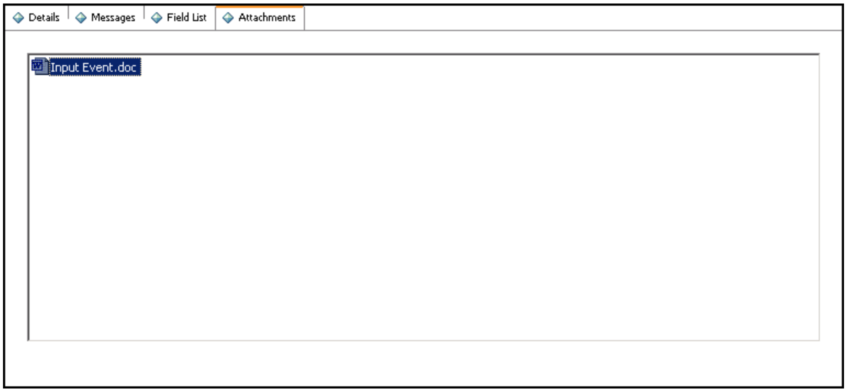
$axces.debug.on=true

in the Initializations section.

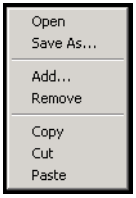
Field	Description
Field List (evlist)	Array, built by the Event Manager, of the fields in the **evfield** field; available to **eventmap** as **$axces.fields**.

Note: The evlist field refreshes in the application after use. If you need to view it for debugging or trace purposes, you must set **$axces.debug=true** in your event registration initialization expressions. The maximum size of the evfields data is 16,000 bytes. Use this feature with discretion because e-mail messages are often quite large.

Attachments tab



The Attachments tab is an OLE container where you can insert various objects related to the Input record. To insert files, right-click the mouse and select the appropriate command from the pop-up menu.



To perform maintenance tasks on an object in the tab, select the object and right-click the mouse. Select the appropriate command from the pop-up menu.

Input event processing

An external application, such as SCAuto/SDK or SCAuto for NetView OS/390, adds all records in the **eventin** file. External programs manipulate the **eventin** records.

For example, SCAuto supports an event called **email**. Electronic mail can be received from external sources and passed to ServiceCenter mail. The sources for electronic mail can be external e-mail systems, alert monitors, or other programs that can send messages. The external SCAuto application packages the data in a standard format and adds it to the **eventin** file. The format is defined in **eventmap** records.

Note: Records in the eventin file that have been processed do not contain a **First Expiration** value in the upper right field.

Normally, events are deleted after they have been processed unless they have been filtered or an exception has occurred during processing. The delete flag is controlled by a condition set in the eventregister file.

If an error occurs due to Format Control processing, event processing terminates for that event and the specific error message writes to the eventin's Messages and to ServiceCenter msglog file.

Once you install and test SCAuto, do one of the following:

- Set all delete flags in the registration records to true.
- Use the ServiceCenter **purge/archive** routines to schedule cleaning up the file on a regular basis.

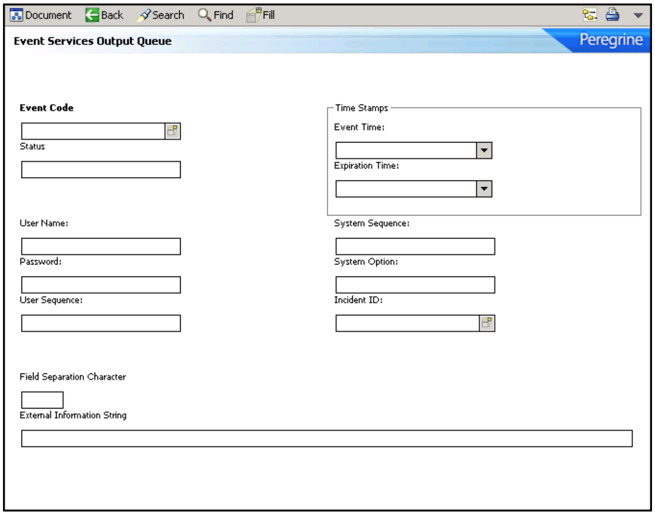
Refer to the *Administering ServiceCenter* online help for more information about the **Purge/Archive** Utility.

Output events

The output event log is called **eventout**. It contains a record for each event processed by Event Services applications and instructions that external software (for example, pager numbers to notify service technicians) uses. Data passes to external applications in a character string using a delimiter character to separate fields.

To review output events

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Queues** > **Output Events** to open the eventout (event.out.g) form.



2 Click **Search** to display a QBE list of current output events.

3 Double-click an event to display the record.

Output fields

The encoded field names recorded in the **eventout** file are for reference only.

Field	Description
Event Code (evtype)	Registration name for the event (required).
Status (evstatus)	Result of the action Event Manager; the actions are: opened, updated, closed, added, deleted, filtered, or error.

Field	Description
Event Time (evtime)	Time the event occurred.
Expiration Time (evexpire)	Expiration time for an event. The time when the event scheduler processes the **eventout** record; if the field is NULL, no processing occurs.
User Name (evuser)	Event user name (optional).
Password (evpswd)	Event user's password (optional).
User Sequence (evusrseq)	User-defined sequence number for event tracking.
System Sequence (evsysseq)	System-assigned sequence number, for event tracking (system provided); used when external software restarts the **eventout** monitoring pointer.
System Option (evsysopt)	Code to identify system options (optional).
Incident ID (evid)	Problem character ID (incident number).
Field Separator Character (evsepchar)	Character that separates fields in the **evfields** field (substitutes ^ if null).
External Information String (evfields)	Data describing the event where the **evsepchar** character separates the fields. For example: `joe.employee@peregrine.com^FALCON^Joe User^ Your ServiceCenter password has been updated by FALCON.` In this example, ServiceCenter user FALCON is sending e-mail to joe.employee@peregrine.com.

Records in the **eventout** file that are processed do not contain an expiration date. Normally, events are deleted from the **eventout** file after processing unless an error occurs. A flag in the external IPAS or SCAuto software can be manipulated to cause record deletion after read; however, since multiple SCAuto processes can read the same record, it is not always feasible to delete on read.

Note: Use the ServiceCenter **purge/archive** routines to schedule cleaning up the **eventout** file on a regular basis. Refer to the *Administering ServiceCenter* online help for more information about the **Purge/Archive** Utility.

External programs manipulate the **eventout** records.

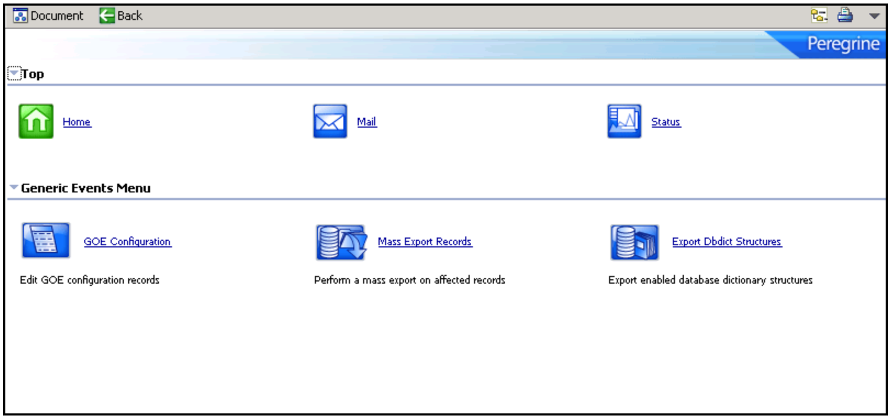
Generic Event Administration

The controls under this option allow for administration of outgoing event records into Connect-It, including the following:

- Editing **eventout** information generation
- Exporting configuration records
- Exporting Database Dictionary structures

To access these event controls

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Generic Event Administration** to open the Generic Event Administration menu.



For details about the Connect-It product, see the Connect-It documentation.

3 Mapping and Filtering

CHAPTER

Once events have been created coming in or out of ServiceCenter, processes need to be set into place to manage and direct the events. Event mapping and event filtering take the event and its constituent data, and direct it in specified ways to create results within other areas of the system.

This chapter describes these processes in the following sections:

- *Mapping* on page 44
- *Event filters* on page 64

Mapping

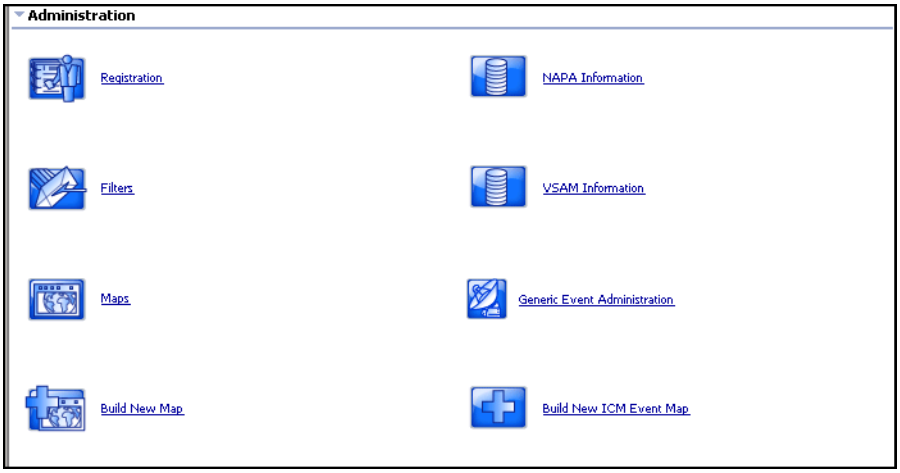
Event mapping information is stored in the **eventmap** file. The two types of maps are input maps and output maps. Input maps contain instructions for moving data from the **eventin** record's **External Information String** (*evfields)* field to the target file, while output maps move information from the source file to the **eventout** record's **External Information String** field.

Important: Event Maps provided with Event Services describe standard events. Changing the relative position of data in the information exchanged between ServiceCenter and the external applications (for example, IPAS) may cause standard events to fail. Create new maps for non-standard events rather than modifying existing maps.

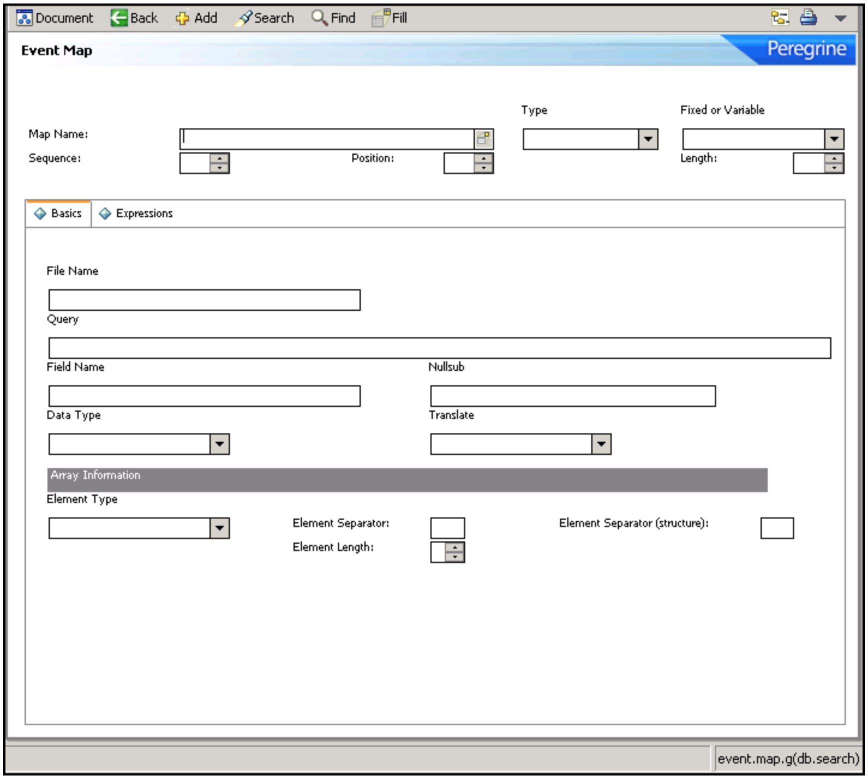
The Event Map form

To review event maps

1 From the ServiceCenter main menu, click **Utilities** > **Event Services**.



2 From the Administration menu, click **Maps** to open the Event Map form.



Header fields

Encoded field input names recorded in the **eventmap** file are included in parenthesis for reference only.

Field	Description
Map Name (evmap)	A unique name that identifies each map; combined with the **evseq** field and the **evtype** field, comprises the unique key.
Type (evtype)	A flag to identify whether this registration is for an input or an output transaction; only input or output are acceptable values.
Fixed or Variable (evmaptyp)	Either Fixed Length or Variable Length; indicates the format of data passed in **eventin** record; default is variable with a delimiter between fields.

Field	Description
Sequence (evseq)	Number indicating the sequence in which data is mapped from the **eventin** record to the target record; when multiple files are updated, certain dependencies may exist that necessitate a prescribed order for field mapping; used in **icm*** maps.
Position (evindex)	Number corresponding to the relative position of data in the **eventin** record's **evfields** field.
Length (evlength)	If **evmaptyp** is Fixed Length, you must provide the length of the field.

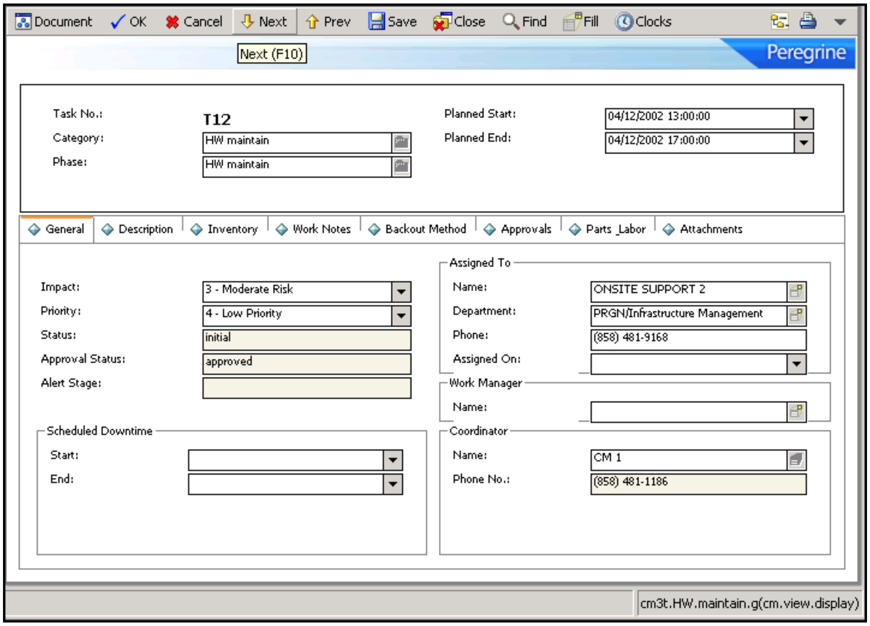
Basic tab fields

Field	Description
File Name (evfile)	Name of the file from (for output) or into (for input) which data is mapped.
Query (evquery)	Query used to select a record from the named file if the file name is different from the one currently in use (that is, the sequence number changes); allows update of multiple files with a single map.
Field Name (evfield)	Name of the field from (for output) or into (for input) which data is mapped.
Nullsub (evnullsub)	Value in this field replaces the contents of the source field if NULL. To keep the value present in a record that is being updated, enter **$axces.field** in the Nullsub field. You can set a global condition to keep the value present in a record that is being updated by setting the **Use Current Data?** condition in the event registration record to **true**.
Data Type (evdtype)	Data type of the field being mapped; the Build Event Maps process sets this value and is automatically set when the event map record is being added or updated. If **evdtype** is Array, you must complete the appropriate fields in the Array Information section of the form.
Translate (evxlate)	Indicates whether to translate the field value to uppercase (**uc**) or lowercase (**lc**); the default is to not translate.
Element Type (eveltype)	Data type of array elements. If **eveltype** is Structure, you must enter a different separator for the **evsepchar** and **evsepchar.struc** fields. If **eveltype** contains a value other than Structure, you must enter a value for either the **evsepchar** or the **evitmlng** field.
Element Separator (evsepchar)	Separation character to use for elements in array-type fields; the default is the **|** (pipe symbol).
Element Length (evitmlng)	If not NULL, defines the length of each element in array type fields. **Note:** This field does not apply if **eveltype** is Structure.
Element Separator (structure) (evsepchar.struc)	Separation character to use for the subelements within the structure of an array of structures; the default is the ` (grave accent).

Mapping arrays of structures

The following example shows an event string that maps information into an existing Change Management task, T12. The last portion of the string maps data to the **parts** and **labor** arrays of structures fields in the **cm3t** table. The array separator | (pipe symbol) delimits each array. Each portion of the array with the | delimiter has another subdivision using the structure separator ` (grave accent).

```
T^^update^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^THIS IS A
TEST^^^^^^^ ^^^^^^^^^^^^20^01/01/01 12:41:23`AAAA`11111`|```
|02/02/02``22222`^03/01/02` falcon`3.5`ACME US`
```



Task No.: T12
Category: HW maintain
Phase: HW maintain
Planned Start: 04/12/2002 13:00:00
Planned End: 04/12/2002 17:00:00

Impact: 3 - Moderate Risk
Priority: 4 - Low Priority
Status: initial
Approval Status: approved
Alert Stage:

Scheduled Downtime
Start:
End:

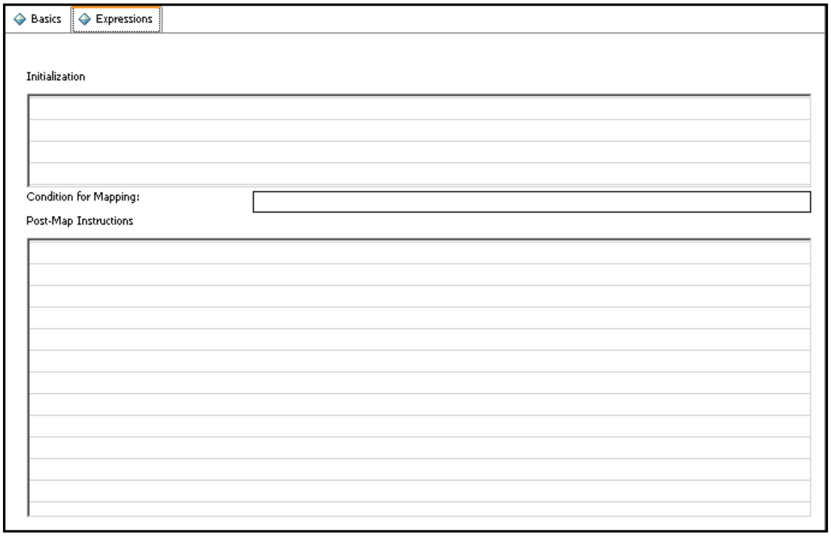
Assigned To
Name: ONSITE SUPPORT 2
Department: PRGN/Infrastructure Management
Phone: (858) 481-9168
Assigned On:

Work Manager
Name:

Coordinator
Name: CM 1
Phone No.: (858) 481-1186

cm3t.HW.maintain.g(cm.view.display)

Expressions tab fields

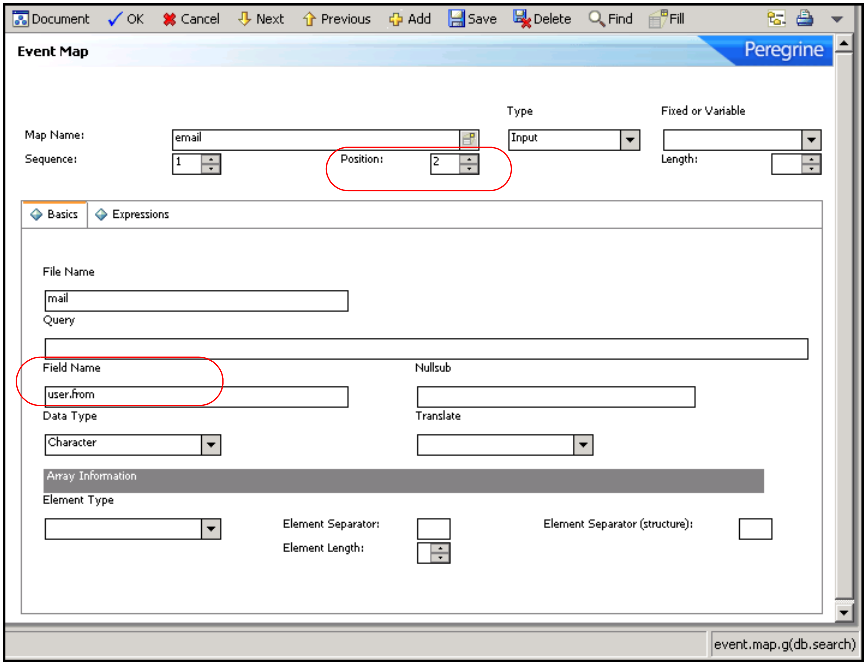


Field	Description
Initialization (evinit)	Array of statements that are executed at run time to initialize variables or initiate action based on the contents of the data passed in the eventin record and/or on global variables available at run time; the global variable $axces.fields represents an array of the fields passed in the **evfield** field of the eventin record.
Condition for Mapping (evmapcond)	Condition that, if true, allows the data to be mapped.
Post-Map Instructions (evcalc)	Array of expressions that are evaluated at run time to execute processing statements after the field is mapped.

Using Event Maps

Each record in the **eventmap** file describes a single field. Event Services uses this information to map data from external sources to ServiceCenter files, and data in ServiceCenter files to a sequence of delimited fields for export to external applications.

For example, when a ServiceCenter user sends mail, certain fields in the ServiceCenter **mail** file are populated. These include **user.to, user.from, user.array, subject** and **text**. When sending e-mail, you must map the information in these fields in a standard, defined sequence so that the SCAuto mail application can translate it to external programs. Likewise, when SCAuto receives mail from an external program and posts it to the **eventin** file, the Event Services application populates the appropriate fields in the ServiceCenter **mail** file.



Map Name: email

Sequence: 1

Position: 2

Type: Input

Fixed or Variable:

Length:

File Name: mail

Query:

Field Name: user.from

Nullsub:

Data Type: Character

Translate:

Element Type:

Element Separator:

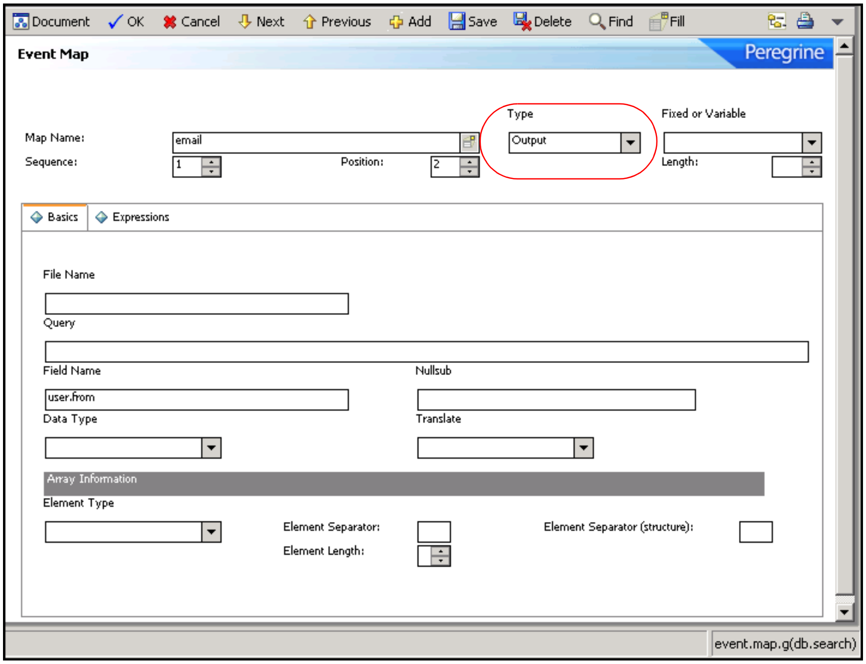
Element Length:

Element Separator (structure):

event.map.g(db.search)

As shown in this record, the **user.from** field in the ServiceCenter **mail** file has a position of 2, and is the second field in the delimited text string written to the **eventin** record’s **Field List**.

For output, the contents of the **user.from** field in the ServiceCenter **mail** file is placed in the second position in the **External Information String** field of the **eventout** record. The **Type** field is changed to **output**.



Map Name: email

Sequence: 1

Position: 2

Type: Output

Fixed or Variable:

Length:

File Name:

Query:

Field Name: user.from

Nullsub:

Data Type:

Translate:

Element Type:

Element Separator:

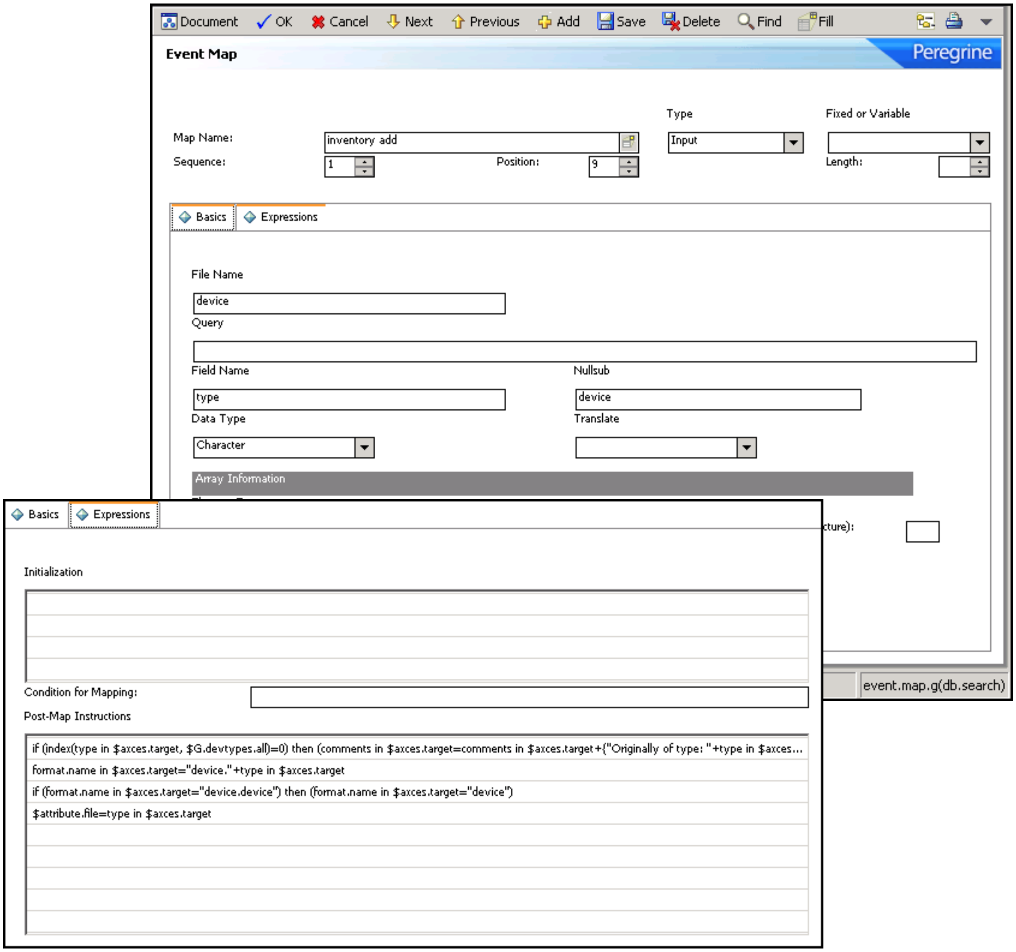
Element Length:

Element Separator (structure):

event.map.g(db.search)

If the mapping records for e-mail are deleted, ServiceCenter uses the default as shown in this Output example that the system provides when you install Event Services.

Event Services also handles mapping to multiple files. For example, SCAuto for NetView OS/390 and SCAuto can send inventory information that is stored in more than one file. The ICM applications use two files to describe each device: the **entity** file and the **attribute** file. The entity file is called **device**; the **attribute** file depends upon the device type, and is identified by the **type** field in the **entity** file. When inventory information is gathered using discovery processes in external applications such as OpenView and passed to ServiceCenter via SCAuto, both files are updated.



Map Name: inventory add

Sequence: 1

Position: 9

Type: Input

Fixed or Variable:

Length:

File Name: device

Query:

Field Name: type

Nullsub: device

Data Type: Character

Translate:

Initialization:

Condition for Mapping:

Post-Map Instructions

```
if (index(type in $axces.target, $G.devtypes.all)=0) then (comments in $axces.target=comments in $axces.target+{"Originally of type: "+type in $axces...
format.name in $axces.target="device."+type in $axces.target
if (format.name in $axces.target="device.device") then (format.name in $axces.target="device")
$attribute.file=type in $axces.target
```

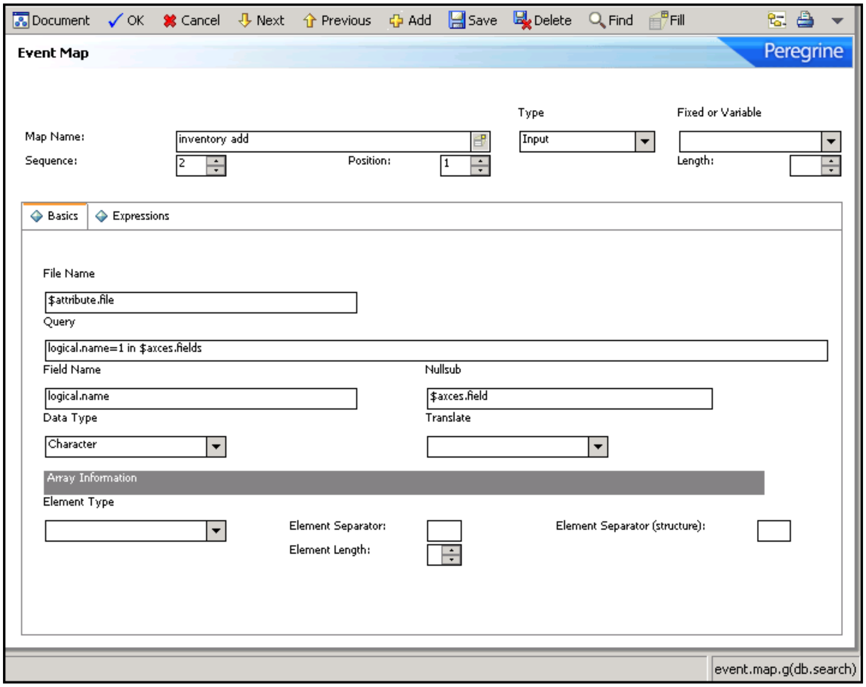
The first step in preparing to map multiple files is to identify the attribute file. This is done using an expression (see line 4) in the **Post Map Instructions** to set the variable **$attribute.file** to the value in the **type** field of the **device** (TARGET) record.

Note: The **Sequence** is 1, and the **File Name** in the map record is **device**.

Until all fields are mapped to the **device** file, **Sequence** remains **1** and **File Name** remains **device**. The query passed in the Registration file already selected the record, therefore no query is necessary.

After the last field for the initial file is mapped, the record is added or updated and a new file is initialized based on the value of **$attribute.file**.

Note: While **$axces.target** and**$axces.field** have special meaning within Event Services, **$attribute.file** is an arbitrary global variable name.



Event Map

Map Name: inventory add

Sequence: 2

Position: 1

Type: Input

Fixed or Variable:

Length:

File Name: $attribute.file

Query: logical.name=1 in $axces.fields

Field Name: logical.name

Nullsub: $axces.field

Data Type: Character

Translate:

Array Information

Element Type:

Element Separator:

Element Separator (structure):

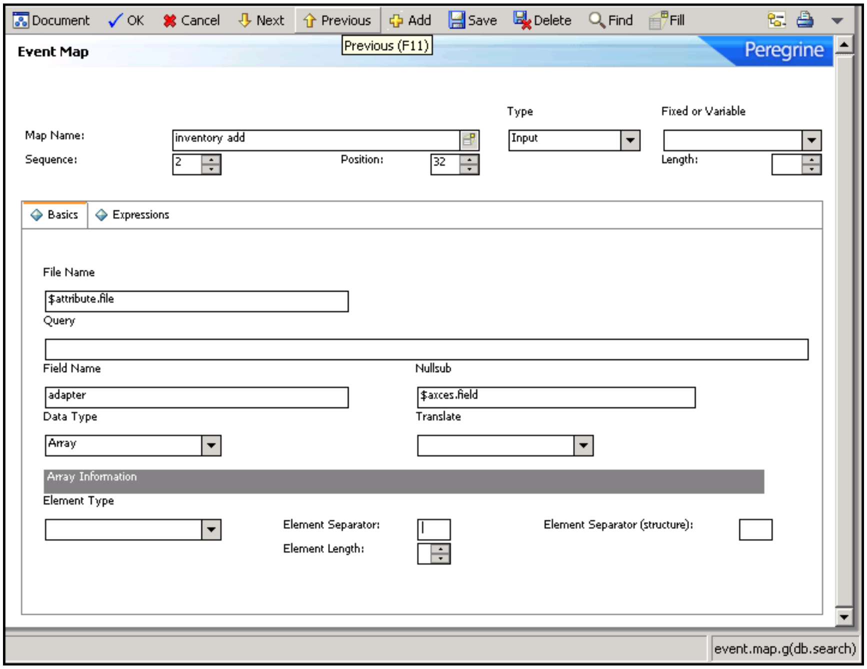
Element Length:

event.map.g(db.search)

When all fields are mapped into the device file, the next map record has a **Sequence** of 2, the File Name is different and a Query is supplied.

- **File Name** now contains the value assigned to the $attribute.file variable.
- **Query** tells Event Services how to select the record to update from the file identified by $attribute.file. The query can be either a literal statement (as shown in the previous example) or a variable set in previous **Post Map Instruction** or **Initialization** fields.

The first mapping for the new file is logical.name, which is stored in **Position 1** (as shown in the previous example) of the **evfields** array field, which is itself represented by the $axces.fields variable in the eventin record.

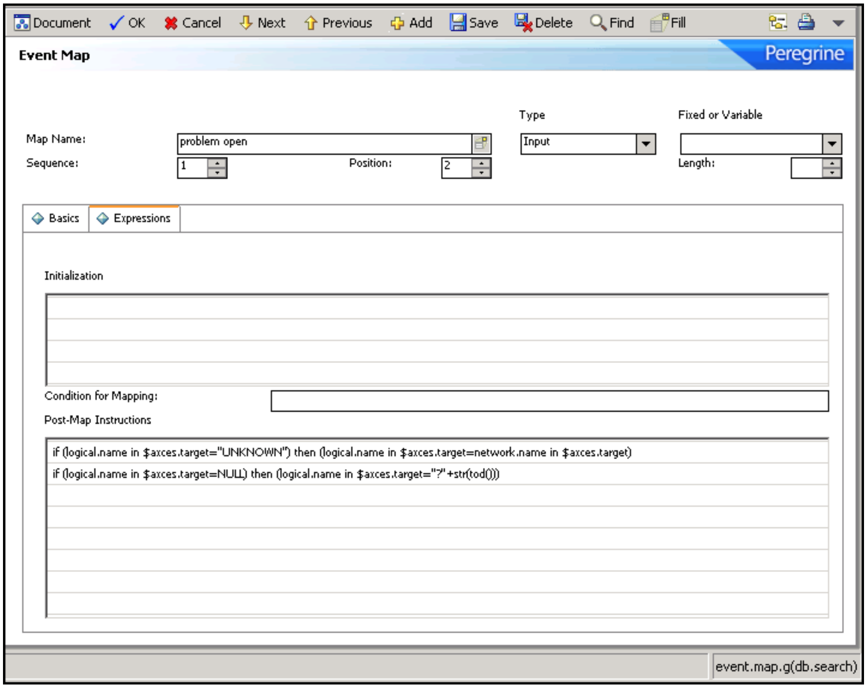


Map Name: inventory add
Sequence: 2
Position: 32
Type: Input
Fixed or Variable:
Length:
File Name: $attribute.file
Query:
Field Name: adapter
Nullsub: $axces.field
Data Type: Array
Translate:
Element Type:
Element Separator:
Element Length:
Element Separator (structure):
event.map.g(db.search)

Subsequent map records move data from the eventin record to the new file.

Note: When updating an existing record, Event Services substitutes the value in the original record for a null value passed from the eventin record.

Mapping also allows complete flexibility of data manipulation during the mapping process. Because Event Services runs as a background task, no input/output routines are available for online validation with user feedback, but you can check field values and make substitutions based on processing statements.



Map Name: problem open
Sequence: 1
Position: 2
Type: Input
Fixed or Variable:
Length:

Initialization

Condition for Mapping:

Post-Map Instructions

```
if (logical.name in $axces.target="UNKNOWN") then (logical.name in $axces.target=network.name in $axces.target)
if (logical.name in $axces.target=NULL) then (logical.name in $axces.target="?"+str(tod()))
```

In the preceding record, the value in network.name replaces logical.name if logical.name is UNKNOWN. The second statement sets logical.name to a constant if it is NULL.

Other common uses for expressions are to set the value of a field to the current date and time and to calculate a value based on information in the record. Event Services applications handles data type and case conversions as long as the **Field Type** field is correctly identified and the data is written to the descriptor structure.

Note: You can use a single Format Control record named login.event to establish initial global variables (such as lists of valid operators) when the event agent is started, just as you can for users when they log into ServiceCenter.

Important: If you are writing data to a field whose name exists in more than one structure in a record, you must explicitly name the field. For example, if you add a field named **assignment** to the **middle** structure of your incident Database Dictionary record and you want to manipulate that field, you must identify it as **middle,assignment**. The field must exist in the target file before any instruction can manipulate it. Make sure the data type is correctly identified.

Note: Event Services data type conversions occur for **character**, **number**, **date/time**, **logical**, and **array** fields only.

Global variables

The following global variables are active when mapping event data.

Variable	Description
$axces	Represents the eventin record.
$axces.fields	Represents the evlist field in the eventin record.
$axces.field	Value of a field in the target record at the time the target record is selected and before information is mapped to it from the event.
$axces.register	Represents the event registration record.
$axces.source	Map record.
$axces.target	Record into which data is mapped; the record selected from the ServiceCenter database to which event information is posted.
$axces.notriml	If set to true, any blank spaces or tabs at the end of the field are not removed.
$axces.notrimr	If set to true, any blank spaces or tabs at the beginning of the field are not removed.

Note: When e-mail events are sent to ServiceCenter, the text field’s leading and/or trailing spaces and tabs are not removed.

Mapping considerations for Inventory Management

While ServiceCenter provides both an entity file (device) and attribute files (for example, server), it is not necessary that both files exist to represent the characteristics of every device type. You can often fully describe a device using only the fields in the device file.

The map record for the **type** field (field #9 in standard events) defines how ServiceCenter selects and displays information about a device once the data is added. The **type** field in the device file refers directly to the associated attribute file of each device. If there is no attribute file associated with a particular device, the **type** field must contain device or be empty (NULL).

Similarly, the **format.name** field in the device record defines the name of the form that displays the device within ServiceCenter and, by extension, the name of the join file that temporarily stores information for review and update. The formatctrl record for the format name stored in the device record must contain **device** as the file name for all device types that do not have associated attribute files.

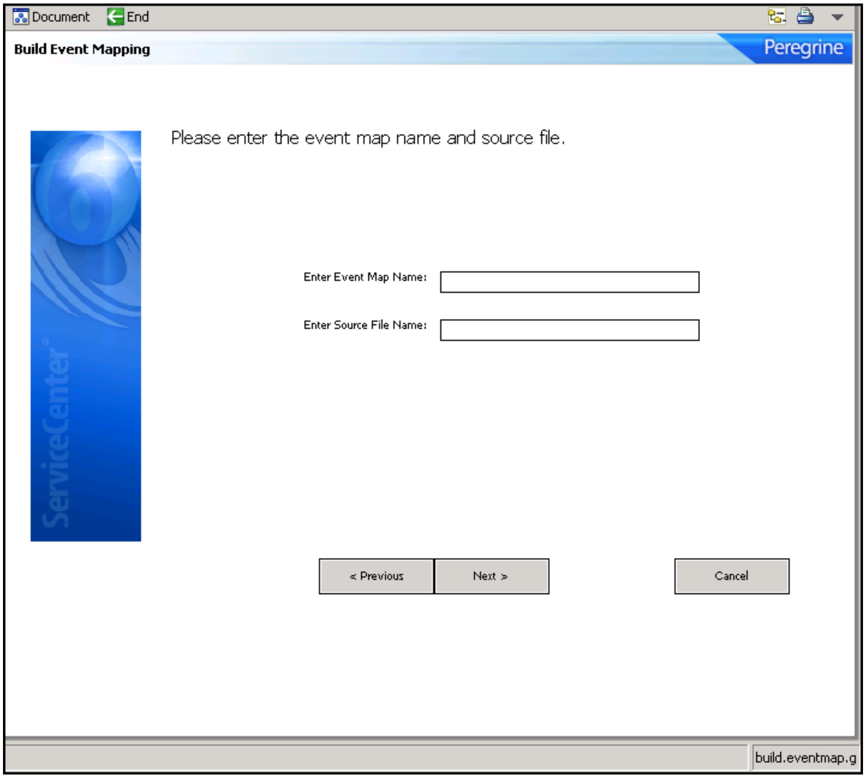
If an external agent detects an unknown device type, ServiceCenter processes the event, updating the device file with the information provided. If no attribute file exists for that device type, a Warning message is written to the event's Message list but the device is still added or updated in ServiceCenter's data repository. If event mapping indicates processing in more than one table, but the number of fields passed to the event is less than the position of the first field in the second table, there is no attempt to open the second table.

Building a new Event Map

You can build both input and output event maps for any file in ServiceCenter.

To build a new map

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Build New Map** to open the Build Event Mapping form.

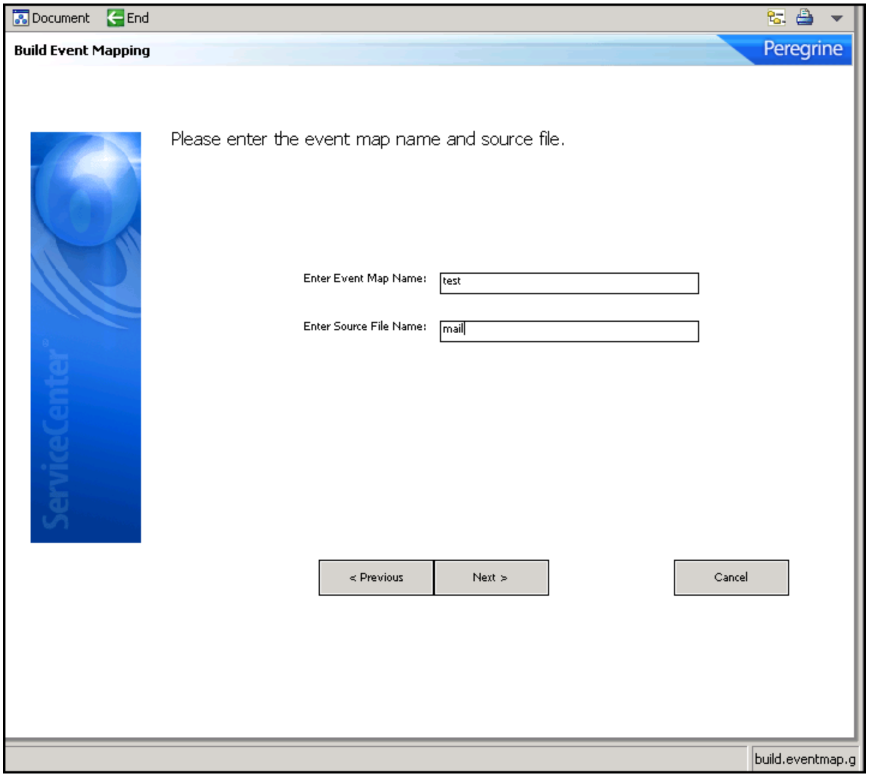


Please enter the event map name and source file.

Enter Event Map Name:

Enter Source File Name:

2 Type the **Map Name** and a **Source** file name.



Build Event Mapping

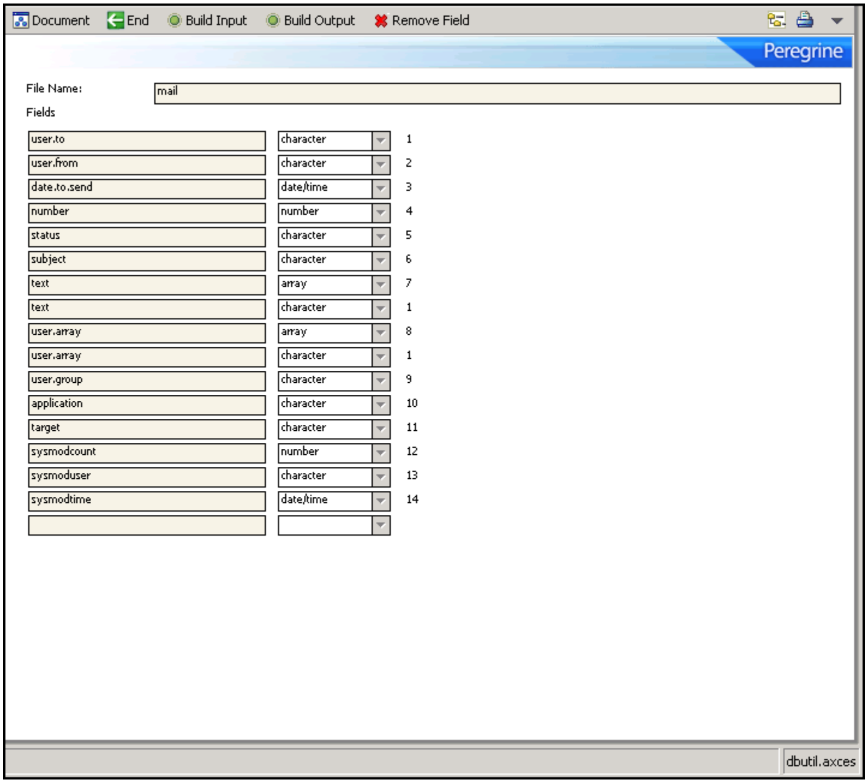
Please enter the event map name and source file.

Enter Event Map Name: test

Enter Source File Name: mail

Each mapping must have a unique name.

3 Press **Enter** to open a list of field names and data types for the file you selected.



File Name: mail

Fields

user.to	character	1
user.from	character	2
date.to.send	date/time	3
number	number	4
status	character	5
subject	character	6
text	array	7
text	character	1
user.array	array	8
user.array	character	1
user.group	character	9
application	character	10
target	character	11
sysmodcount	number	12
sysmoduser	character	13
sysmodtime	date/time	14

If you do not provide a source file name, ServiceCenter displays a QBE list of files where you can make a selection.

Warning: ServiceCenter issues a warning if the event map name already exists. In this case, building a new input map overwrites an existing input map and building a new output map overwrites an existing output map. If an input map exists and you are building an output map of the same name (or vice versa), the existing map is not removed.

The top of the screen contains buttons that allow you to manipulate and build a map record.

Button	Property
Build Input	Builds the records that map information from the eventin file to the selected ServiceCenter file.
Build Output	Builds the records to map information from the selected ServiceCenter file to a formatted string to be passed to SCAuto using the eventout file.
Remove Field	Deletes fields before a map is created. Place the cursor in the field you want to remove and click **Remove Fields**. Repeat this action for each field you want to remove.

Note: If an array field is part of your mapping, delete the second instance of the field in the list presented when building a new map, leaving only the array field.

Rules for building maps

The purpose of event mapping is to relate elements in a list to fields in a record. An external event, such as SCAutomate, or SCAuto for NetView OS/390, passes data into the ServiceCenter eventin file in a field called **fields**. Each element is separated from the others with a delimiter, or separation character. In the following example, the ^ character separates the five fields.

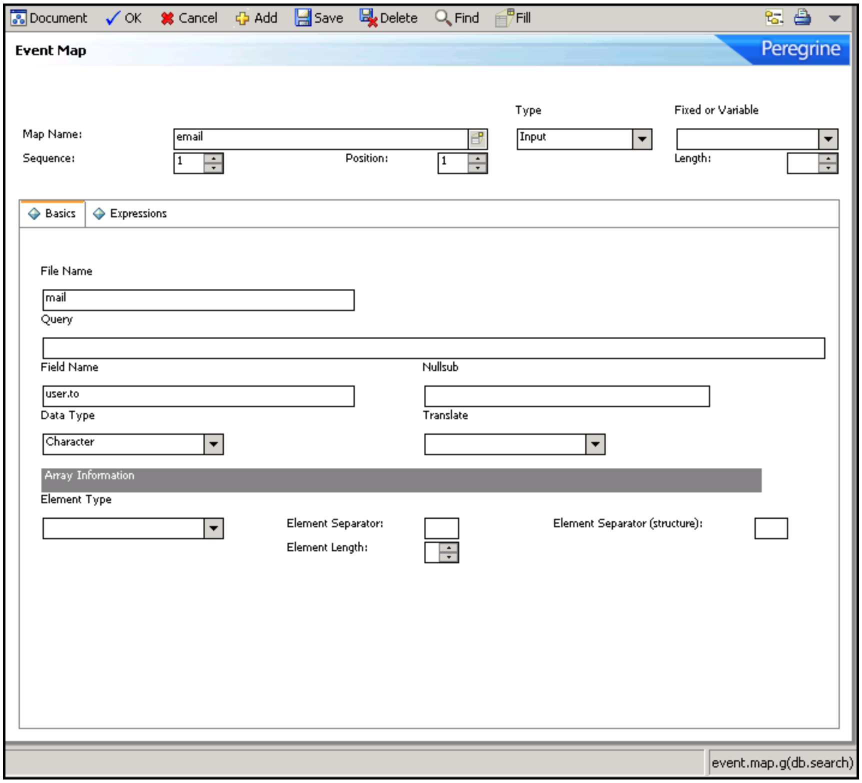
```
john@peregrine^falcon^toby;al;joe^Meeting today^Tue 12 Aug
```

Internally, Event Services converts this string to a list ($axces.fields):

```
john@peregrine
falcon
toby;al;joe
Meeting today
Tue 12 Augcol
```

The event processor assumes that fields with a type of date/time are in the time zone of the ServiceCenter system (that is, the time zone defined in the System Wide Company Record). If the event background process has its own operator record, that operator's time zone is used. For synchronous processing, the session processing the event handles the date/time in the time zone where it is defined.

Mapping defines the link between the elements in the internal list (**evlist**) and fields in a ServiceCenter file. The first field, **john@peregrine**, is mapped to the **mail** file's **user.to** field.



Map Name: email
Type: Input
Fixed or Variable:
Sequence: 1
Position: 1
Length:
File Name: mail
Query:
Field Name: user.to
Nullsub:
Data Type: Character
Translate:
Element Type:
Element Separator:
Element Separator (structure):
Element Length:

event.map.g(db.search)

For best results when building new maps that use array fields, follow these guidelines:

- Select the first instance of any array fields (such as **user.array** in the **mail** file) so the proper type is built for the field.
- Only scalar and array fields can be directly mapped; all other types must be manipulated using expressions.

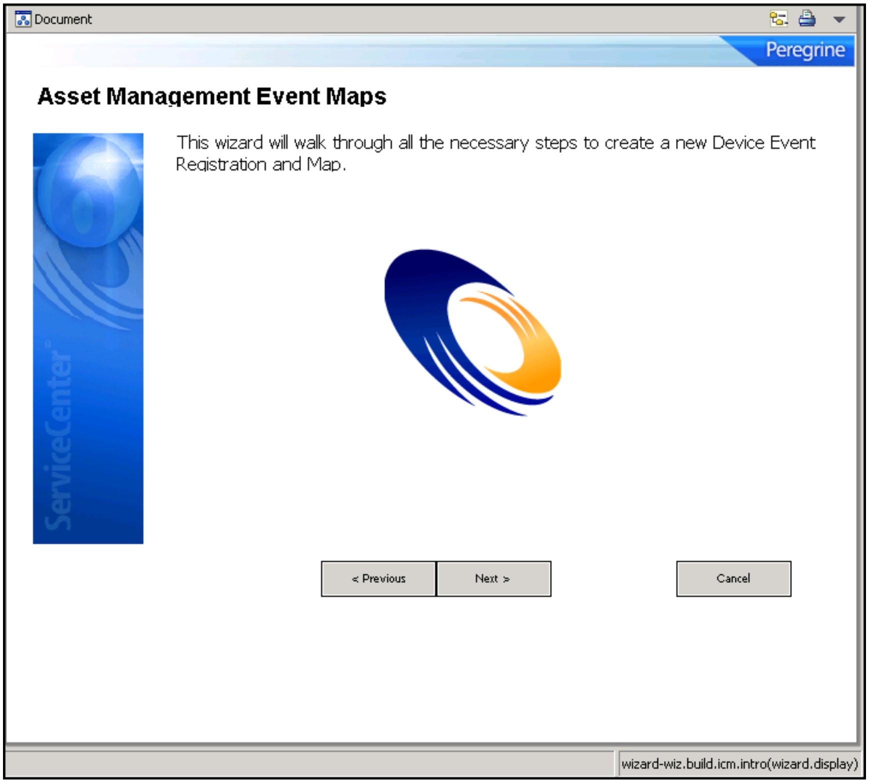
If possible, build maps first and then design external applications to use the maps.

Building a new ICM Event Map

This option enables the generation of event registrations and maps based on the actual field names that exist for a particular device type. These do not supersede the existing ICM events. They are a different way of processing the ICM data that can be passed from Event Services. This method is used mainly for SMS related data.

To create a new ICM event map and registration

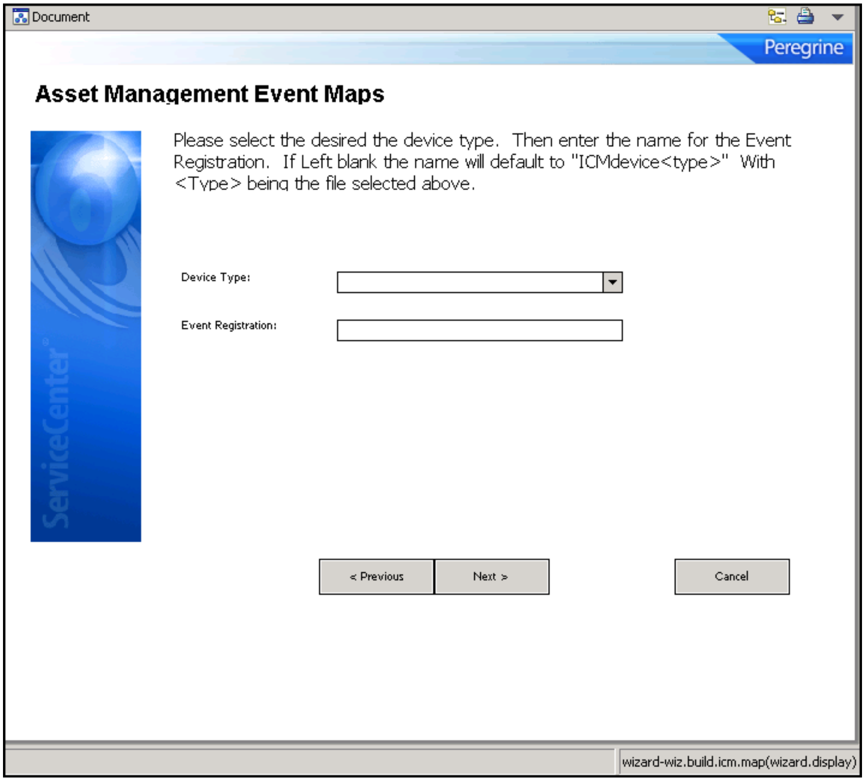
1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Build New ICM Event Map** to open the Asset Management Event Maps form.



Asset Management Event Maps

This wizard will walk through all the necessary steps to create a new Device Event Registration and Map.

2 Click **Next** to start the wizard.



Document

Peregrine

Asset Management Event Maps

Please select the desired the device type. Then enter the name for the Event Registration. If Left blank the name will default to "ICMdevice<type>" With <Type> being the file selected above.

Device Type:

Event Registration:

wizard-wiz.build.icm.map(wizard.display)

3 Click the arrow to select a device type from the drop-down list.

4 Type a name for the Event Registration, or leave the field blank to use the default naming convention: ICMdevice<*type*>.

5 Click **Next**.

You see the following confirmation message:

This wizard has now created the Device Event Registration and Map.

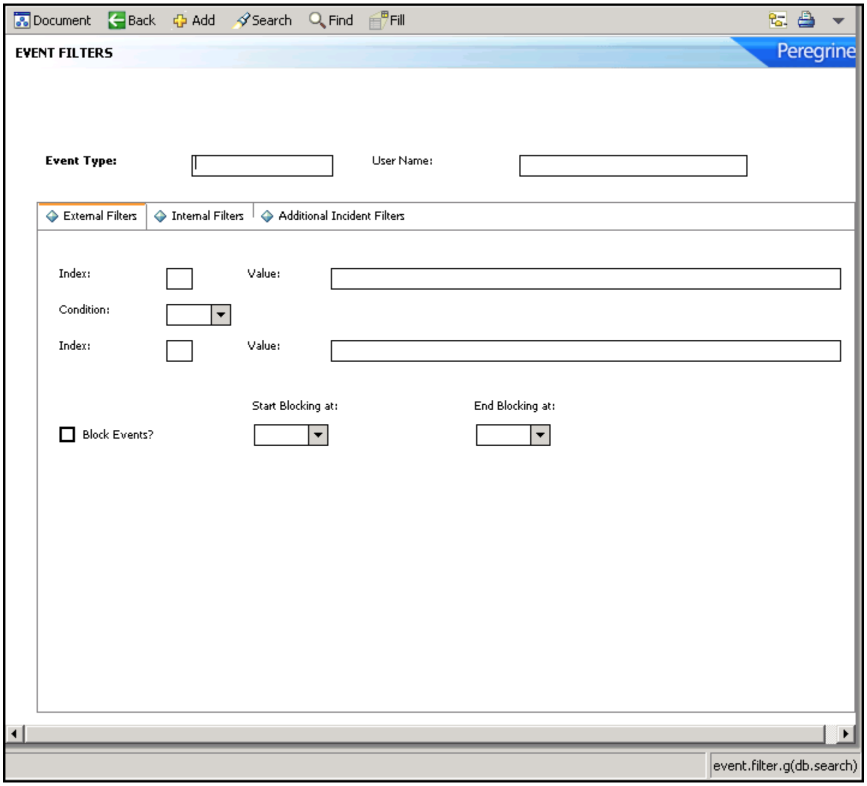
6 Click **Finish**.

Event filters

Event filtering information is stored in the eventfilter file. This file instructs SCAuto and SCAuto for NetView OS/390 when to block incoming events. If an event is not blocked, filters also can prevent opening incident tickets based on recurrence intervals and counts, and on incident intervals.

To review event filters

- From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Filters**.



Fields

The encoded field names recorded in the **eventfilter** file are included for reference only.

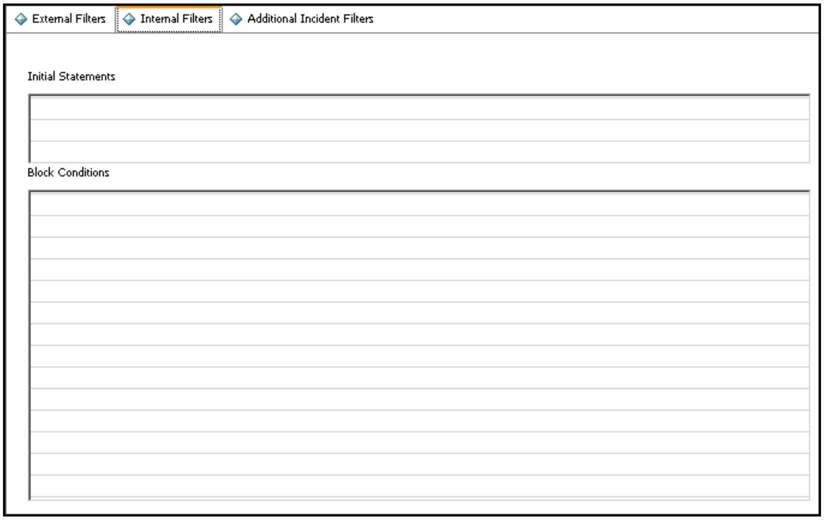
Header

Field	Description
Event Type(evtype)	Unique identifier for the event filter; must match the code in the **eventin** record.
User Name (evuser)	Name of the user or process, passed from external application; this field is required when blocking events from being written to the **eventin** file by the external scheduler.

External Filters tab

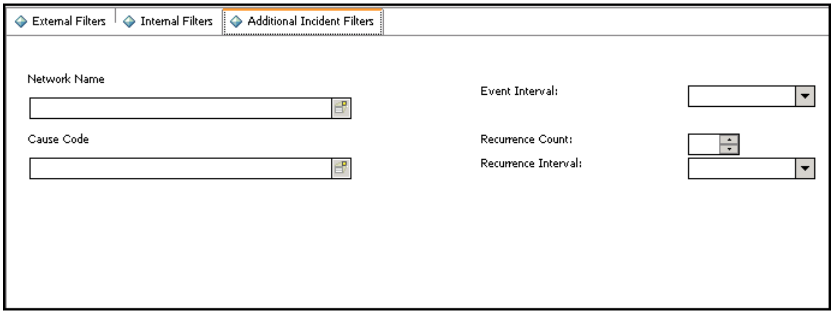
Field	Description
Index (evindex1)	Position in the **eventin** record's **evfields** field that identifies the first mask field.
Value (evvalue1)	Value that causes the event to be masked if it appears in the position indicated by **evindex1** in the **eventin** record's **evfields** field.
Condition (evcondition)	Value of *and* or *or* that concatenates the clauses built with the **evindex** and **evvalue** fields.
Index (evindex2)	Position in the **eventin** record's **evfields** field that identifies the second mask field.
Value (evvalue2)	Value that causes the event to be masked if it appears in the position indicated by **evindex2** in the **eventin** record's **evfields** field.
Block Events? (evblock)	Logical field that indicates whether events are blocked entirely; this field is required when blocking events from being written to the **eventin** table by the external scheduler. See *Blocking* on page 68 for more information.
Start Blocking at (evstime)	Beginning time for masking events.
End Blocking at (evetime)	Ending time for masking events.

Internal Filters tab



Fields	Description
Initial Statements (evinit)	Array of statements that execute at run time to initialize variables or initiate action based on the contents of the data passed in the eventin record and/or on global variables available at run time; the global variable $axces.fields represents an array of the fields passed in the **evfield** field of the eventin record.
Block Conditions (evblockcond)	List of conditions which, if any are true at run time, block the event and cause the registered application to exit normally; the status in the eventin record is then filtered.

Additional Incident Filters tab



Network Name:

Cause Code:

Event Interval:

Recurrence Count:

Recurrence Interval:

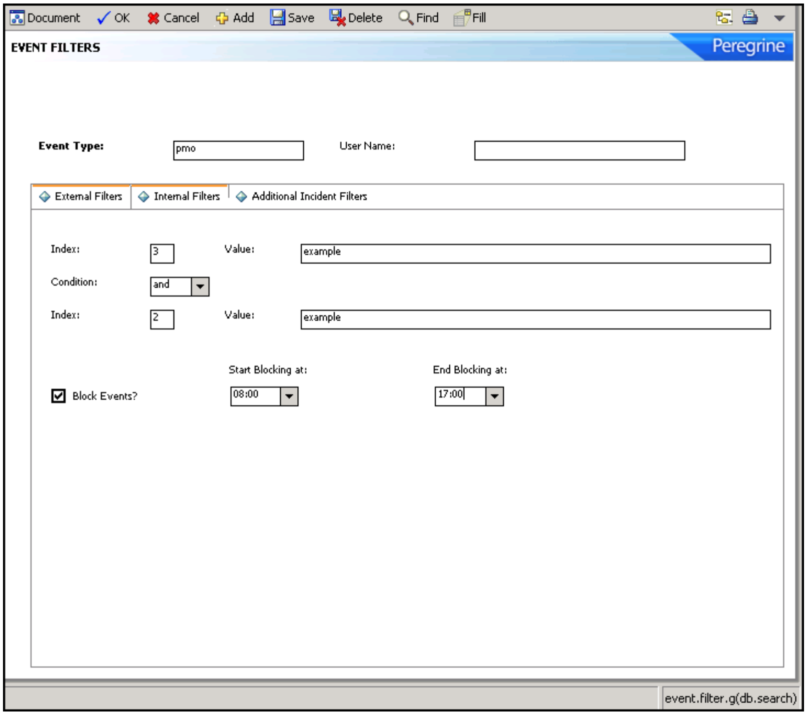
Fields	Description
Network Name (evnetnm)	Unique network identifier for the device; the external application masks all events; contains *SCAuto* for the master filter used for all internal blocking action.
Event Interval (interval)	Amount of time an event must be active before an incident is opened in ServiceCenter; effective only when **evblock** is **false**.
Cause Code (evcode)	Code, usually sent by the external agent, that identifies the fault.
Recurrence Count (recurrence.count)	If completed, the number of times an event must be received for a particular **evnetnm** or **evcode** before an incident is opened in ServiceCenter; effective only when **evblock** is **false**.
Recurrence Interval (recurrence.interval)	If completed, the amount of time (for example, 00:05:00) in which the recurrence.count is in effect; effective only when **evblock** is **false**.

Blocking

The external SCAuto and SCAuto for NetView OS/390 applications use the **External Filters** tab of the filter record to prevent the insertion of eventin records in the ServiceCenter database. The contents of the **User Name** field must either match that of the external process or be empty (NULL).

The **Block Events?** condition must be set to *true* to prevent records from being added to the eventin file. The **Start Blocking at** and **End Blocking at** values are optional, however they allow for a block to be placed over a specified time frame allowing a more customized administration.

In the following record, all incident open events are blocked from 08:00 to 17:00.



Event Type: pmo

User Name:

Index: 3 Value: example

Condition: and

Index: 2 Value: example

Start Blocking at: 08:00

End Blocking at: 17:00

☑ Block Events?

event.filter.g(db.search)

You can also prevent the insertion of events for specific network devices, domain names and error types by using the **Index, Value**, and **Condition** fields. Use these fields independently or in conjunction with the **Start Blocking at** and **End Blocking at** fields to populate other fields on the form.

- **Index** refers to the position of the data in the event message.
- **Value** refers to the actual data contained at that position.

For example, a pmo event contains the following message:

```
peregrine^peregrine^^6 58916865^Node Down^^^^SNMP
Trap(IPAS)^net.hware^^^^^^^^^^^
```

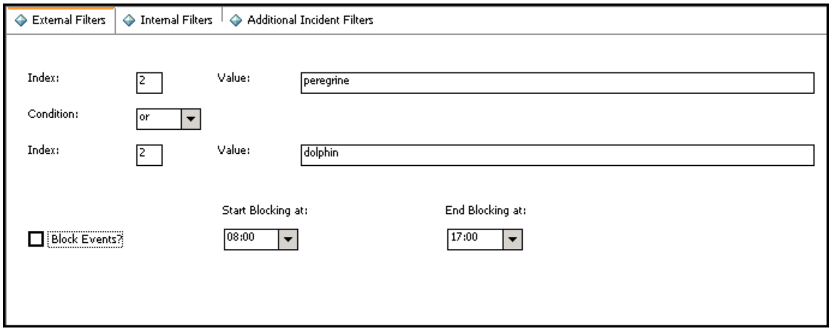
The ^ character separates fields in the message. The first field, which references the logical name of the device (refer to *Mapping* on page 44), contains peregrine. To block the insertion of all incident open events reported for the device peregrine, type pmo in the **Event Type** field, 2 in the first **Index** field and peregrine in the first **Value** field.

Note: Only Index values of 2 or 3 are supported for incident open actions.

To block incident open events from both peregrine and another server named dolphin, type information as previously described and type or in the **Condition** field, 2 in the second **Index** field and dolphin in the second **Value** field. If you specify a condition (and or or), then you must complete both **Index** and both **Value** fields.

Important: To prevent insertion of records in the eventin file, the **Block** field must be true.

In the following tab, all inventory add (icma) events are blocked between 08:00 and 17:00 if they come from either the peregrine or dolphin server. This action avoids unnecessary adds and updates if installation activity is scheduled to occur on the network during this time.



Index: 2 Value: peregrine

Condition: or

Index: 2 Value: dolphin

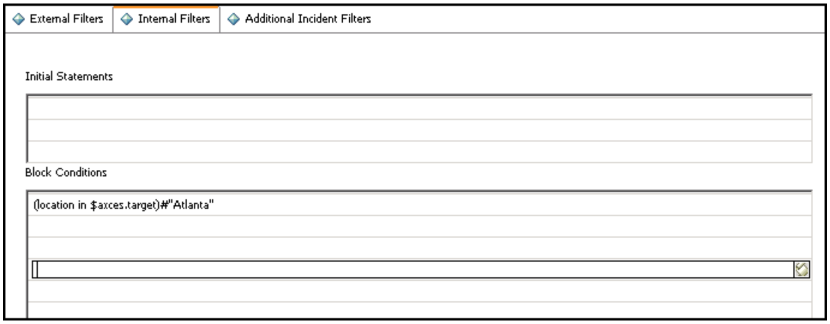
Start Blocking at: 08:00

End Blocking at: 17:00

□ Block Events?

The number of filters available for external blocking is unlimited; the external process (SCAuto or SCAuto for NetView OS/390) reads the **eventfilter** file to select records with the same **Event Code** and **User Name** (or User Name=NULL) and with **Block Events?**=true until it finds one that satisfies the criteria for the event being processed. If none is found, the event is inserted in the **eventin** file.

Once records are added to the eventin file, Event Services assumes the filtering task using **Internal Filters**. Event Services first selects the filter with the same **Event Code** as that of the event being processed and with a **Network Name** of SCAuto. This filter must contain **all** internal blocking conditions. If an eventin record satisfies one of the **Block Conditions**, it is updated to reflect a **Status** of blocked. The event action (for example, incident open or inventory add) does not take place.



Initial Statements

Block Conditions

(location in $axces.target)#"Atlanta"

With **incident open** event types (pmo), the **Additional incident Filters** take effect if no blocking condition exists. **This filtering mechanism is available only when opening new incidents.** Filters are selected using the following search criteria and in the order listed:

- The **Event Type** is the same as that of the event being processed and the **Network Name** is the same as the network name specified in the eventin record and the **Cause Code** is the same as the cause code specified in the eventin record.
- The **Event Type** is the same as that of the event being processed and the **Network Name** is the same as the network name specified in the eventin record.
- The **Event Type** is the same as that of the event being processed and the **Network Name** is *AXCES* and the **Cause Code** is the same as the cause code specified in the eventin record.
- The **Event Type** is the same as that of the event being processed and the **Network Name** is *AXCES.*

Using this event as an example:

```
peregrine^peregrine^^6 58916865^Node Down^^^^SNMP
Trap(IPAS)^net.hware^^^^^^^^^^^
```

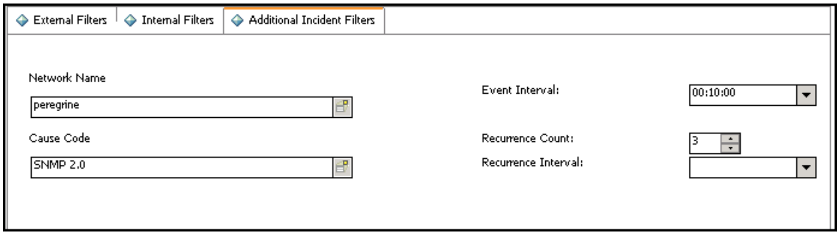
The queries are:

```
evtype=”pmo” and evnetnm=”peregrine” and evcode=”6 58916865”
evtype=”pmo” and evnetnm=”peregrine”
evtype=”pmo” and evnetnm=”AXCES” and evcode=”6 58916865”
evtype=”pmo” and evnetnm=”AXCES”
```

You can permanently block problem open by entering a Network Name or Cause Code. This has the same effect as a **Block Condition** except that the status in the **eventin** record is filtered rather than blocked.

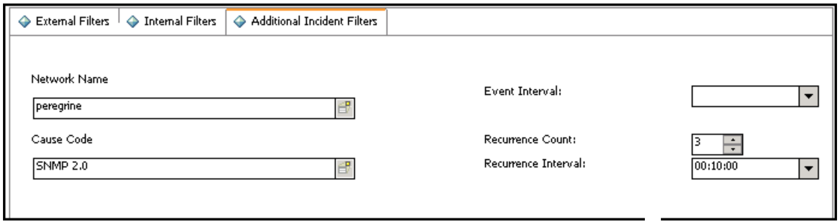
You can also use the **Event Interval**, **Recurrence Count**, and **Recurrence Interval** fields to limit problem open activity based upon frequency and duration.

The filter in the following record prevents any events from server **peregrine** with cause code of **SNMP 2.0** from opening a problem unless three events are received within a ten minute interval.



Network Name: peregrine
Cause Code: SNMP 2.0
Event Interval: 00:10:00
Recurrence Count: 3
Recurrence Interval:

The filter in the following record prevents any events from server **peregrine** from opening a problem unless 3 events are received and remain active for more than ten minutes.



Network Name: peregrine
Cause Code: SNMP 2.0
Event Interval:
Recurrence Count: 3
Recurrence Interval: 00:10:00

4 CHAPTER ServiceCenter/Network Discovery Integration

Peregrine Network Discovery provides network monitoring capabilities within ServiceCenter. Depending on your Network Discovery license, Network Discovery can automatically populate ServiceCenter's device records (Inventory/Configuration Management database) with data about the devices on your network.

Network Discovery can also send events to ServiceCenter through Event Services and automatically open problem tickets when a problem is detected on the network. You also can launch specific Network Discovery elements from ServiceCenter to quickly gather information about a device or problem.

Note: To launch Network Discovery from ServiceCenter, you must have a Network Discovery account and password. Refer to the *Network Discovery User Guide* for information.

The chapter describes:

- *How Network Discovery and ServiceCenter work together* on page 74
- *Accessing Network Discovery from ServiceCenter* on page 76
- *Accessing ServiceCenter from Network Discovery* on page 78

How Network Discovery and ServiceCenter work together

You can export data to ServiceCenter that Network Discovery collects. Users can then access parts of Network Discovery through ServiceCenter, and vice versa.

Network Discovery can export two types of data to ServiceCenter.

Data type	Description
Device data	Device data describes the network devices, such as device type (for example, a workstation or a router), description, Operating System, and so on.
Alarm data	Network Discovery constantly monitors the state of the network and its devices. Users can configure Network Discovery to trigger alarms based on specific thresholds.

For more information about using Network Discovery, refer to the *Network Discovery User Guide*.

Exporting Device data from Network Discovery to ServiceCenter

When you export data from Network Discovery, the data populates the ServiceCenter Inventory/Configuration Management (ICM) database.

The two ways to export data from Network Discovery are:

- With Connect-It
- Pre-configured exports from Network Discovery

Note: Although you can use both methods, it is recommended that you use the Connect-It scenario to initially populate the ICM database, and then use pre-configured exports to make updates.

Both methods use the following ServiceCenter Event Services events to update the ICM:

- ICMmainframe
- ICMtelecom
- ICMcomputer
- ICMnetcomp

- ICMofficeelec

These events are tailored to receive certain types of data that are specific to these devices.

With Connect-it

You can run a Connect-It scenario manually or on a scheduled basis.

Pre-configured Export from Network Discovery

A pre-configured export from Network Discovery export runs whenever Network Discovery:

- discovers a new device
- records a change on a device
- determines that a device has been removed from the network

ServiceCenter reacts to these network changes by adding or changing a device record in the ICM, or by flagging the device as "deleted."

Exporting Alarm data from Network Discovery to ServiceCenter

When Network Discovery sees an alarm on a device, it can automatically open (or close) a ticket in ServiceCenter.

Note: Alarms are triggered in Network Discovery when an attribute (for example, CPU Utilization) reaches a threshold. All alarms have default thresholds, but users can change these thresholds at any time.

This uses the following ServiceCenter Event Services events:

- NDpmo: to open or update a ticket
- NDpmc: to close a ticket

You must set up your Network Discovery Event Filters properly to send these events to ServiceCenter. For more details, see the *Network Discovery User Guide*.

In the Network Discovery event database, the alarm is logged when it is first detected, and then again when the condition abates. The alarm types include:

- critical
- major
- minor

- info
- OK

Note: Information events are also generated when there is no alarm, but a significant occurrence is detected for a device, such as when a device is added or deleted.

Accessing Network Discovery from ServiceCenter

You can access Network Discovery from any of these ServiceCenter components:

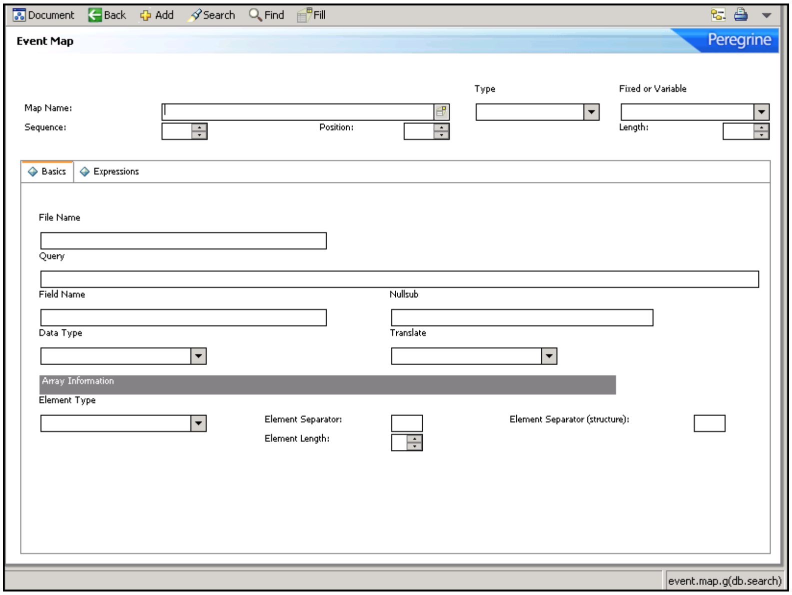
- Incident Management
- Inventory/Configuration Management
- Change Management

These components provide hyperlinks that point back to Network Discovery so you can get more details about specific devices or alarms.

Note: Network Discovery is web-based, so its features appear in a browser window.

To access the ND Problem Open event map

1 From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Maps** to open the Event Map(event.map.g) form.



2 Type ND problem open in the **Map Name** field.

3 Click **Search** or press Enter.

A QBE list of all the ND problem open map records opens.

Note: For inventory additions and changes, Connect-It sets the **ind.removed** field to false. For deletions, it is set to true.

Accessing ServiceCenter from Network Discovery

Once ServiceCenter opens a ticket, the ticket number is displayed in these Network Discovery components:

- Events Browser
- Alarms Viewer
- Device Manager
- Port Manager
- Attribute Manager

From the Events Browser, you can right-click on the ticket number and open a ServiceCenter window. This only works if you properly set up your connection to ServiceCenter. See the *Network Discovery User Guide* for details.

Opening and closing incident tickets

Network Discovery events provide information to Event Services for opening and closing incident tickets. The NDpmo and NDpmc events, respectively, trigger these actions on tickets.

Note: Use the NDpmo event to update an existing ticket or open a new ticket if one does not exist.

To view the appropriate map

1. From the ServiceCenter main menu, click **Utilities** > **Event Services** > **Administration** > **Maps**.
2. Type problem open, problem update, or problem close in the **Map Name** field.
3. Click **Search** or press Enter to display a QBE list of all the problem open, problem update, or problem close map records.

5 CHAPTER

Change Management Event Services

The Change Management module of ServiceCenter is fully supported by Event Services. This allows users outside of the ServiceCenter system to perform all standard functionality of Change Management from an external system, for example, SAP or PeopleSoft. The Event Services implementation is bi-directional, allowing external systems to synchronize with the ServiceCenter system.

This chapter provides the ServiceCenter system administrator with a basic understanding of the input and output events used to communicate data in and out of Change Management using Event Services. An administrator level of knowledge of Change Management and Event Services is required.

This chapter contains the following sections:

Input events

A correctly formatted **eventin** record must be created within ServiceCenter to use an external system to produce an action within ServiceCenter's Change Management module. You can format the **eventin** record with an SCAutomate product.

The **eventin** record fields specific to the Change Management implementation are:

Field	Description
Event Code (evtype)	Name of the corresponding Event Registration record to use for this event. This must always be **cm3rin** for changes and **cm3tin** for tasks.
User Name (evuser)	User name in this field is interpreted as the operator for this event. The Change Management environment used depends on which user is entered in this field.
External Information String (evfields)	Delimited data fields that correspond to a specific event mapping.

Input event registrations

The following two registrations are used for input events:

Event Code	Input/Output	Event Map	Application	Description
cm3rin	Input	cm3r	axces.cm3	Used for Changes
cm3tin	Input	cm3t	axces.cm3	Used for Tasks

One of these two event codes must appear in the **eventin** record, depending on whether the event is related to a change or a task.

Setting up the external information string

The External Information String, or EIS, is the **evfields** field of the **eventin** record. This field carries the specific data of the change or task into the ServiceCenter system. These fields are placed in a single string with a user-specified separation character (the default is the ^ character). The first four fields contain specific functions that determine which change/task is being processed and what action the system should take. These fields are passed in a specific order:

Sequence	Field Description
1	Change/Task number of the object to be acted upon. This field is blank when opening a change/task.
2	The foreign ID. This field is the identifier of the change/task used by the external system. This field is used if a different number is used outside of ServiceCenter.
3	Action Token indicates which logical action to take, either: open, update, close, reopen, approve, unapprove, disapprove.
4	The Change Group or Operator performing an approval action (only used for approve, unapprove, or disapprove.).

Determining the correct change/task

The first two EIS fields determine the unique identifier of the change or task both in ServiceCenter and in an external system (if applicable).

- The first field contains the unique number that corresponds to the number field in the cm3r or cm3t database dictionary. This field is blank if the action is open.
- The second field of the EIS corresponds to the foreign.id field of the change or task. This field specifies the unique identifier of the change or task in the external system that is sending the request. If the ServiceCenter number is not specified, the system attempts to find the correct record by comparing the foreign.id to this field.

Supported actions

Event Services uses the third field of the EIS to determine what type of action to perform on the specific change or task specified by one of the first two fields. The supported actions are:

Action	Description
approve	Approve a change/task
disapprove	Disapprove a change/task
unapprove	Unapprove a change/task
open	Create a new change/task
update	Update an existing change/task
close	Close current phase and go to the next phase if it exists
reopen	Reopen a change/task in the current phase

The third field of the EIS must contain one of these actions to correctly process the event.

Approval actions

When the action is an approval action (either an approve, disapprove, or unapprove), the Change Management Group or Operator Name that is performing the approval action must be specified in the fourth field of the EIS. The group or operator specified must match one of the approval groups specified in the change or task record for the approval action to complete properly.

Data fields

The remaining fields in the EIS contain field level data that Event Services uses to populate the change or task record being processed. If the action performed is not an open, these fields write over any existing data in the change or task. If a field in the EIS is blank, the existing data in the change or task is used.

The exact field that each piece of data corresponds to can be determined by examining the proper input event map for changes (`cm3r`) or tasks (`cm3t`).

Keeping ServiceCenter in-synch with an external system

When ServiceCenter is used with a separate external system, the changes and tasks must be synchronized between the two systems. Event Services supplies two methods of sending output to the external system for this task.

First, a simple acknowledgment can be sent to the external system. This acknowledgment contains enough data to map the ServiceCenter change/task number to the unique ID used in the external system, along with enough messages to determine if the input event was successful.

Alternatively, a complete output event may be sent to an external system in order to synchronize every piece of data between the two systems.

Acknowledgments

In order to synchronize the unique numbers of each system, the **cm3rinac** and **cm3tinac** event registrations are used:

Event Code	Input/Output	Event Map	Application	Description
cm3rinac	Output	cm3ack	axces.write	Used for Changes
cm3tinac	Output	cm3ack	axces.write	Used for Tasks

Both event types use the cm3ack event map definition. This mapping passes the following fields in the EIS of the eventout record:

Sequence	Field Description
1	Change/Task number of the object being acknowledged.
2	The foreign ID. This is the identifier of the change/task used by the external system. This field is used if a different number is used outside of ServiceCenter.
3	Action Token indicating which action was performed on this object (open, update, and so on).

Sequence	Field Description
4	The status of the eventin record created by the original event. This field may be used to determine if there were any errors encountered when processing the original event.
5	An array of up to 5 messages sent during the original event (ex: Change 15 updated, Location XXX is invalid). These messages can be used to determine if a Format Control or validation error occurred during the original event.

The acknowledgment events can be turned on or off in the cm3rin or cm3tin Event Registration records by modifying the value associated with the boolean1 parameter on the application tab. When this parameter value is set to true an acknowledgment event is sent out each time an input event is processed, while a setting of false keeps the acknowledgment event from being sent.

Sending complete output events

The standard output events for Change Management are triggered by the cm3messages file. When the change scheduler processes a cm3message, the value is checked in the **Event Services Reg** (axces.out) field in the corresponding cm3message record. If the value matches an output event (most likely cm3rout or cm3tout), that event is processed and an eventout record is written. This gives an administrator great flexibility when deciding what types of events (opens, alerts, etc.) cause the output event to be written.

The output maps used for these events are cm3r and cm3t. These maps correspond to their related input maps with the exception of the third and fourth fields. The third field contains the name of the event that caused the event to process (for example, cm3r open or cm3t update). The fourth field is used as a place-holder to keep the data fields of the input and the output event synchronized and always contains the words not used.

Change event examples

Input examples

Open a change

For example, open a change with the following parameters:

- The MAC category for pc001, with an external foreign ID of CM01, requested by falcon, assigned to bob.helpdesk.

The change contains a simple description while letting all other fields use default values.

The event register has the following specific fields:

Field	Value
evtype	cm3rin
evuser	falcon

The EIS is:

```
^CM01^open^^^MAC^^^falcon^^^^bob.helpdesk^^^^^^^^^^^^^^^^^^^^^^^^
^^^^^Move PC001 to Mike's
office.^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^pc001^^^...
```

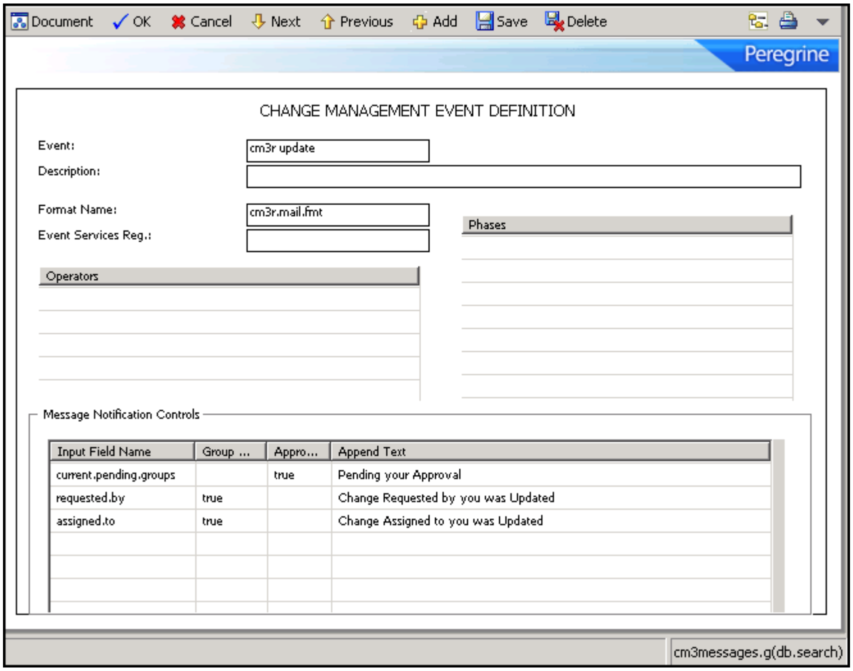
The field positions correspond to the cm3r input event map as follows:

Position	Field Name	Value
2	foreign.id	CM01
3	actiondummy	Open
6	category	MAC
9	requested.by	falcon
13	assigned.to	bob.helpdesk
42	description	Move PC001 to Mike's office.
76	logical.name	pc001

Output example

Using cm3messages to output changes when updated

Entering cm3rout in the cm3r update record triggers an output event whenever a change is updated.



CHANGE MANAGEMENT EVENT DEFINITION

Event: cm3r update

Description:

Format Name: cm3r.mail.fmt

Event Services Reg.:

Message Notification Controls

Input Field Name	Group ...	Appro...	Append Text
current.pending.groups		true	Pending your Approval
requested.by	true		Change Requested by you was Updated
assigned.to	true		Change Assigned to you was Updated

6 Event Agent Operations

CHAPTER

Automatic monitors within ServiceCenter, known as agents, can be set to collect data and create events appropriately within the system. You can use the Event Scheduler to set up these agents, or you can activate them automatically or manually (by user input).

Information about event agents includes:

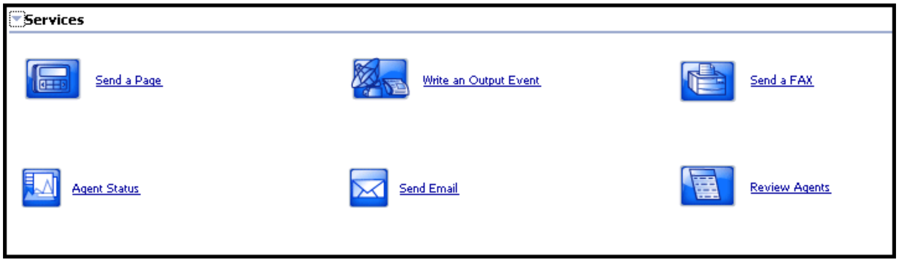
Event scheduling

The **schedule** file contains a record for each SCAuto agent. It contains instructions indicating how often the **agent** reads a queue, and which application to execute if the read returns records.

Reviewing scheduled events

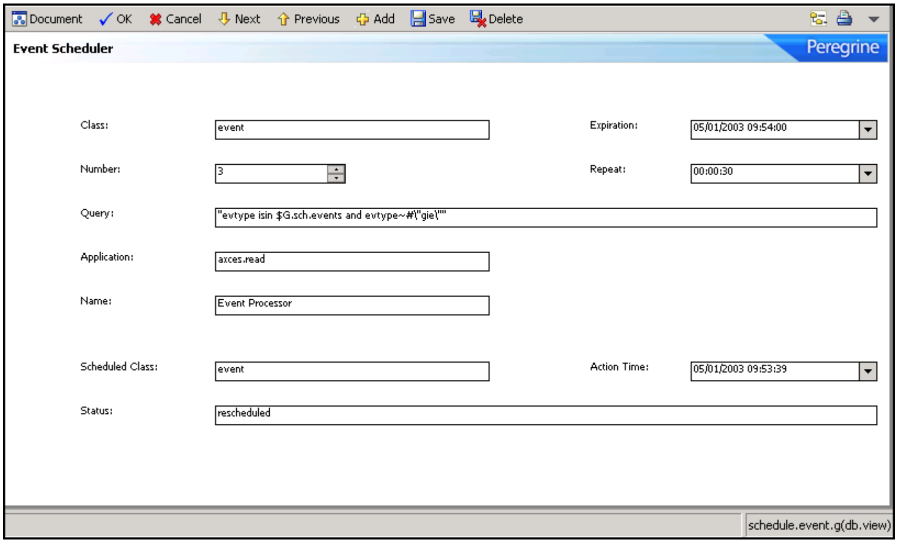
To review SCAuto event schedules

1 From the ServiceCenter main menu, click **Utilities** > **Event Services**.



2 From the Services menu, click **Review Agents** to open a QBE list of event agents.

3 Select an agent from the list to open the Event Scheduler.



Class:	event	Expiration:	05/01/2003 09:54:00
Number:	3	Repeat:	00:00:30
Query:	"evtype isin $G.sch.events and evtype~#\"gie\""		
Application:	axces.read		
Name:	Event Processor		
Scheduled Class:	event	Action Time:	05/01/2003 09:53:39
Status:	rescheduled		

Schedule fields

The encoded field names recorded in the schedule file are included for reference only.

Field	Description
Class (class)	Schedule class; must match the name of the agent as defined in the **info** startup record.
Expiration (expiration)	Data and time when the agent next is activated.
Number (number)	Unique number to identify the schedule record.
Repeat (repeat)	Interval defining the **sleep** time for the application.
Query (query)	Optional query that you can combine with the class to allow multiple agents.
Application (application)	Name of the ServiceCenter application that the agent calls.
Name (name)	Name associated with the agent. **Note:** For OS/390/SCAuto, SCAuto for NetView OS/390 and NAPA agents, the name must either be blank or match the name of the associated record in the config file. For example, if the config record that describes the input vsam file is named VSAMIN, the name in the agent record must be VSAMIN. If the name is blank, ServiceCenter uses the name of the class to select the config record. If a config record cannot be found with the name (or, if the name is blank), the class, or schedule class defined in the agent, the associated application fails with an error.
Scheduled Class (sched.class)	Class that was used when the application last executed.
Action Time (action.time)	Last time the application executed.
Status (status)	Current status of the event scheduler: ■ application running ■ rescheduled ■ application failed due to error

When the event agent starts, the event schedule record must have a Class of **event** (or whatever you specify the event scheduler's name to be) and must have an expiration earlier than the current time. Set the expiration to the current date and time before starting the scheduler.

Since the event scheduler is a serial process, you may want to have more than one scheduler read events in the event queue. This is particularly true when inventory activity is high, preventing incident management activity.

Use the **Query** field to further define what type of event to select from the **eventin** file.

The user-specified query entered in the schedule record is appended automatically to the default event scheduler query, evtime<=tod(), to form a more specific query. If the **Query** field is left blank, only the default query is applied.

Note: The system always places the time portion of the query in front of the user-specified query.

If you define a query for use against the **eventin** file, be sure it is fully-keyed for maximum performance.

Important: The agent processor attempts to restart any applications that ended while running (that have a status of **application running**). If you change for one of your agents, make sure there are no other agents with the same schedule class and a status of **application running**.

OS/390 (MVS)/SCAuto agents

SCAutomate allows you to read and write any number of VSAM files. For each VSAM file read or written, there must be a separate scheduler with a unique value in the **Class** and **Name** fields and a separate **config** record that defines the data set name (for example, **netview** for the SCAuto for NetView OS/390 agent).

All events read from a VSAM file are written to the **eventin** file. They must be in standard SCAutomate **eventin** format.

All events written to a VSAM file originate from the **eventout** file. They must use standard **eventout** format. Once the **vsam.write** scheduler processes an **eventout** record, the **evexpire** field is set to NULL and the **Status** is updated with either error or written.

Maintaining agent status

You can start and stop agents within ServiceCenter by using:

- System startup
- Status window
- Event agent check

System startup

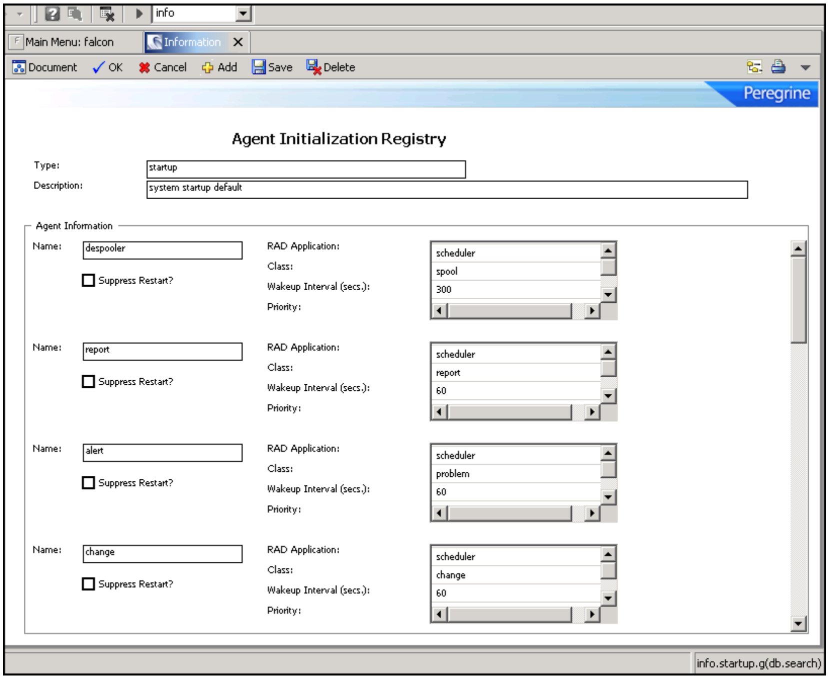
To view the startup info record

1 Type info in the command line and press Enter, or from the ServiceCenter main menu, click **Utilities** > **Maintenance**.

2 From the System menu, click **Startup Information** to open a blank Agent Initialization Registry record.

3 Enter startup in the **Type** field.

4 Click **Search** or press Enter.



Agent Initialization Registry

Type: startup

Description: system startup default

Agent Information

Name: despooler
□ Suppress Restart?
RAD Application: scheduler
Class: spool
Wakeup Interval (secs.): 300
Priority:

Name: report
□ Suppress Restart?
RAD Application: scheduler
Class: report
Wakeup Interval (secs.): 60
Priority:

Name: alert
□ Suppress Restart?
RAD Application: scheduler
Class: problem
Wakeup Interval (secs.): 60
Priority:

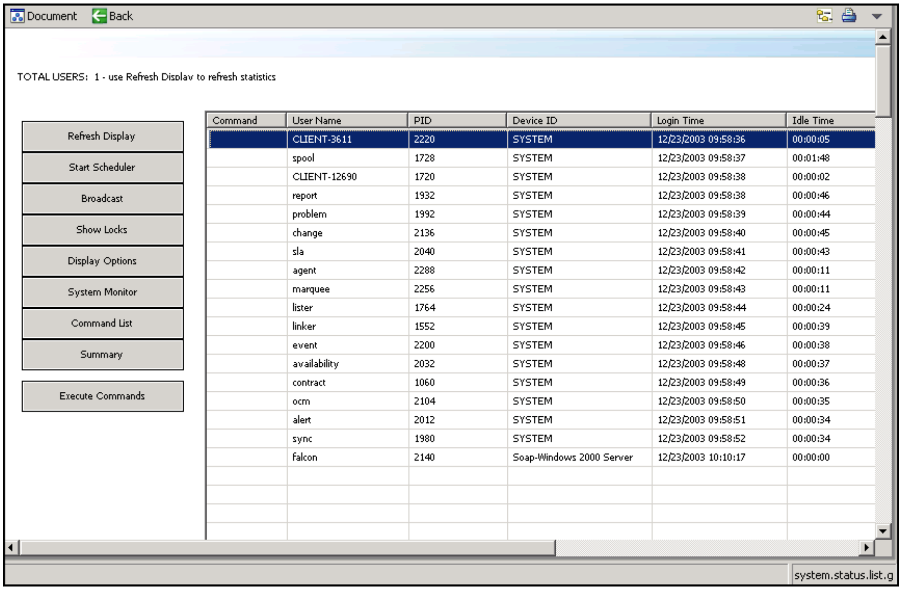
Name: change
□ Suppress Restart?
RAD Application: scheduler
Class: change
Wakeup Interval (secs.): 60
Priority:

info.startup.g(db.search)

At system startup, all agents defined in this record are initialized.

System status window

Click **System Status** (from the Top section of the ServiceCenter main menu), or type **status** in the command line to display the system status window. From this window you can start or stop (kill) individual agents by name if you are either using an express client or are directly logged into ServiceCenter from its server.



TOTAL USERS: 1 - use Refresh Displav to refresh statistics

Command	User Name	PID	Device ID	Login Time	Idle Time
	CLIENT-3611	2220	SYSTEM	12/23/2003 09:58:36	00:00:05
	spool	1728	SYSTEM	12/23/2003 09:58:37	00:01:48
	CLIENT-12690	1720	SYSTEM	12/23/2003 09:58:38	00:00:02
	report	1932	SYSTEM	12/23/2003 09:58:38	00:00:46
	problem	1992	SYSTEM	12/23/2003 09:58:39	00:00:44
	change	2136	SYSTEM	12/23/2003 09:58:40	00:00:45
	sla	2040	SYSTEM	12/23/2003 09:58:41	00:00:43
	agent	2288	SYSTEM	12/23/2003 09:58:42	00:00:11
	marquee	2256	SYSTEM	12/23/2003 09:58:43	00:00:11
	lister	1764	SYSTEM	12/23/2003 09:58:44	00:00:24
	linker	1552	SYSTEM	12/23/2003 09:58:45	00:00:39
	event	2200	SYSTEM	12/23/2003 09:58:46	00:00:38
	availability	2032	SYSTEM	12/23/2003 09:58:48	00:00:37
	contract	1060	SYSTEM	12/23/2003 09:58:49	00:00:36
	ocm	2104	SYSTEM	12/23/2003 09:58:50	00:00:35
	alert	2012	SYSTEM	12/23/2003 09:58:51	00:00:34
	sync	1980	SYSTEM	12/23/2003 09:58:52	00:00:34
	falcon	2140	Soap-Windows 2000 Server	12/23/2003 10:10:17	00:00:00

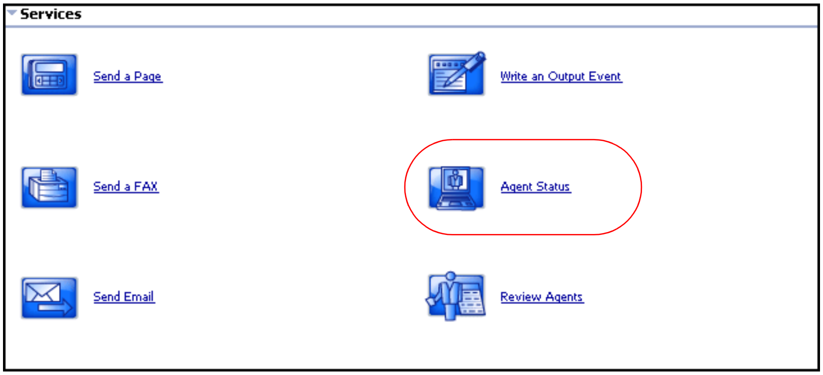
Event agent check

From Event Services you can start and stop any SCAuto or event agent without respect to your client status as long as the ServiceCenter **problem** agent is active. Using this feature, agents are scheduled to start, and the **problem** agent is their activation agent. The specific agents controlled from this option include:

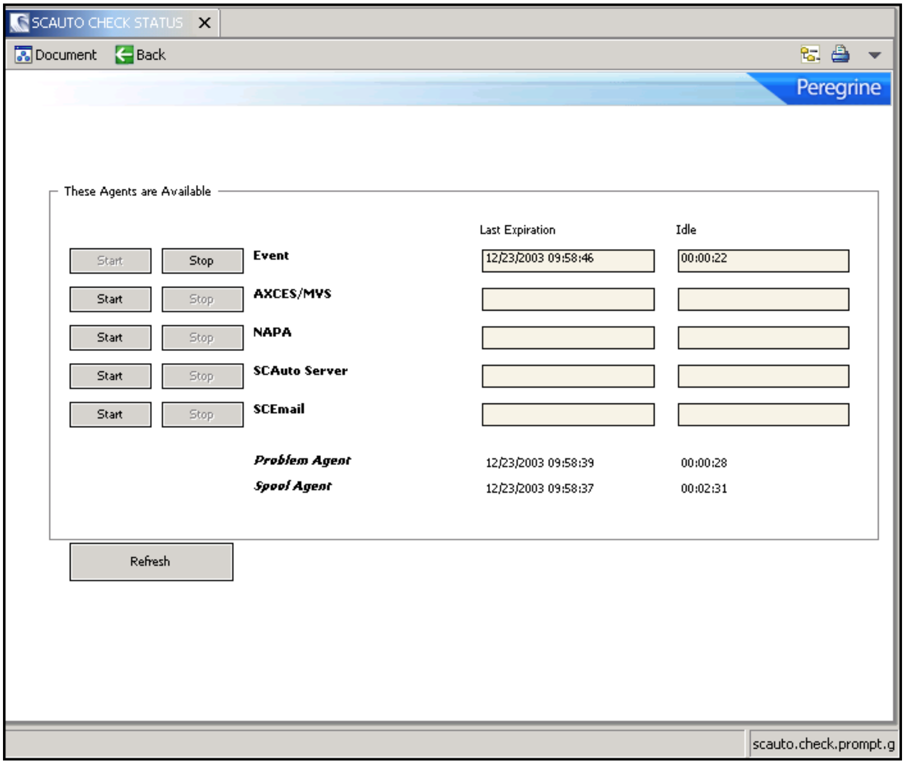
event
vsamout (SCAuto/MVS)
scemail
vsamin (SCAuto/OS/390)
scauto
netview (SCAuto for NetView OS/390 or NAPA)

To maintain SCAuto agents

1 From the ServiceCenter main menu, click **Utilities** > **Event Services**.



2 From the Services menu, click **Agent Status** to open the form that lists all the available agents.



These Agents are Available

			Last Expiration	Idle
Start	Stop	Event	12/23/2003 09:58:46	00:00:22
Start	Stop	AXCES/MVS		
Start	Stop	NAPA		
Start	Stop	SCAuto Server		
Start	Stop	SCEmail		
		Problem Agent	12/23/2003 09:58:39	00:00:28
		Spool Agent	12/23/2003 09:58:37	00:02:31

Each active agent shows a **Last Expiration** time and an **Idle** time. The **Last Expiration** time is the initialization time for the agent; the **Idle** time is the amount of time elapsed since the agent last woke up to check for work.

If an agent is inactive, there is no **Last Expiration** or **Idle** time, and the **Start** button is available.

3 Click **Start** to initialize the agent.

Note: The sleep interval is defined in the agent's info startup record.

4 Click **Stop** to disable an active agent.

Important: Since the **problem** agent schedules activation and deactivation of the agent, you must wait for it to wake up before your selected agent is started or stopped.

The OS/390 (MVS)/SCAuto agent automatically establishes both the **vsamin** and the **vsamout** agents.

You can define additional MVS/SCAuto agents to read from or write to other VSAM files, or event agents to selectively process input events, but these agents must be started and stopped using either the System Startup or the Status Window methods.

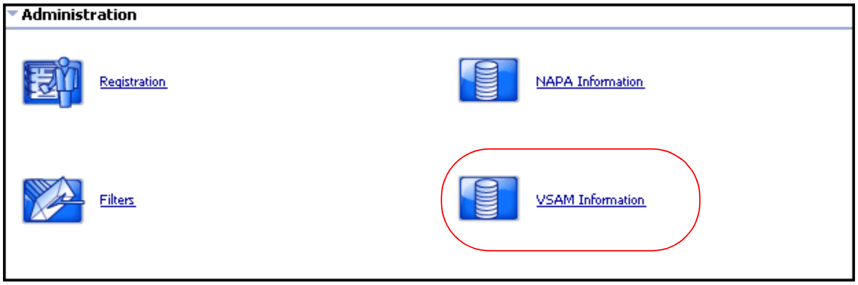
The VSAM information record

The vsaminfo file contains records that reflect the status of external VSAM files read by Event Services tasks. The scheduler uses this information to automatically open, update, and close problems and to maintain inventory records in the database.

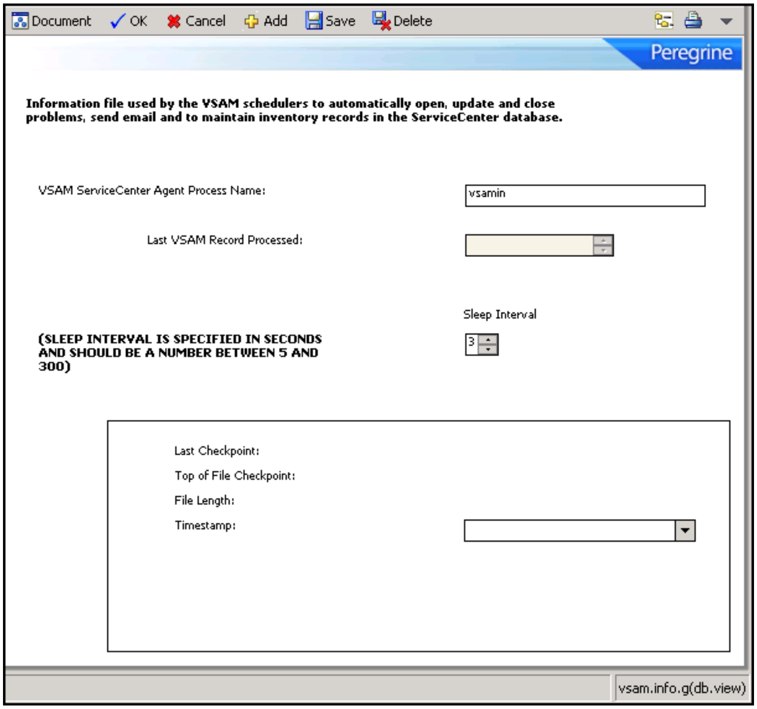
Reviewing the vsaminfo record

To review the VSAM information record

1 From the ServiceCenter main menu, click **Utilities** > **Event Services**.



2 From the Administration menu, click **VSAM Information** to open the VSAM Information form.



Information file used by the VSAM schedulers to automatically open, update and close problems, send email and to maintain inventory records in the ServiceCenter database.

VSAM ServiceCenter Agent Process Name: vsamin

Last VSAM Record Processed:

Sleep Interval 3

(SLEEP INTERVAL IS SPECIFIED IN SECONDS AND SHOULD BE A NUMBER BETWEEN 5 AND 300)

Last Checkpoint:

Top of File Checkpoint:

File Length:

Timestamp:

vsam.info.g(db.view)

VSAM information fields

The encoded field names that the **vsaminfo** file uses are included for reference only.

Field	Description
VSAM ServiceCenter Agent Process Name (name)	Name of the scheduler; this name must match the class in the schedule record.
Last VSAM Record Processed (item)	Index of the last VSAM record processed; do not modify this value.
Sleep Interval (sleep)	Number of seconds, between 5 and 300, to sleep if there is no NetView activity.
Last Checkpoint (sequence)	Checkpoint ID for the last record processed; do not modify this value.
Top of File Checkpoint (top)	Checkpoint ID for the first record in the VSAM file; do not modify this value.
File Length (length)	Length of the VSAM file (number of records; do not modify this value.
Timestamp (timestamp)	Timestamp in the last record processed; do not modify this value.

Note: The **vsam.read** application maintains the VSAM Information record.

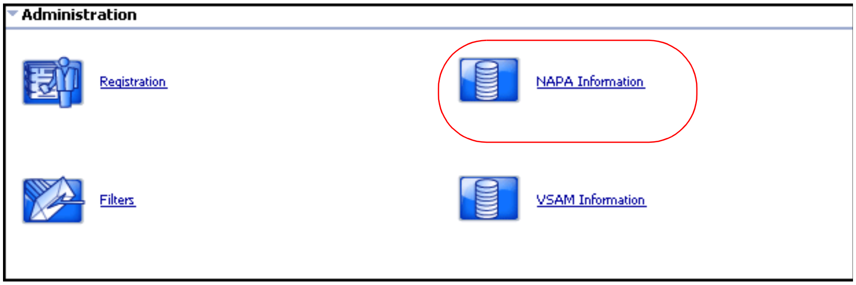
The NAPA information record

The **vsaminfo** file contains records that reflect the status of external VSAM files read by Event Services tasks. The scheduler uses this information to automatically open, update, and close problems, and to maintain inventory records in the database. IBM's NetView products write the NAPA information. OS/390 (MVS)/SCAuto then provides that information to ServiceCenter.

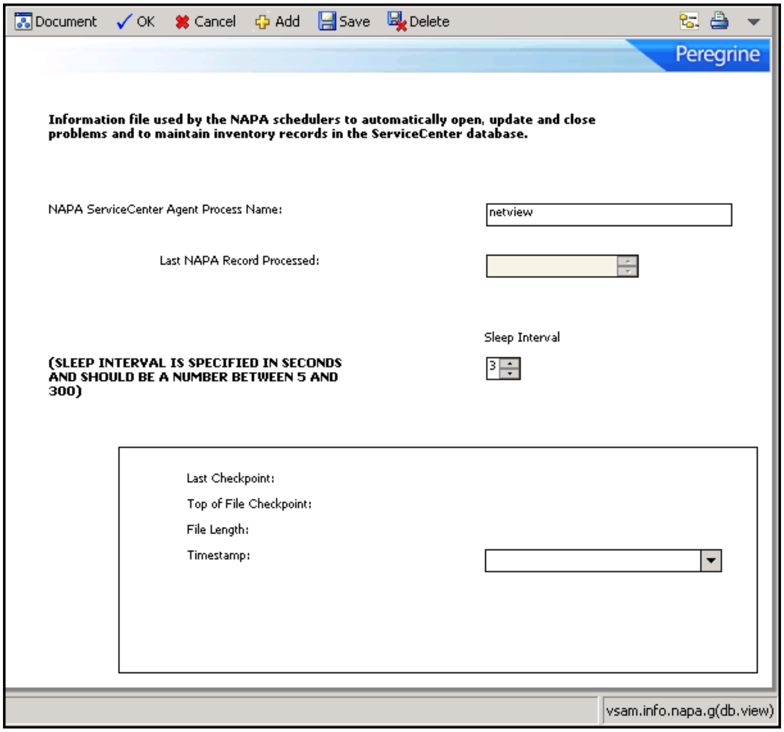
Reviewing the napainfo record

To review the NAPA information record

1 From the ServiceCenter main menu, click **Utilities** > **Event Services**.



2 From the Administration menu, click **NAPA Information** to open the NAPA Information form.



Information file used by the NAPA schedulers to automatically open, update and close problems and to maintain inventory records in the ServiceCenter database.

NAPA ServiceCenter Agent Process Name: netview

Last NAPA Record Processed:

Sleep Interval 3

(SLEEP INTERVAL IS SPECIFIED IN SECONDS AND SHOULD BE A NUMBER BETWEEN 5 AND 300)

Last Checkpoint:

Top of File Checkpoint:

File Length:

Timestamp:

vsam.info.napa.g(db.view)

NAPA information fields

The encoded field names that the **vsaminfo** file uses are included for reference only.

Field	Description
NAPA ServiceCenter Agent Process Name (name)	Name of the scheduler; this name must match the class in the schedule record.
Last NAPA Record Processed (item)	Index of the last VSAM record processed. **Note:** Do not modify this value.
Sleep Interval (sleep)	Number of seconds, between 5 and 300, to sleep if there is no NetView activity.
Last Checkpoint (sequence)	Checkpoint ID for the last record processed. **Note:** Do not modify this value.
Top of File Checkpoint (top)	Checkpoint ID for the first record in the VSAM file. **Note:** Do not modify this value.
File Length (length)	Length of the VSAM file (number of records). **Note:** Do not modify this value.
Timestamp (timestamp)	Timestamp in the last record processed. **Note:** Do not modify this value.

The **vsam.read** application maintains the NAPA Information record.

CHAPTER 7

SCemail

SCemail provides a monitor to handle ServiceCenter e-mail events. This monitor connects ServiceCenter into standard e-mail facilities and allows ServiceCenter operators and applications to send mail using e-mail. Any mail system that supports SMTP (Simple Mail Transfer Protocol) or has an SMTP gateway or bridge can receive e-mail from SCemail. SMTP is not required to use SCemail. Mail support in OS/390 (formerly MVS mainframe) environments is extended to support any e-mail that the TSO **Transmit** command can process. Mail support on Windows systems includes support for MAPI-compliant mail servers.

This chapter contains the following sections:

E-mail events

A standard e-mail event that ServiceCenter creates is the opening of a problem with a valid **Contacts** field. This event can notify individuals of a problem in their area of expertise. You also can create e-mail events using the **User Utilities Send Mail** function.

In addition to the standard creation of e-mail events in ServiceCenter, any RAD application can create an event. An example of this is implementing e-mail notification for problems that reach a certain status.

SCemail vs. SCAutoMail

SCemail is not the same product as SCAutomate Mail. SCemail only sends mail from ServiceCenter; it does not receive mail from external mail applications. SCemail runs as a stand-alone application; SCAutomate Mail is an SCAutomate client adapter.

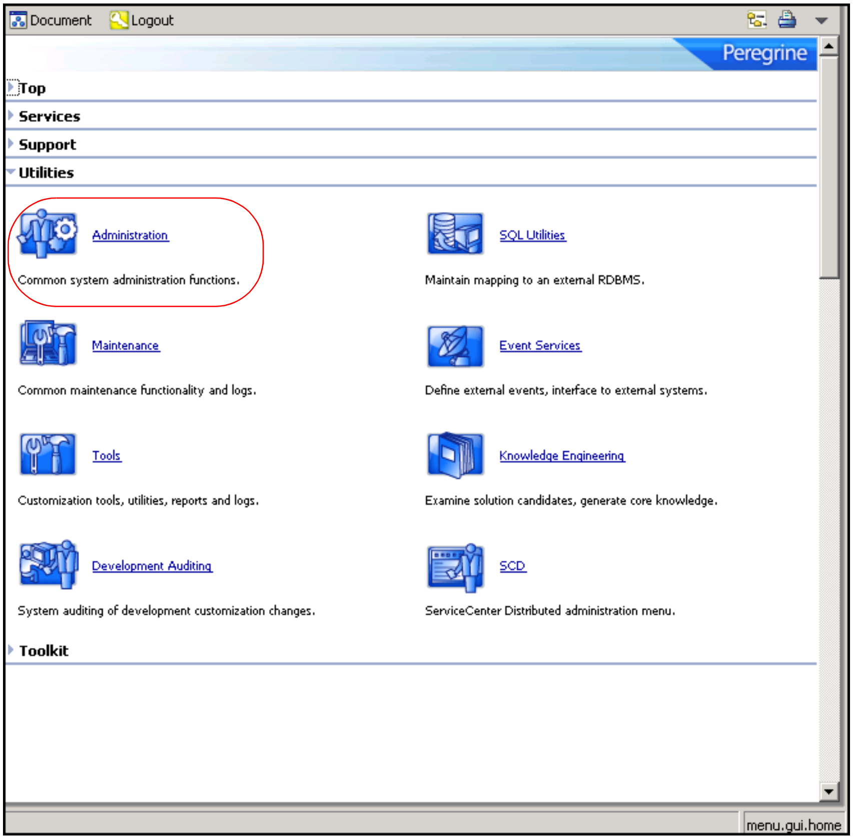
Sending ServiceCenter mail to e-mail

Sending ServiceCenter mail to e-mail users is a quick process. Your System Admin must login and change the user's operator file to point to the external e-mail address for that user.

To modify a user's operator file for e-mail

1 Login to ServiceCenter using a client with **SysAdmin** authority.

2 From the ServiceCenter main menu, click **Utilities** in the system administrator's home menu.



Administration

Common system administration functions.

SQL Utilities

Maintain mapping to an external RDBMS.

Maintenance

Common maintenance functionality and logs.

Event Services

Define external events, interface to external systems.

Tools

Customization tools, utilities, reports and logs.

Knowledge Engineering

Examine solution candidates, generate core knowledge.

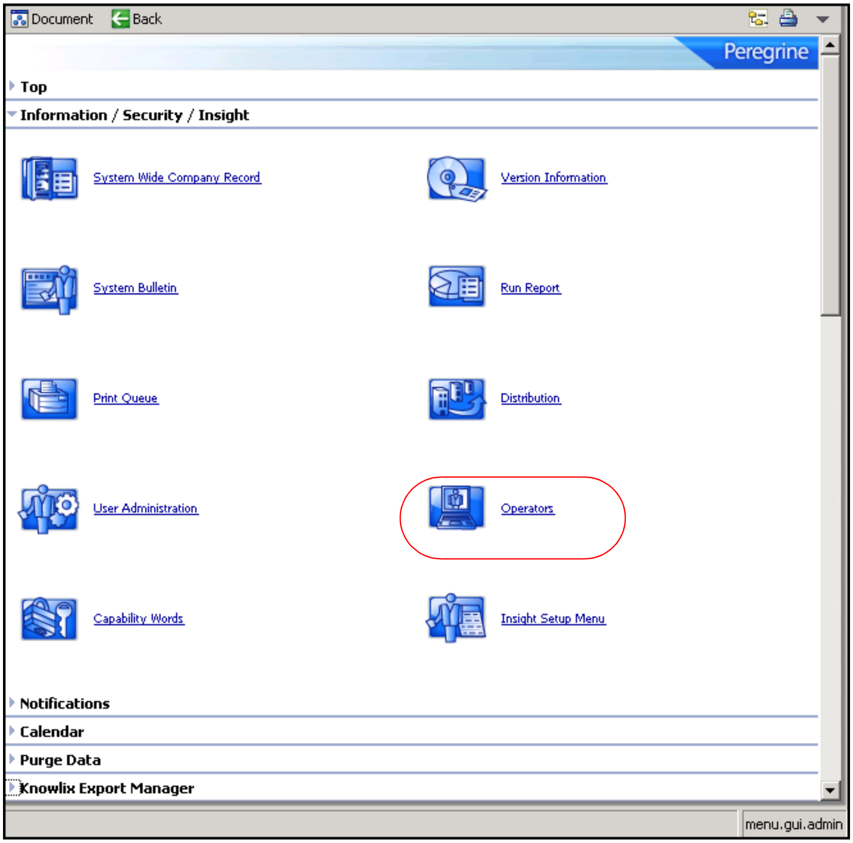
Development Auditing

System auditing of development customization changes.

SCD

ServiceCenter Distributed administration menu.

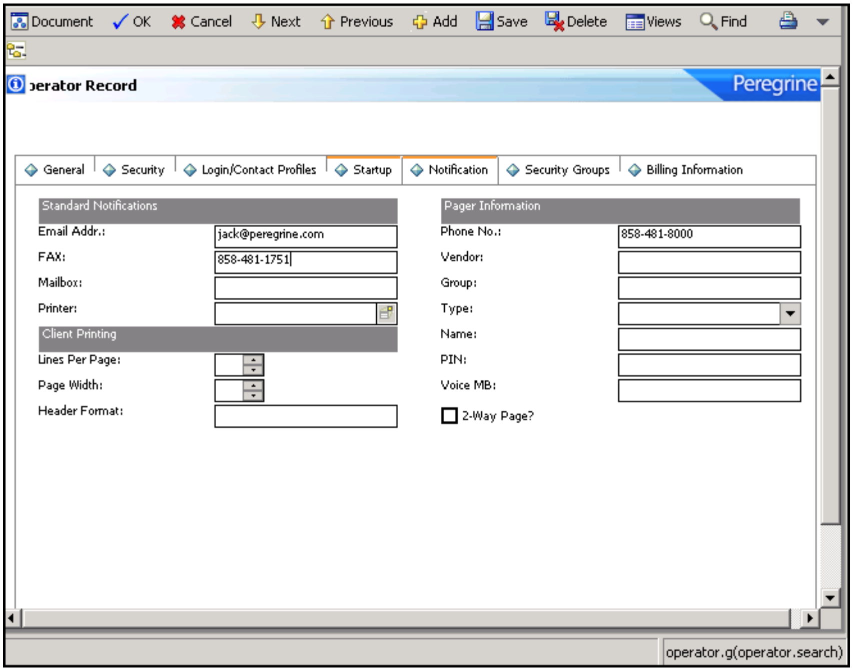
3 Click **Administration** to open the Administration menu.



4 Click **Operators** to open a blank operator record.

5 In the **Login Name** field, type the login name of the operator record you want to modify.

6 Press **Enter**.



ɔerator Record

Standard Notifications

Email Addr.: jack@peregrine.com

FAX: 858-481-1751

Mailbox:

Printer:

Client Printing

Lines Per Page:

Page Width:

Header Format:

Pager Information

Phone No.: 858-481-8000

Vendor:

Group:

Type:

Name:

PIN:

Voice MB:

□ 2-Way Page?

operator.g(operator.search)

7 From the Notification tab, type the e-mail address for that respective user in the **Email Addr.** field.

8 Click **Save** to record the operator record.

Changes to existing ServiceCenter Mail Utility

The existing ServiceCenter Mail Utility checks the operator file for valid operator names before allowing mail to be sent. The Event Services version of this application expands the checking for valid users to those defined in the ServiceCenter contacts file.

The purpose of the checking is to obtain the e-mail address from the operator or contacts file's **email** field. If the addressee's name does not select a record from either file, ServiceCenter assumes that there is no such addressee and does not send mail. You can override this default by creating a login.event Format Control record and, in the Calculations section, setting the **add** condition to true and the **calculation** expression to the following:

```
$email.noaddr.ok=true
```

This causes ServiceCenter to assume that whatever name is passed to the e-mail event as the addressee is the complete e-mail address and attempts to send mail using that address.

SCemail

You can use SCemail works in Windows, UNIX and OS/390 environments.

Windows

The SCemail program sends e-mail within ServiceCenter. Under Windows, SCemail uses the Messaging Application Program Interface (MAPI). Microsoft Exchange, Lotus Notes, Lotus cc:Mail and other mail vendors support this interface.

Mail profiles

MAPI uses the concept of a profile. A MAPI profile contains the information necessary to log on to a group of mail services. A profile is not the same as a user login, and a single user can have multiple entries within one MAPI profile.

For example, your SCemail profile is **Joe**, and that profile contains the MS Exchange, cc:Mail, Lotus Notes, and so on login and mailbox account information that allows you to interface with those systems, for example, **M:\mail, JJohnson**.

When using SCemail, you need to sign on using the SCemail profile, not the external mail account or login names. It is for this reason that each user must have a unique SCemail profile in addition to having a standard mail account.

Profiles began with MAPI in Windows 95 and Windows NT 4.0. The default Windows NT 3.51 system does not use profiles unless additional software upgraded the MAPI system (such as Microsoft Exchange Client or Lotus cc:Mail).

Important: SCemail does not work under Windows unless you upgrade MAPI.

For best performance and accuracy, assign SCemail its own MAPI profile and its own mailbox or mail account. This mail account acts as a gateway from ServiceCenter.

Adding a new profile

Refer to the documentation with your particular mail products if you need additional information.

To add a new profile

1. From the Windows main menu, click **Start** > **Settings** > **Control Panel**.
2. Click the **Mail** or **Mail and Fax** icon.
3. Click **Show Profiles** to display the profiles for your computer.
4. Click **Add** to open the Setup Wizard.
5. Select the service to use.

   Note: SCemail only uses one service, so do not select more than one.

6. Following the wizard directions, name and configure the profile.

   This is where you assign the mailbox or mail user for SCemail to use.

   You can test this profile by logging on with a MAPI-compliant mail client (Microsoft Outlook, cc:Mail).

Starting SCemail

To send mail, SCemail must log on to Windows mail.

Note: You must have ServiceCenter installed and operational.

To start SCemail and log on to Windows-based e-mail

1. From a DOS prompt, change to your ServiceCenter RUN directory. For example, `cd c:\scserver\run`
2. From the ServiceCenter RUN directory, type `scemail` and then the mail profile name.

   Note: Use quotation marks if the profile name contains spaces.

   ```
   scemail "<profile name>"
   ```

   This starts the SCemail background processor.
3. Check the **sc.log** file (usually located in the top level of your ServiceCenter folder) to verify that the SCemail background processor started successfully.

   If the processor started successfully, the **sc.log** file displays the following message: `SCemail: Initializing.`

SCemail on Windows helpful hint

There are different syntax variations when entering your address in the operator record. Type the name that is in the address book of an external mail client. You can also use SMTP style addresses of the form `username@host.com`.

Once you make the previous corresponding changes to the operator record, any user that has access to send mail can send ServiceCenter mail. If mail sent from ServiceCenter is undeliverable, it returns to the user with an error message.

Optional parameters

You can add the following optional parameters when starting the SCemail background processor.

Parameter	Description
-keepmail	Do not delete mail events once sent successfully.
-sleep <n>	Number of seconds to sleep between checking for events and mail. Default is 10 seconds.
-gui	Allow a pop-up dialog if additional login information is required (no profile was passed on the command line, or a password is required).
-debug	Print more diagnostics to sc.log. This turns on -keepmail as well.

Note: SCemail also processes the sc.ini file for additional parameters and can pass the parameters on the command line (for example, -log:file places the SCemail diagnostics in a different file).

Additional Windows compatibility and setup notes

If you use Lotus Notes, the following restrictions apply:

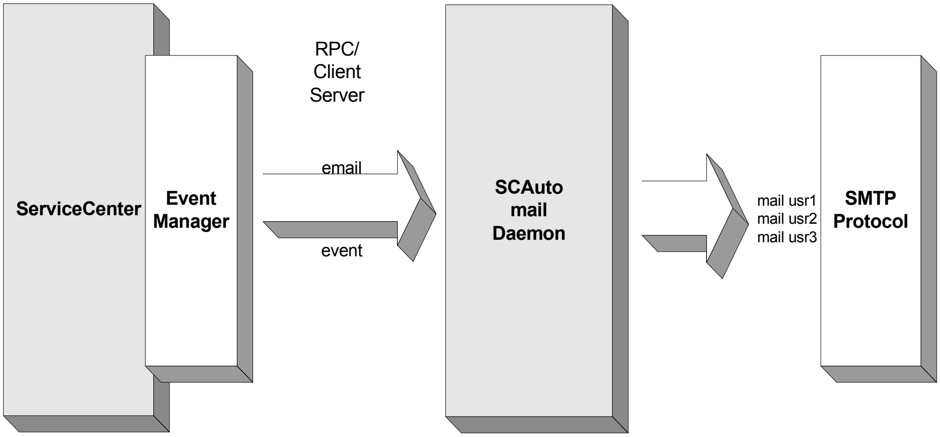
- You must have Lotus Notes version 4.11 or later to work with MAPI.
- Be sure to install Windows Messaging (a part of Windows), Microsoft Outlook, cc:Mail, or other MAPI-compliant mail client before installing Lotus Notes. This applies even if you do not use those mail clients because Lotus Notes does not install the necessary MAPI libraries.
- After setting up a profile for use with Lotus Notes, edit the properties of the profile and select the Delivery tab. Change the selection under **Deliver new mail to the following location** to read **Lotus Notes Message Store**.
- When SCemail starts, it prompts for a password, even if you type one on the command line, regardless of the -gui parameter.
- Do not install Microsoft Office 97 on the machine that is running Lotus Notes and SCemail. Office 97 upgrades MAPI automatically to a version that does not work well with Lotus Notes, and may not work with other MAPI service providers. This restriction holds for Lotus Notes 4.5, but can be removed in a later version.

If you use Lotus cc:Mail, the following restrictions apply:

- You must have Lotus cc:Mail for Windows version 7 or later to work with MAPI. (This requires release 6, or DB8, postoffice).
- If the profile has a password, then you must pass the -gui flag when you start SCemail, otherwise SCemail terminates with an error. You can avoid this by selecting the **Remember Password** check box when you log on with a normal cc:Mail client.
- Periodically check the **Outbox** of SCemail's profile for deleted messages to be purged.
- SCemail only runs as a Windows service if the mail service providers are tightly coupled. This is true even if SCemail is started from ServiceCenter, as ServiceCenter runs as a Windows service. As of this writing, the only mail service provider that does this is Microsoft Exchange Server. For other mail service providers, you must run SCemail from an interactive desktop.

UNIX

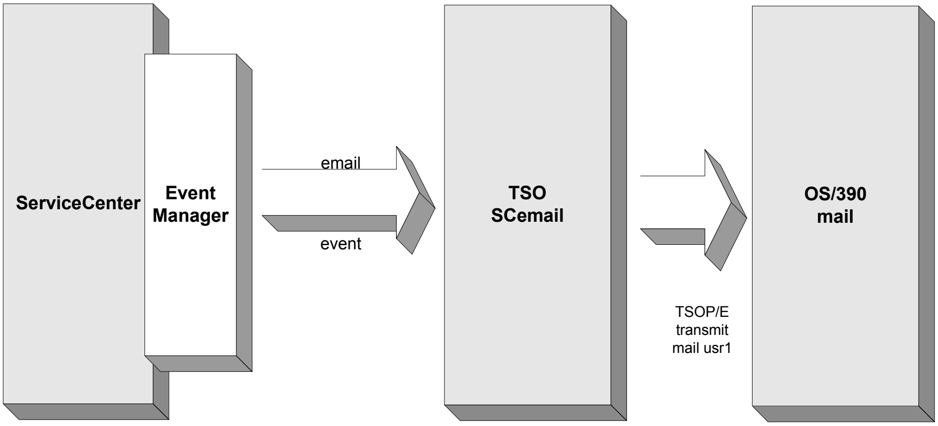
UNIX e-mail support consists of a daemon (scemail) process that reads output e-mail events and sends them to the addressed parties. The following example illustrates the SCAuto e-mail monitor.



- The mail monitor reads mail from ServiceCenter and delivers it to the SMTP network. Output mail is formatted by the ServiceCenter mail routines to contain the mailing address.
- Two files are created in the runtime directory: a checkpoint file and a log file. The checkpoint file maintains a pointer to the ServiceCenter **eventout** file that keeps redundant mail from being sent after a restart. If the checkpoint file is not found, all mail events are sent. The log file contains error and execution information.
- UNIX SCemail requires the standard UNIX MAIL utility.

OS/390

OS/390 e-mail support consists of a batch TSO address space that reads output e-mail events and sends them to the addressed parties using the **TRANSMIT** command. The following figure illustrates the OS/390 SCemail monitor.



Application	Description
message.fc	Called from Format Control, sends messages under user control.

Parameters

Name	Value	Default
index	Message Level: 1 Information 2 Action 3 Error	1
prompt	Message Class	msg
text	Message Text	none
name	User Name	operator()
string1	Message Name	none
number1	Message Number	none
query	Mail Class	none
names.1	Mail Target	none

Programming considerations

- In text mode, different Message Levels generate messages with different attribute settings. For example, error messages are red and information messages are white.
- Make sure that the Message Class matches one of the records in the **msgclass** file, for example, **problem close**. To send e-mail, for example, you must have a **msgclass** record with a type of **email** for the Message Class name specified.
- The Message Text can be either a string or an array. You can generate an array of the screen contents using the **genout**() function, for example, and then insert lines of text at the top of the array. For information regarding setting the appropriate array properties, go to the *Administering ServiceCenter* online help.
- The User Names can contain either a list of operator names or a single operator name. For internal (SC) messages, the names in User Names must be operator Ids defined in the operator table. For e-mail type messages (Message Class **email**) the User Names must be either operator IDs defined in the operator table or **contact.names** defined in the **contacts** file, with an e-mail address specified in the relevant table.

- The Message Name parameter is used to identify the message. In SC applications it is usually the name of the application or application area that generates the message. This parameter is not required.
- The Message Number parameter is used to identify a message within the area specified by the Message Name parameter. This parameter is not required.
- The Mail Class parameter is used within the Incident Management applications to identify the problem number so that mail already sent can be selected and updated. If used, it should contain the string pm.main and the Mail Target should also be supplied. This parameter is specific to Incident Management and is not required.
- The Mail Target parameter, when used, must contain the problem number (in number form). This parameter is specific to Incident Management and is not required.

Sending e-mail

Using Format Control

SCAutomate supports a generic e-mail function. You can write e-mail events to the eventout file using a subroutine call to message.fc from Format Control.

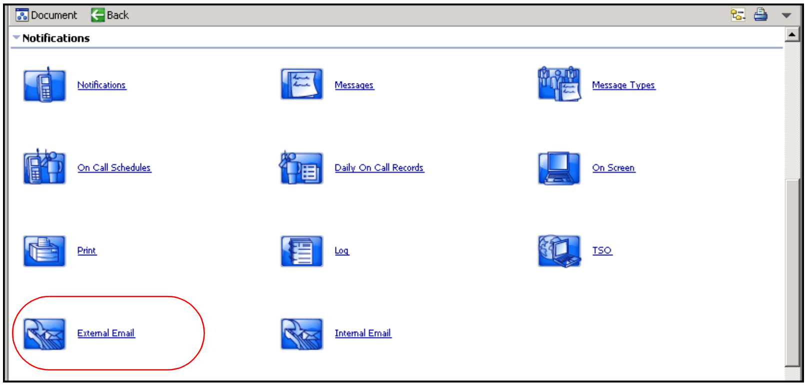
Parameter Name	Parameter Value
index	1
prompt	msg
text	Message Text
name	operator()
string1	Message Name
number1	Message Number
query	Mail Class
names,1	Mail Target

From Incident Management

Incident Management uses message classes to determine how messages are handled when incidents are opened, updated (including escalation) and closed.

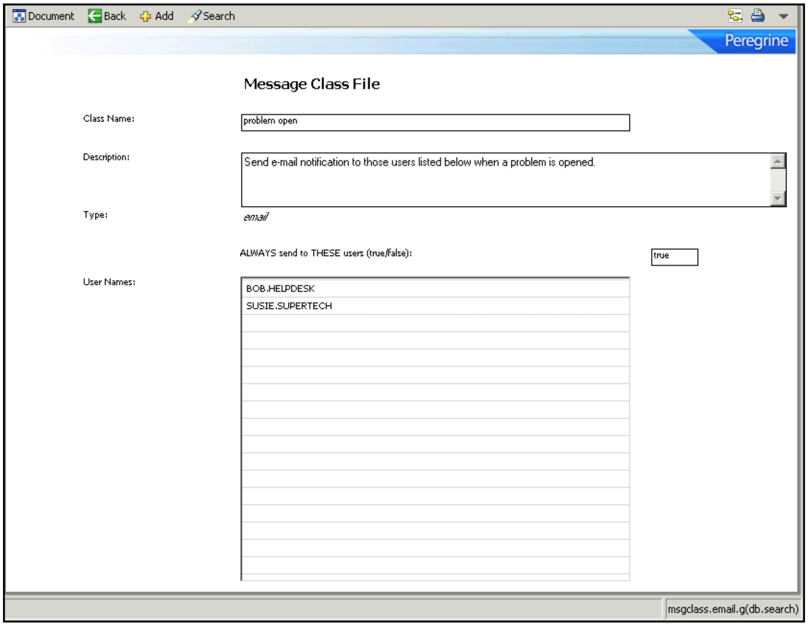
To configure ServiceCenter to always send e-mail to all members of the assignment group when an incident is opened

1 From the ServiceCenter main menu, click **Utilities** > **Administration Notifications.**



2 Click **External email.**

3 Create a record with a **Class Name** of **problem open**.



Message Class File

Class Name: problem open

Description: Send e-mail notification to those users listed below when a problem is opened.

Type: email

ALWAYS send to THESE users (true/false): true

User Names:
BOB.HELPDESK
SUSIE.SUPERTECH

msgclass.email.g(db.search)

4 Click **Add** to save the record.

5 To send e-mail with the same rules upon update, escalation and close, use the same procedure to add records for **problem update** and **problem close**.

Note: If you need more filtering on when to send e-mail, for example, if you only want to send e-mail to the **Contact Name** when a problem is closed, see *Using Format Control* on page 111 to use the Format Control for the category and function used (for example, problem.software.close).

8 CHAPTER Format Control Options

This chapter discusses using Format Control to generate output in Event Services.

This chapter contains the following sections:

Generating eventout records

Use Format Control to generate **eventout** records in Incident Management and Inventory and Configuration Management.

Format Control

Application	Description
axces.write	Called from Format Control, builds an **eventout** record that the SCAutomate interface uses.

Parameters

Name	Value	Default
record	The record to be written	none
name	The name of registration type	none
string1	The separation character	^
text	The system sequence ID	system generated
prompt	The user sequence ID	none
query	The user name	operator name

Programming considerations

- The record parameter is required. The application closes if this parameter is not provided.
- The registration name must exist in the eventregister file. If it does not, the application closes.
- If no eventregister record with a type of output can be found, the input registration record is used.
- Mapping is defined either by the format name or the map name. For most SCAuto/SDK events, use the Map Name to properly format fields.

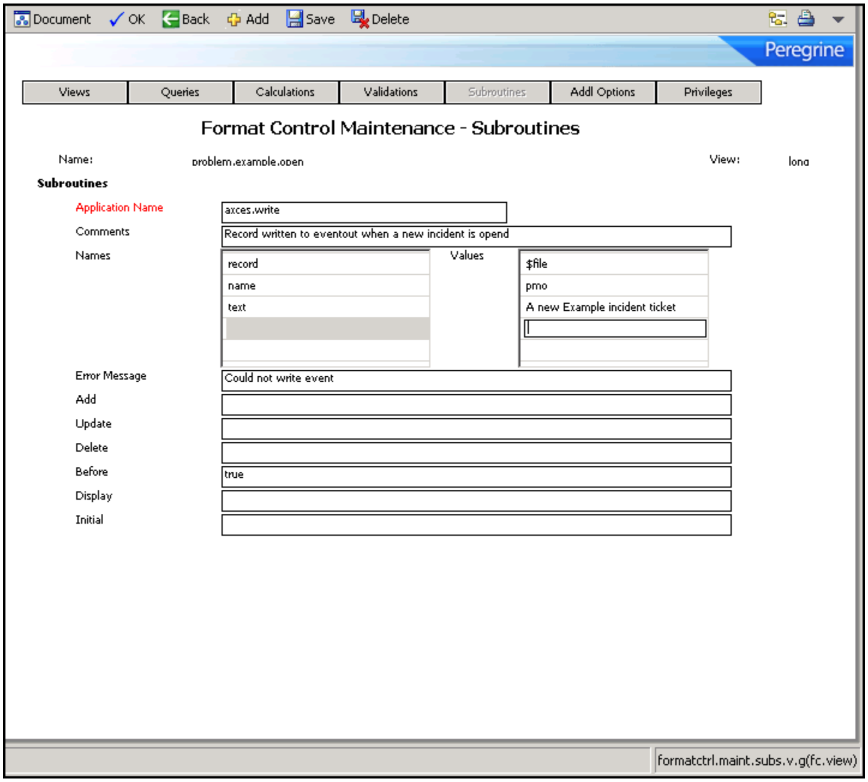
- If you define a separation character, make sure it is not one that occurs naturally in fields in the event.
- ServiceCenter generates the system sequence ID unless you supply one. The maximum length is 16 characters.

Incident Management

When Event Services opens, updates, or closes problems, a record may be written to the eventout file. This record contains information from the problem (described in the output eventmap record for the event) that is passed to an external process using the SCAuto/IPAS external interface. You can elect to write to the eventout file when Help Desk operators open and close tickets so that the information is passed to the external interface.

The **axces.write** application creates a character string of fields from a structure and writes them to **eventout**. An Event Registration record identifies the event type and the name of the Event Map records used to define which fields are selected from the record. The application is called as a Format Control subroutine passing two parameters; the first is the record from which data is mapped, and the second is the Event Type, as defined in the Event Register. For example, to write an **eventout** record when an **example** type incident is opened, use the following parameters.

Parameter Name	Parameter Value
record	$file
name	pmo



Format Control Maintenance - Subroutines

Name: problem.example.open View: long

Subroutines

Application Name: axces.write

Comments: Record written to eventout when a new incident is opend

Names	Values
record	$file
name	pmo
text	A new Example incident ticket

Error Message: Could not write event

Add:

Update:

Delete:

Before: true

Display:

Initial:

formatctrl.maint.subs.v.g(fc.view)

To write to the **eventout** file on **problem close**, the Format Control is attached to the **problem.example.close** format. In each case, the subroutine is called if the condition for **add** returns **true**.

Note: The Incident Management category **example** writes an **eventout** record for each open, update and close action.

Note: The standard event requires that certain fields populate in a particular position in the information passed to **eventout**:

- The first position is reserved for the e-mail address.
- The second position is reserved for the incident number.
- The fourth position is reserved for a time stamp (such as problem open or problem close time).
- The eighteenth position is reserved for the logical name of the device.
- The thirty-fifth position is reserved for the network name of the device.

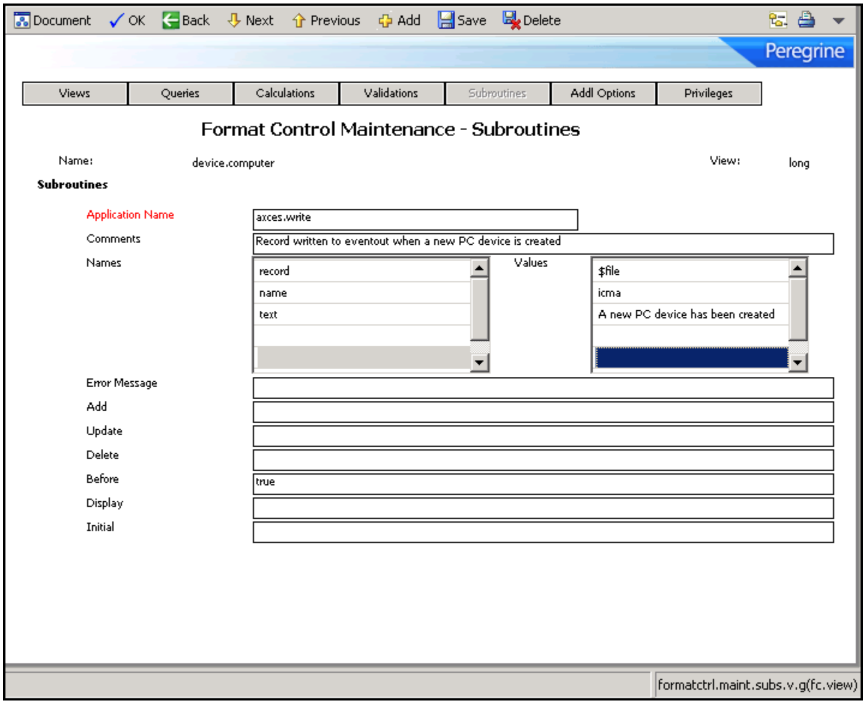
For standard events, these fields must be populated and must remain in their relative positions in the character string. The **eventmap** records for **output** define and maintain this information.

Inventory and Configuration Management

When Event Services adds, updates, or deletes inventory items, a record may be written to the **eventout** file. This record contains information from the device record (described in the output **eventmap** record for the event) that is passed to an external process using the SCAuto external interface. You can elect to write to the **eventout** file when operators maintain inventory items so that the information passes to the external interface.

The **axces.write** application creates a character string of fields from a structure and writes them to **eventout**. An Event Registration record identifies the event type and the name of the Event Map records that defines which fields are selected from the record. The application is called as a Format Control subroutine passing two parameters; the first is the record from which data is mapped, and the second is the Event Type, as defined in the Event Register. For example, to write an **eventout** record when a new device is added, use the following parameters.

Parameter Name	Parameter Value
record	$file
name	icma



Format Control Maintenance - Subroutines

Name: device.computer
View: long

Subroutines

Application Name: axces.write
Comments: Record written to eventout when a new PC device is created

Names	Values
record	$file
name	icma
text	A new PC device has been created

Error Message:
Add:
Update:
Delete:
Before: true
Display:
Initial:

formatctrl.maint.subs.v.g(fc.view)

Note: The Inventory device **type example** writes an **eventout** record for each add, update and delete operation.

Generating page messages

Format Control

SCAutomate supports a generic **page** function. Page events are written to the **eventout** file using a subroutine call to **axces.page** from Format Control.

Application	Description
axces.page	Called from Format Control, builds an **eventout** record that the Telalert pager axces interface uses.

Parameters

Name	Value	Default
name	The name of the contact	none
prompt	The numeric message	none
text	The alphanumeric message	none
string1	The separation character	^
query	The page response code	none
values	a list of addressees	none
names,1	a pager phone number	none
names,2	a pager PIN number	none
names,3	the name of a group	none

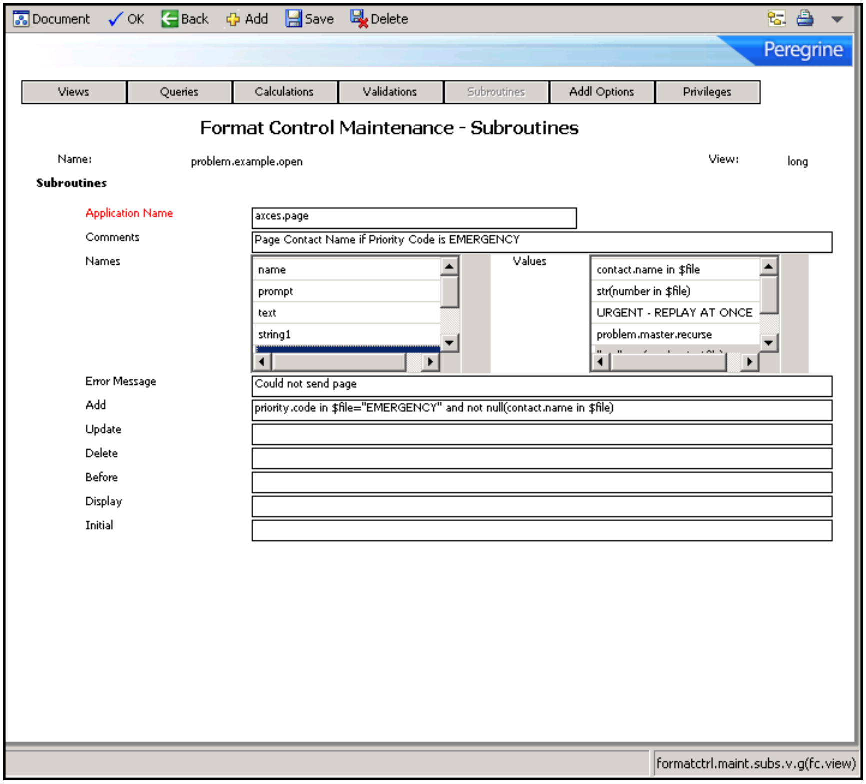
Programming considerations

- The **name** parameter or the **names, 3** parameter or the **values** parameter or the **names, 1** parameter is required. The application closes if one of these parameters is not provided.
- If more than one of the name parameters (that is, name, values and names,3) is provided, all receive a page as long as the associated contacts or operator record contains a pager phone number. Duplicate names receive only one page.
- The output event substitutes "" whenever a field is NULL except where noted.

- The output event concatenates fields from the contacts record as follows: Pager Vendor (telalert if NULL), Pager Name, Pager Group, Pager Type, Pager Phone #, Pager Pin #, Voice Mailbox, Numeric Message, Text Message. Fields are separated by the separation character.
- If a Pager Group is identified in the contacts record, the Pager Phone # is not passed.
- The page event is written directly to the eventout file.
- The group referred to by the **names,3** parameter is defined in the distgroup file with a type of page.
- While you can pass a pager phone number and a message to axces.page, usually a contact or operator name is provided since the pager instructions are stored in the contacts file.
- If there is no record in the contacts or operator file matching the value passed in the contacts parameter (or one of the entries in the values parameter, or one of the operators defined in the group named in the **names,3** parameter), a page event is not processed. There are fields in the contacts file that define pager vendor, phone number, PIN, and so on. Complete these fields properly for successful paging to occur.
- If a parameter is passed in the query parameter, the pageresp input event uses it to identify the type of event processing that occurs. For example, to update a particular problem with the response from a page, pass pm and the problem number (for example, pm9700123). The registration record determines the application to call by examining the data in the first position of the **evfields** field.

Incident Management

Format Control determines rules for sending a page when opening, updating or closing problems. For testing purposes, the category called **example** sends a page upon problem open if the Contact Name field is completed. To extend the service to other categories (or upon update, close or alert), access their associated Format Control and copy information from the problem.example.open Format Control record's subroutine definition for axces.page. For example, to page the Contact Name when a software problem reaches each alert stage, copy the axces.page subroutine definition from the problem.example.open Format Control record to the problem.software.alerts Format Control record.



Format Control Maintenance - Subroutines

Name: problem.example.open

View: long

Subroutines

Application Name: axces.page

Comments: Page Contact Name if Priority Code is EMERGENCY

Names	Values
name	contact.name in $file
prompt	str(number in $file)
text	URGENT - REPLAY AT ONCE
string1	problem.master.recurse

Error Message: Could not send page

Add: priority.code in $file="EMERGENCY" and not null(contact.name in $file)

Update

Delete

Before

Display

Initial

formatctrl.maint.subs.v.g(fc.view)

Sending fax messages

SCAutomate supports a generic **fax** function using the Replix FAX product. You can write fax events to the **eventout** file using a subroutine call to **axces.fax** from Format Control or from the Send a FAX button on the Event Services menu. You can also send any report or any mail message as a fax.

To support report Fax output, a record of type **FAX** must exist in the ServiceCenter **config** table. This record limits the number of pages that a fax message sends. You must supply the device name at the time the report (or printout) is generated. Fax messages generated from the Send a Fax button or from ServiceCenter mail, or using Format Control, do not require a config record. By definition, their size cannot exceed 32,000 bytes.

Format Control

Application	Description
axces.fax	Called from Format Control, builds an **eventout** record used by the Replix FAX axces interface.

Parameters

Name	Value	Default
names,1	The name of the sender	none
name	The name of the recipient	none
prompt	The FAX phone number	none
string1	The separator character	^
query	The name of the company	none
text	The format name or text string	none
names,2	The FAX title	none
record	The record variable	none

Programming considerations

- The name or prompt parameter is required; the application will exit if one of these parameters is not provided.
- If the contacts file is searched for a record with contact.name equal to the value passed in name. If no record is found, or if the selected record does not have a fax number defined, the fax is not sent.
- If a record variable is passed in the record parameter, pass the format name in the text parameter. The application uses *genout()* to build the fax output. Alternatively, you can pass a string in text; the string must use the pipe symbol (|) to separate lines of text.
- The output is written directly to the eventout table.

Creating output events

Format Control

You can use ServiceCenter's Format Control processing to create output events based on business rules. These events include paging, sending e-mail messages, and sending Fax documents. For more complete information and examples of Format Control utilities within the ServiceCenter and SCAutomate environments, go to the *Administering ServiceCenter* online help topics.

Application	Description
axces.fax.msg	Called from Format Control, builds schedule record that sends a fax.

Parameters

Name	Value	Default
file	A completed mail record	none
boolean1	The background flag	false

Programming considerations

- You must pass only a mail record to this application. In Format Control, you can set one up using secondary queries and using a query of false.
- The file parameter is required; the application closes if this parameter is not provided. Pass the file variable that contains the mail record.
- The user.array field in the file variable must be populated with at least one name.
- Both the contacts and the operator tables (in that order) are searched for each name in the user.array field; if no fax number is defined in the selected record (or if no record is selected) and the background flag is false, a prompt allow you to enter the recipient name and telephone number.
- A separate fax is sent to each name in the user.array field.
- Records are added to the spool file, and the background spool scheduler uses runoff to add records to the eventout file.
- A FAX config record must exist.
- The runoff application must have a compile date later than 5/14/96; reference SCR 7343.

APPENDIX

A Basic Troubleshooting

If you followed all the directions and are still encountering issues with the SCAutomate implementation, refer to the following common questions. Check these items and resolutions before contacting Peregrine Customer Support.

Frequently asked questions

Why are no problems opening, even though there are pmo records in the Event Input queue?

1. Verify the records in the queue have processed.
   - If the records have processed, there should be no **Event Time** value.
   - The **Status** field should contain a value.
   - Any messages should appear in the **Messages** field.
2. Verify there is an active event agent.
   a. Click Agent Status from the Services menu of the Event Services menu.
   b. Open the event agent.
      - The **Stop** button should be enabled.
      - A **Start Time** and an **Idle Time** should be displayed.
   c. Click **Refresh** to reset idle time to 00:00:00. It should begin increasing again.

d If the **Start** button is enabled and there is no **Start** and **Idle Time**, click **Start** and wait until the *problem* agent recycles.

3 Verify the following, and then wait for the event processor to recycle:

a The event schedule record exists.

b The **Class** field has a value of event.

c The **Status** field has a value of rescheduled.

4 If there is an active event agent, check the Event Registration table.

- Are there entries for Event *pmo* with a Type of *input*?
- Is the Execute Condition *true*?
- Compare the content of the *pmo* registration to the values documented in *Reviewing event registration* on page 26..

5 Verify there are event maps matching the **Event Map Name** values in the registration record.

- The same rules apply to all event types, not just *pmo*.

6 Verify that an active category is provided.

Why am I not receiving email even after installing ServiceCenter and opening a problem?

1 Verify you are a member of the assignment group for the problem.

If not, you will not receive notification of any kind.

2 Determine whether you are attempting to send mail to yourself when you open a problem.

ServiceCenter does not send mail to the individual who is opening, updating or closing a problem, regardless of their membership in the assignment group.

3 Log on as someone else.

4 Open a new problem.

5 Determine whether the operator to whom you are sending mail has an email address specified in his or her operator record.

6 Make sure it is correct.

7 Check the **Message Class** file for External Email records.

- Is there one for *problem open*?

8 If not, add one.

9 Verify there are records in the event output queue with a **type** of *email*.

10 If so, determine whether the scemail agent or another email agent is active.

   a Click **Agent Status** on the Services tab of the Event Services menu.

   b Open the Event agent.

   The Stop button should be enabled, and a **Start Time** and an **Idle Time** should be displayed.

   c Click **Refresh** to reset idle time to 00:00:00. It should begin increasing again.

   d If the **Start** button is enabled and there is no **Start** and **Idle Time**, click **Start** and wait until the *problem* agent recycles.

11 Determine if there is an output type event registration record for *email*.

12 Compare its contents to those described in *Reviewing event registration* on page 26.

13 If the SCEMAIL agent or another email agent is active and you still do not receive mail, kill the agent.

14 Open a problem and check the event output queue for new events with a type of *email*.

15 If a new email event is added to the queue, restart the SCEMAIL agent or another email agent.

When the mail has been sent, the event will be either deleted (if the -d flag is set) or updated.

Important: Always check the ServiceCenter Message Log and any external log files for errors. All SCAutomate errors are logged with a class of *event management errors*.

How do I send email only when I open problems with a priority code of "emergency"?

1 Click the **Administration** button on the Utilities tab of the ServiceCenter main menu.

2 Click the **External email** button to open the message class file.

3 Remove any External Email record for *problem open*.

4 Return to the ServiceCenter main menu.

5 Click the **Tools** button on the Utilities menu.

6 Click the **Macro** button.

7 Search for the incidents macro that sends the email.

8 Change the **Condition** field value to:

```
nullsub(priority.code in $L.new, ““)=”1”
```

How do I know mail sent to myself was received?

1 From the ServiceCenter main menu, click the Mailbox icon.

2 Click **Read Mail.**

3 Click **All Mail.**

Your message should appear in the list of mail messages.

How do I quickly test sending a fax message?

1 From the Services tab of the Event Services Menu, click **Send a FAX.**

2 Complete a message.

3 Click **FAX.**

4 Review the event output queue for an event of type *fax.*

How do I quickly test whether the SCAuto Pager is properly installed?

1 From the Services tab of the Event Services Menu, click **Send a Page.**

2 Complete a message.

3 Click **Page.**

4 If you are not paged within a minute or two, check to make sure the SCAUTO agent is active. Use the following procedure to do so:

- Click the Status button on the ServiceCenter main menu.

  There should be an entry under **User Name** for SCAUTO.

5 If SCAUTO is not active, you can start it if you are running an express client or are logged on from the server using `scenter`.

a Click **Start Scheduler.**

b Click on the entry for *scauto.startup*.

6 If the SCAUTO agent is active and you still do not receive a page,

a kill the agent, by placing a **k** in the command column beside agent.

b Click **Execute Commands.**

c Send a new page and check the event output queue for new events with a type of *page*.

7 If a new page event is added to the queue, restart the SCAUTO agent.

When the page has been sent, the event will be either deleted (if the -d flag is set) or updated.

How do I test sending a problem to my external program once SCAuto/SDK is installed?

1 From the Services tab of the Event Services Menu, click **Write an Output Event**.

2 Click **Problem**.

The first record in the probsummary file is written to the eventout queue.

3 Open a problem using the category **example**.

4 Note the problem number.

5 Open the Event Output queue.

6 Search for an event with a **Type** of *pmo* and a **Fields** field beginning with ^ followed by the problem number of the problem created in step 3.

How do I test sending a new device to my external program once SCAuto/SDK is installed?

1 From the Services tab of the Event Services Menu, click **Write an Output Event**.

2 Click **Inventory**.

The first record in the device file is written to the eventout queue.

3 Add a new device of type **example.**

4 Note the Logical Name.

5 Open the Event Output queue.

6 Search for an event with a **Type** of *icma* and a **Fields** field beginning with the Logical Name added in step 3.

How do I set the category from my message when I am opening problems via email?

1 Put each field assignment on a separate line in your mail message, uniquely identified by a label.

2 Use mapping expressions to extract the information and populate the appropriate fields in the problem

Example:

```
The mail message looks like this:
Fri, 12 Jan 01 14:40:41 -08:00
Re: Test to assign a category
John Jones <john@mac.acme.com>
CATEGORY: example
This is line 1 of the text of mail.
This is line 2 of the text of mail.
```

In the eventin record, the **evfields** field should appear as follows:

```
xjohn^^^^Fri, 12 Jan 01 14:40:41 -08:00|Re: Test to assign a category|John
Jones<john@hp800.peregrine.com>|CATEGORY:
example||This is line 1 of the text of mail.|This is line 2 of the text of
mail.|^^^^^^^^^^^^^John Jones <john@mac.acme.com>^^
```

- In the *problem open* event map record for the **category** field, enter the following *Initialization* statements:

  ```
  $axtype=type in $axces.target
  if (index(“axmail”, evuser in $axces)>0) then $axtype=type in $axces.target
  if (index(“axmail”, evuser in $axces)>0) then ($ax.action=denull(action in
  $axces.target);$axl=lng($ax.action))
  if (index(“axmail”, evuser in $axces)>0) then for $axpos = 1 to $axl do
  ($axt=$axpos in $ax.action;if $axt#”CATEGORY then ($axtype=substr($axt,
  10, lng($axt) - 9);$ax.action=delete($ax.action, $axpos);action in
  $axces.target=$ax.action))
  ```

3 Then enter these Instructions:

```
if (index(“axmail”, evuser in $axces)>0) then category in $axces.target=$axtype
cleanup($axtype);cleanup($axt);cleanup($axpos);cleanup($axl)
cleanup($ax.action)
```

This procedure (substituting other field names) allows specification of any problem field values within the body of the email message as long as the map record in which the instructions are entered has a higher sequence number than that of the **action** (or **update.action**) field.

Can I have my problem events processed separately, so they aren’t held up by other events?

1 Copy the event agent to a new agent called (for example) *probevent*.

2 Copy its associated info record, substituting *probevent* for *event*.

3 Modify the event agent’s query field to say `evtype~#”pm”`.

4 Modify the probevent agent’s query field to say `evtype#”pm”`.

You can do the same thing for output events created by the SCAuto/OS/390 agents.

APPENDIX

B Common Events

ServiceCenter delivers out-of-box events with the Event Services module. The following tables identify some of the more commonly-used events within ServiceCenter.

Common module events

Use the tables as a quick reference for each event's function.

Service Management events

Event	Description
smin	Service Management incoming service request or help issue.
smout	Service Management Output event.

Incident Management events

Event	Description
pmo	Opens an incident.
pmu	Updates an incident.
pmc	Closes an incident.

Inventory Management events

Event	Description
icma	Adds an inventory item to the device file or updates the item if it already exists in the file.
icmu	Updates an inventory item.
icmd	Marks an inventory item for deletion.
prgma	Adds a software inventory item.
prgmu	Updates an inventory item.
prgmd	Deletes a software item.

Change Management events

Event	Description
cm3rin	Used for all incoming change events.
cm3rout	Created when a cm3message is activated. It represents a generic Output message from a change phase.
cm3rinac	Used for acknowledging success in processing an incoming cm3rin event.
cm3tin	Used for incoming change events that communicate a generic message to a change task.
cm3tout	Created when a cm3message is activated. It represents a generic Output message from a change task.
cm3tinac	Used for acknowledging success in processing an incoming cm3tin event.

Request Management events

Event	Description
rmoin	Request from an external application to open an order in Request Management.
rmoappr	Request from an external application to enter an approval for an existing order from one of the order's required approval group, or an approval user.
rmlin	Request from an external application to enter a new line item in an existing order in Request Management.
rmqin	Request from an external application to enter a new quote in an order in Request Management.
rmqappr	Request from an external application to enter an approval for a quote in an existing order from one of the quote's required approval group or an approval user.

Service Level Management events

Event	Description
outagestart	Request from an external application to begin an outage against a device with an SLA.
outageend	Request from an external application to end an outage against a device with an SLA.
slaresponse	Request from an external application to enter a response time metric against a device with an SLA.

Standard events

ServiceCenter event registration currently supports events enabling integration with ERP, SAP, and other external system interfaces. The following section contains the RAD routines that each event calls. Search the *List: RAD routines* topic in the ServiceCenter online help for the parameters of these routines. Where applicable, the parameter descriptions that follow contain information specific to the event.

CTSCPY *(1)*

Event type	Output
Description	This event to SAP uses eventmap cm3tctsc. It is a generated message to SAP to copy a CTS Transport from one system to another.
Routine called	axces.write

CTSCPY *(2)*

Event type	Input
Description	This event from SAP uses eventmap cm3tctsc. It processes the acknowledgment of SAP CTS Copy messages.

Routine called	axces.cm3	
Parameters	**Name** string1	**Description** In this case, use cm3r.
	text	The action is update.
	boolean1	The value is false
	query	The query is: "erp.parent.unique.id=\""+str(2 in evlist in $axces)+"\" and erp.development.sid=\""+str(4 in evlist in $axces)+"\" and erp.sid=\""+str(5 in evlist in $axces)+"\""

CTSIMP *(1)*

Event type	Output	
Description	This event to SAP uses eventmap cm3tctsi. It builds a message to request a specific SAP Instance perform an Import of a given Transport.	
Routine called	axces.write	
Parameters	**Name** prompt	**Description** In this case, use erp.gateway.id in $L.change.

CTSIMP *(2)*

Event type	Input
Description	This event from SAP uses eventmap cm3tctsi. It processes the Input acknowledge message from SAP regarding Import of Transport.
Routine called	axces.cm3

Parameters	Name	Description
	string1	In this case, use cm3t.
	text	The action is close.
	boolean1	The value is false
	query	The query is: "erp.parent.unique.id=\""+str(1 in evlist in $axces)+"\" and erp.sid=\""+str(3 in evlist in $axces)+"\" and erp.client=\""+str(4 in evlist in $axces)+"\""

CTSIMP2

Event type	Output
Description	This event handles scheduling of Output Import events to SAP using eventmap cm3tctsi.
Routine called	axces.cm3.cts.write

Parameters	Name	Description
	record	Since this is invoked from cm3.message.pro, use $L.change
	name	In this case, use erp.sid in $L.change.
	prompt	In this case, use erp.client in $L.change.
	query	Use sap cts import scheduled for rescheduled import messages.
	boolean1	Use erp.override.reschedule in $L.change.

Note: Out-of-box, CTSIMP2 is called whenever approval is received for a SAP Instance Import task. The routine axces.cm3.cts.write determines an acceptable time to send a message to the target system. With the acceptable time calculated, a schedule record is generated to send the CTSIMP message at the calculated time.

CTSRQCLS *(1)*

Event type	Input	
Description	This event to SAP uses eventmap cm3rcts. It sends a message to a SAP instance instructing it to release a transport.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use erp.development.gateway.id in $L.change.

CTSRQCLS *(2)*

Event type	Input	
Description	This event from SAP uses eventmap cm3rcts. It is a received message from SAP acknowledging transport release.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	string1	In this case, use cm3r.
	text	In this case, use update.
	boolean1	The value is false
	query	The query is: "header,number="+str(1 in evlist in $axces)+" and header,last=true"

CTSRQOPN *(1)*

Event type	Output
Description	This system event sent to SAP uses the cm3rcts eventmap. It sends a message to a SAP instance instructing it to open a SAP Transport Request with certain ServiceCenter supplied data.

Routine called	axces.write	
Parameters	**Name** prompt	**Description** In this case, use erp.development.gateway.id in $L.change.

CTSRQOPN *(2)*

Event type	Input	
Description	This event from SAP uses eventmap cm3rcts. It is a received message from SAP acknowledging Transport Request creation. It closes the first phase of the Change and updates fields with data returned from SAP.	
Routine called	axces.cm3	
Parameters	**Name** string1	**Description** In this case, use cm3r.
	text	This registration is a result of an acknowledgement message of a ServiceCenter-originated Request Open. In this case, the proper action is close [this phase and advance to the next].
	boolean1	The value is false
	query	The query is: "header,number="+str(1 in evlist in $axces)+" and header,last=true"

CTSRQOPN *(3)*

Event type	Input
Description	This input event from SAP uses eventmap cm3rctso. It is a received message from SAP sent when a Transport Request opens on the SAP side without first opening within ServiceCenter. It causes a Change to open within ServiceCenter with data received from SAP.
Routine called	axces.cm3

Parameters	Name	Description
	string1	In this case, use cm3r.
	text	Since this registration is only invoked when a Transport opens in SAP and must be opened in ServiceCenter, the value is open.
	boolean1	The value is false.
	query	The query is: "header,number="+str(1 in evlist in $axces)+" and header,last=true"

CTSRQUPD *(1)*

Event type	Output	
Description	This event to SAP uses eventmap cm3rcts. It sends a message to a SAP Instance to update certain Transport Request data elements.	
Routine called	axces.write	
Parameters	Name	Description
	prompt	In this case, use erp.development.gateway.id in $L.change.

CTSRQUPD *(2)*

Event type	Input
Description	This is an Input event from SAP. It uses eventmap cm3rcts. It is a received message from SAP sent when a Transport Request has been updated on the SAP side. It is either an acknowledgment of a ServiceCenter originated request or a notification of a SAP originated action.
Routine called	axces.cm3

Parameters	Name	Description
	string1	In this case, use cm3r.
	text	Since this registration conveys update information, the value is update.
	boolean1	The value is false.
	query	The query is: "header,number="+str(1 in evlist in $axces)+" and header,last=true"

CTSTKCLS *(1)*

Event type	Output	
Description	This event to SAP uses eventmap cm3tcts. It sends a message to SAP to close a Transport Task within SAP.	
Routine called	axces.write	
Parameters	Name	Description
	prompt	In this case, use erp.development.gateway.id in $L.change.

CTSTKCLS *(2)*

Event type	Input
Description	This event from SAP uses eventmap cm3tcts. It is a received message from SAP that a Transport Task closed on the SAP side. It is either an acknowledgment of a ServiceCenter-originated request or a notification on a SAP-originated action.
Routine called	axces.cm3

Parameters	Name	Description
	string1	In this case, use cm3t.
	text	In this case, use close.
	boolean1	The value is false.
	query	Indicates the query to use to find the pre-existing record that this Input event updates. The query is: "header,number="+str(10 in evlist in $axces)+" and header,last=true"

CTSTKOPN *(1)*

Event type	Output	
Description	This is event to SAP uses eventmap cm3tcts. It sends a message to SAP indicating that a Transport Task opened within SAP.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use erp.development.gateway.id in $L.change.

CTSTKOPN *(2)*

Event type	Input
Description	This event from SAP uses eventmap cm3tcts. It is an acknowledgment message from SAP indicating that a Transport Task closed on the SAP side.
Routine called	axces.cm3

Parameters	Name	Description
	string1	In this case, use cm3t.
	text	In this case, a Transport Task opened first on the SAP side. You must update ServiceCenter to reflect this information, so you must open a new task. The action is open.
	boolean1	The value is false.
	query	The query is: "header,number="+str(10 in evlist in $axces)+" and header,last=true"

CTSTKOPN *(3)*

Event type	Input	
Description	This event from SAP uses eventmap cm3tctso. This message indicates that a Transport Task opened on the SAP side within ServiceCenter's Change Management using data supplied from the SAP system.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	string1	In this case, use cm3t.
	text	In this case, a Transport Task opened first on the SAP side. You must update ServiceCenter to reflect this information, so you must open a new task. The action is open.
	boolean1	The value is false.
	query	The query is: "header,number="+str(10 in evlist in $axces)+" and header,last=true"

CTSTKUPD *(1)*

Event type	Output	
Description	This event to SAP uses eventmap cm3tcts. It is sent from ServiceCenter to SAP to update a Transport Task on the SAP side to match changes on the ServiceCenter side.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use erp.development.gateway.id in $L.change.

CTSTKUPD *(2)*

Event type	Input	
Description	This event from SAP uses eventmap cm3tcts. It is a message received from SAP when a Transport Task update on the SAP side. It can either be an acknowledgment of a ServiceCenter-originated update or notification of a SAP-originated update.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	string1	In this case, use cm3t.
	text	The value is update.
	boolean1	The value is false.
	query	The query is: "header,number="+str(10 in evlist in $axces)+" and header,last=true"

ERPHR *(1)*

Event type	Input
Description	This event establishes contact with the ERP system using eventmap contactserp.
Routine called	axces.database

Parameters	Name	Description
	prompt	In this case, use contactserp.
	string1	In this case, use contacts.
	text	In this case, use add.
	query	In this case, use contact.name=1 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is false.
	name	In this case, use operator.scauto.

ERPHR *(2)*

Event type	Output
Description	This event uses eventmap contactserp.
Routine called	axces.write

ERPSTATES *(1)*

Event type	Input
Description	This event determines the state of the ERP system using eventmap stateerp.
Routine called	axces.database

Parameters	Name	Description
	prompt	In this case, use stateerp.
	string1	In this case, use state.
	text	In this case, use add.
	query	In this case use, state.code=1 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is false.
	name	In this case, use operator.scauto.

ERPSTATES *(2)*

Event type	Input
Description	This event uses eventmap stateerp.
Routine called	axces.write

GetResRM

Event type	Input	
Description	This provides access to Request Management.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	text	In this case, use 3 in evlist in $axces.
	prompt	In this case, use evmap in $axces.register.
	query	In this case, it is not set.
	string1	In this case, the file is ocmq.
	boolean1	In this case, the value is true.

GetResRM

Event type	Output
Description	This provides access to Request Management.
Routine called	axces.write

GetResRML

Event type	Input	
Description	An input event that provides access to Request Management.	
Routine called	GetResRML	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	text	In this case, use 3 in evlist in $axces.
	prompt	In this case, use evmap in $axces.register.
	query	In this case, the query is not set.
	string1	In this case, ocml.
	boolean1	In this case, the value is true.

GetResRML

Event type	Output
Description	This provides access to Request Management.
Routine called	axces.write

HotNews

Event type	Output
Description	HotNews defines an eventout type of HotNews.
Routine called	None

ICMapplication

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm application.
	string1	In this case, device.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMapplication.

ICMcomputer

Event type	Input
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.
Routine called	axces.database

Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm computer.
	string1	In this case, join computer.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMcomputer.

ICMdevice

Event type	Input
Description	Use this event type when you add data records to the device file. These events use the icm device eventmaps.
Routine called	icm.process.event

Parameters	**Name**	**Description**
	prompt	In this case, use icm device.
	string1	In this case, device.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	The default value is false.
	name	In this case, use ICMdevice.

ICMdevicenode *(1)*

Event type	Input
Description	Use this event type to add or update information about a network node (a device that appears as a discrete item in a network) to the Inventory Configuration Management module. These events use the icm networkcomponents mappings.

Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm networkcomponents.
	string1	In this case, joinnetworkcomponents.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMnetworkcomponents.
	name,1	Specifies the formatctrl record to use for processing.

ICMdevicenode *(2)*

Event type	Input
Description	If you use the ICMdevicenode event to send information to ServiceCenter, after the initial database operation completes, this secondary event registration causes a logging record to write to the eventout table.
Routine called	axces.write

ICMdisplaydevice

Event type	Input
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use icm displaydevice.
	string1	In this case, joindisplaydevice.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMdisplaydevice.

ICMexample

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Routine called	axces.database	
Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use icm example.
	string1	In this case, joinexample.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMexample.

ICMfurnishings

Event type	Input
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.
Routine called	axces.database

	Name	Description
Parameters	record	In this case, the value is $axces.
	prompt	In this case, use icm furnishings.
	string1	In this case, joinfurnishings.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMfurnishings.

ICMhandhelds

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm handhelds.
	string1	In this case, joinhandhelds.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMhandhelds.

ICMmainframe

Event type	Input
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.
Routine called	axces.database

Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm mainframe.
	string1	In this case, joinmainframe.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMmainframe.

ICMnetworkcomp

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm networkcomponents.
	string1	In this case, joinnetworkcomponents.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMnetworkcomponents.

ICMofficeelec

Event type	Input
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use ICM application.
	string1	In this case, joinofficeelectronics.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMofficeelectronics.

ICMserver

Event type	Input
Description	Use this event type to add or update information about a server to the Inventory Configuration Management module. These events use the icm computer mappings (since servers are a subtype of computers, you can reuse the mappings.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use icm computer.
	string1	In this case, joincomputer.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMserver.
	name,1	Specifies the formatctrl record to use for processing.

ICMsoftwarelicense

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm softwarelicense.
	string1	In this case, joinsoftwarelicense.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMsoftwarelicense.

ICMstorage

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm storage.
	string1	In this case, joinstorage.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMstorage.

ICMtelecom

Event type	Input	
Description	ServiceCenter inventory regulation event when a device of this type is added to the system.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use icm telecom.
	string1	In this case, jointelecom.
	text	In this case, use add.
	query	In this case, logical.name=1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use ICMtelecom.

IND

Event type	Input	
Description	An event that adds or updates inventory items to the device file.	
Routine called	scauto.inventory	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use device.
	text	In this case, use add.
	query	In this case, logical.name=7 in $axces.fields.
	boolean1	In this case, the value is true.
	name	In this case, use icma, the legacy name.

NDpmc

Event type	Input	
Description	This Network Discovery event closes automatically-generated incidents.	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use close.
	query	In this case, $ax.query.passed.
	boolean1	In this case, nullsub(evstatus in $axces, "close")~#"error".

NDpmc

Event type	Output	
Description	This event returns an incident number when Network Discovery closes an incident.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	string1	The delimiter character is ^.
	query	In this case, evuser in $axces.
	prompt	In this case, nullsub(evusrseq in $axces, evsysseq in $axces).

NDpmo

Event type	Input	
Description	This event from Network Discovery opens, updates, or reopens an incident.	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use open.
	query	In this case, $ax.query.passed.
	boolean1	In this case, nullsub(evstatus in $axces, "") ~#"error".
	cond.input	In this case, $ax.open.flag.

NDpmo

Event type	Output	
Description	This event returns an incident number when Network Discovery opens, updates, or reopens an incident.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	name	In this case, pmo.
	string1	The delimiter character is ^.
	query	In this case, evuser in $axces.
	prompt	In this case, nullsub(evusrseq in $axces, evsysseq in $axces).

PSSDELETE

Event type	Input	
Description	This event deletes selected records from a ServiceCenter file.	
Routine called	pss.delete	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use $L.name.

SALESQUOTE

Event type	Input	
Description	This event moves an eventin record to the eventout file and changes the evtype.	
Routine called	axces.move.intoout	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	name	In this case, SALESORDERPARSE.

SAPGRT

Event type	Output
Description	This goods receipt Output event calls no application. It submits receipt notification to SAP system for processing.
Routine called	None

SAPGRT

Event type	Input	
Description	This goods receipt event calls submits receipt notification to SAP system for processing.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	text	In this case, use update.
	prompt	In this case, use evmap in $axces.registrater.
	query	In this case, use number=21 in evlist in $axces.
	string1	In this case, use ocml.
	name	In this case, use falcon.

SAPGTE

Event type	Input	
Description	This event updates existing line items.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	text	In this case, use update.
	prompt	In this case, use evmap in $axces.registrater.
	query	Provides conditional query. In this case, use number=21 in evlist in $axces.
	string1	In this case, use ocml.
	name	In this case, use falcon.

SAPHR *(1)*

Event type	Input	
Description	Event processing edits to contact file originating in ServiceCenter, routed through SAP, and returned to ServiceCenter.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	prompt	In this case, use contactssap.
	string1	In this case, use contacts.
	text	In this case, use add.
	query	Provides conditional query. In this case, use contact.name=2 $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is false.
	name	In this case, use operator.scauto.

SAPHR *(2)*

Event type	Output	
Description	This event routes contact file changes to SAP.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use contactssap.
	string1	In this case, use contacts.
	text	In this case, use add.
	query	In this case, use contact.name=2 in $axces.fields.
	name	In this case, use operator.scauto.

SAPHRMD

Event type	Input	
Description	This event processes SAP-originating contacts file changes.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	prompt	In this case, use contactssap.
	string1	In this case, use contacts.
	text	In this case, use add.
	query	In this case, use contact.name=2 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is false.
	name	In this case, use operator.scauto.

SAPORD

Event type	Output
Description	This sales order event to SAP calls no application. It routes order information to SAP for processing.
Routine called	None

SAPORD

Event type	Input
Description	This sales order event from SAP breaks events into appropriate constituent parts.
Routine called	axces.sap.hybrid.evin

SAPORDQ

Event type	Input	
Description	This is the header component of the SAPORD event.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	Header component that the SAPORD event creates.
	text	In this case, use update.
	query	In this case, use number=9 in evlist in $axces.
	string1	In this case, use ocmq.
	name	In this case, use falcon.

SAPQTE

Event type	Output
Description	This sales quote event from ServiceCenter calls no routine.
Routine called	None

SAPQTE

Event type	Input
Description	This sales quote event from SAP breaks events into constituent parts.
Routine called	axces.sap.hybrid.evin

SAPQTEQ

Event type	Input	
Description	This is the header component of the SAPQTE sales quote.	
Routine called	axces.rm	
Parameters	Name	Description
	record	Header component that the SAPQTE event creates.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocmq.
	text	In this case, use update.
	query	In this case, use number=10 in evlist in $axces.
	name	In this case, use falcon.

SAPREQ

Event type	Output
Description	This purchase requisition event from ServiceCenter calls no application.
Routine called	None

SAPREQ

Event type	Input	
Description	This purchase requisition event is from SAP.	
Routine called	axces.sap.hybrid.evin	
Parameters	Name	Description
	record	Use the $axces variable.
	name	In this case, use SAPREQO.
	prompt	In this case, use sapreql.

SAPREQO

Event type	Input	
Description	This event is a SAPREQ component from SAP.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	Use the $axces variable.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocmo.
	text	In this case, use update.
	query	In this case, use number=5 in evlist in $axces.

ScAcBrand

Event type	Input	
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter vendor file to the corresponding AssetCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcBrand.
	string1	In this case, use vendor.
	text	In this case, use add.
	query	In this case, vendor = 1 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use vendor.

ScAcCompany

Event type	Input	
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter company file to the corresponding AssetCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcCompany.
	string1	In this case, use company.
	text	In this case, use add.
	query	In this case, customer.id = 1 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use company.

ScAcContacts

Event type	Input
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter contacts file to the corresponding AssetCenter file.
Routine called	axces.database

	Name	Description
Parameters	record	In this case, the value is $axces.
	prompt	In this case, use ScAcContacts.
	string1	In this case, use contacts.
	text	In this case, use add.
	query	In this case, contact.name = 1 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use contacts.

ScAcDept

Event type	Input
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter department file to the corresponding AssetCenter file.
Routine called	axces.database

	Name	Description
Parameters	record	In this case, the value is $axces.
	prompt	In this case, use ScAcDept.
	string1	In this case, use dept.
	text	In this case, use add.
	query	In this case, dept.id = 1 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use dept.

ScAcDevice

Event type	Input	
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter device file to the corresponding AssetCenter file.	
Routine called	scauto.inventory	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use device.
	text	In this case, use add.
	query	In this case, logical.name = 1 in $axces.fields.
	boolean1	In this case, the value is false.
	name	In this case, use icma.

ScAcLocation

Event type	Input
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter location file to the corresponding AssetCenter file.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcLocation.
	string1	In this case, use location.
	text	In this case, use add.
	query	In this case, location = 2 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use location.

ScAcModel

Event type	Input
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter model file to the corresponding AssetCenter file.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcModel.
	string1	In this case, use model.
	text	In this case, use add.
	query	In this case, part.no = 1 in $axces.fields.
	boolean1	In this case, the value is val("false",4).
	cond.input	In this case, the value is true.
	name	In this case, use model.

ScAcModelBundle

Event type	Input	
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter model file to the corresponding AssetCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcModelBundle.
	string1	In this case, use model.
	text	In this case, use add.
	query	In this case, part.no = 1 in $axces.fields.
	boolean1	In this case, the value is val("false",4).
	cond.input	In this case, the value is true.
	name	In this case, use model.

ScAcModelVendor

Event type	Input
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter modelvendor file to the corresponding AssetCenter file.
Routine called	axces.database

	Name	Description
Parameters	record	In this case, the value is $axces.
	prompt	In this case, use ScAcModelVendor.
	string1	In this case, use modelvendor.
	text	In this case, use add.
	query	In this case, part.no = 1 in $axces.fields and vendor = 2 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use modelvendor.

ScAcVendor

Event type	Input	
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter vendor file to the corresponding AssetCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcVendor.
	string1	In this case, use vendor.
	text	In this case, use add.
	query	In this case, vendor = 2 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use vendor.

ScAcVendorBACK

Event type	Input	
Description	This event allows ServiceCenter and AssetCenter to integrate data from the ServiceCenter vendor file to the corresponding AssetCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use ScAcVendor.
	string1	In this case, use vendor.
	text	In this case, use add.
	query	In this case, vendor=9 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is false.

ScFcOrderLine

Event type	Input	
Description	This event allows ServiceCenter and FacilityCenter to integrate data from the ServiceCenter omcl file to the corresponding FacilityCenter file.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	query	In this case, foreign.id=2 in evlist in $axces.
	string1	In this case, use ocml.
	boolean1	In this case, the value is false.

ScFcOrderLine

Event type	Output
Description	This event allows ServiceCenter and FacilityCenter integration.
Routine called	axces.write

TcScCompDel

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter company file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in TcScCompDel.
	string1	In this case, use company.
	text	In this case, use delete.
	query	In this case, customer.id=1 in $axces.fields.
	condition,1	In this case, the value is true.

TcScCompany

Event type	Input
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter company file to the corresponding TeleCenter file.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use TcScCompany.
	string1	In this case, use company.
	text	In this case, use add.
	query	In this case, customer.id= 1 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use company.

TcScContacts

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter contacts file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use TcScContacts.
	string1	In this case, use contacts.
	text	In this case, use add.
	query	In this case, contact.name = 1 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use contacts.

TcScDept

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter department file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use TcScDept.
	string1	In this case, use dept.
	text	In this case, use add.
	query	In this case, dept.id = 2 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use dept.

TcScDeptDel

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter department file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use TcScDeptDel.
	string1	In this case, use dept.
	text	In this case, use delete.
	query	In this case, dept=1 in $axces.fields.
	condition,1	In this case, the value is true.

TcScDeptdel

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter department file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use deptdel.
	string1	In this case, use dept.
	text	In this case, use delete.
	query	In this case, dept=1 in $axces.fields.
	condition,1	In this case, the value is true.

TcScLocation

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter location file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use TcScLocation.
	string1	In this case, use location.
	text	In this case, use add.
	query	In this case, location = 2 in $axces.fields.
	boolean1	In this case, the value is false.
	cond.input	In this case, the value is true.
	name	In this case, use location.

TcScLocDel

Event type	Input	
Description	This event allows ServiceCenter and TeleCenter to integrate data from the ServiceCenter location file to the corresponding TeleCenter file.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use TcScLocDel.
	string1	In this case, use location.
	text	In this case, use delete.
	query	In this case, location .full.name= 1 in $axces.fields.
	condition,1	In this case, the value is true.

WMI

Event type	Output	
Description	This event calls axces.write.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	record	Use the $axces variable.
	name	In this case, use WMI.
	string1	In this case, use ^.
	query	In this case, evuser in $axces
	prompt	In this case, nullsub(evusrseq in $axces, evsysseq in $axces).

WMI

Event type	Input	
Description	This event calls wmi.inventory.check.	
Routine called	wmi.inventory.check	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use device.
	text	In this case, use add.
	query	In this case, $L.temp.query
	boolean1	In this case, the value is true.
	name	In this case, use icma.
	cond.input	In this case, the value is val("true",4).

XIND

Event type	Input	
Description	This event calls scauto.inventory.	
Routine called	scauto.inventory	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use device.
	text	In this case, use add.
	query	In this case, logical.name=7 in $axces.fields.
	boolean1	In this case, the value is true.
	name	In this case, use icma, the legacy name.

approval

Event type	Output	
Description	This event sends approvals for Request Management and Change Management.	
Routine called	axces.write	

approval

Event type	Input	
Description	This event processes approvals for Request Management and Change Management.	
Routine called	es.approval	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	text	In this case, use ApprovalLog.
	name	In this case, evmap in $axces.register.
	cond.input	In this case, the value is false.

cm3rin

Event type	Input	
Description	Use this event for all incoming change events.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	record	Use the $axces variable.
	prompt	In this case, evmap.
	text	In this case, use 3 in evlist in $axces.
	boolean 1	In this case, the value is true.
	string1	In this case, use cm3r.

cm3rinac

Event type	Output
Description	This is sent if the write eventout is set to true. **Note:** This returns failed events so the calling application receives notification when an error occurs.
Routine called	axces.write

cm3rout

Event type	Output
Description	This is created when a cm3 message is created and you enter cm3rout in axces.out.
Routine called	axces.write

cm3tin

Event type	Input	
Description	Use this for all incoming change tasks.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	prompt	In this case, evmap in $axces.register.
	text	In this case, use 3 in evlist in $axces.
	boolean1	In this case, the value is true.
	string1	In this case, use cm3t.

cm3tinac

Event type	Output
Description	This is sent if the write eventout is set to true. **Note:** This returns failed events so the calling application is notified when an error occurs.
Routine called	axces.write

cm3tout

Event type	Output
Description	This is created when a cm3 message fires and you enter cm3tout in axces.out.
Routine called	axces.write

dbadd

Event type	Input	
Description	This adds an item to a specified ServiceCenter file when you satisfy the filter criteria. It updates the file if the item already exists.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	prompt	In this case, use scauto test.
	string1	In this case, use scautotest.
	text	In this case, use add.
	query	In this case, use field.1=1 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is true.
	name	In this case, use scautotest.

dbdel

Event type	Input	
Description	This deletes an item from a specified ServiceCenter file if the filter criteria are satisfied.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	prompt	In this case, use scauto test.
	string1	In this case, use scautotest.
	text	In this case, use delete.
	query	In this causes field.1=1 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is true.
	condition,1	In this case, the value is false.
	name	In this case, use scautotest.

dbupd

Event type	Input	
Description	This updates an item to a specified ServiceCenter file when you satisfy the filter criteria.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	prompt	In this case, use scauto test.
	string1	In this case, use scautotest.
	text	In this case, use update.
	query	In this case, field.1=1 in $axces.fields.
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is true.
	name	In this case, use scautotest.

email

Event type	Output	
Description	This is the standard interface to convert ServiceCenter mail to standard e-mail format.	
Routine called	axces.email	
Parameters	**Name** text	**Description** In this case, use ^.

email

Event type	Input
Description	This is the standard interface to receive external e-mail and convert to ServiceCenter mail.
Routine called	axces.email.receive

epmc

Event type	Input	
Description	This event uses the e problem close map to initiate the problem close process associated with the Get-It interface.	
Routine called	axces.apm	
Parameters	**Name** prompt	**Description** In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use close.
	query	In this case, $ax.query.passed.
	boolean1	In this case, the value is the conditional statement nullsub(evstatus in $axces, "close")~#"error".

epmc

Event type	Output	
Description	This event uses the e problem close map to initiate the problem open process associated with the Get-It interface.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use nullsub(evuserseq in $axces, evsysseq in $axces).
	string1	In this case, use ^.
	query	In this case, evuser in $axces.
	name	In this case, use pmc.

epmo

Event type	Input	
Description	This event uses the e problem open map to initiate the problem open process associated with the Get-It interface.	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use open.
	query	In this case, use $ax.query.passed.
	boolean1	In this case, the value is the conditional statement nullsub(evstatus in $axces,"")~#"error".
	cond.input	In this case, the value is the conditional statement $ax,open.flag.

epmo

Event type	Output	
Description	This event uses the e problem open map to write that the problem opened in association with the Get-It interface.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use nullsub(evuserseq in $axces, evsysseq in $axces).
	string1	In this case, use ^.
	query	In this case, evuser in $axces.
	name	In this case, use pmo.

epmosmu

Event type	Input	
Description	This event opens an incident from a call.	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use open.
	query	In this case, use $ax.query.passed.
	boolean1	In this case, the value is the statement nullsub(evstatus in $axces,"")~#"error".
	cond.input	In this case, the value is the statement $ax.open.flag.

epmosmu

Event type	Output	
Description	This event writes the record after an incident opens from a call.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use nullsub(evuserseq in $axces, evsysseq in $axces).
	string1	In this case, use ^.
	query	In this case, evuser in $axces.
	name	In this case, use pmo.

epmu

Event type	Input	
Description	This event uses the e problem open map to initiate the problem update process associated with the Get-It interface.	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use update.
	query	In this case, use $ax.query.passed.
	boolean1	The value is the conditional statement nullsub(evstatus in $axces,"update")~#"error".

epmu

Event type	Output	
Description	This event uses the e problem update map to write that the problem updated in association with the Get-It interface.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	prompt	In this case, use nullsub(evuserseq in $axces, evsysseq in $axces).
	string1	In this case, use ^.
	query	In this case, evuser in $axces.
	name	In this case, use pmu.

esmin

Event type	Input	
Description	This event opens a call in Service Management.	
Routine called	axces.sm	
Parameters	**Name**	**Description**
	record	In this case, use the value of $axces.
	prompt	In this case, evmap in $axces.register.
	string1	In this case, incidents.
	query	In this case, $ax.query.passed.
	boolean1	In this case, true.
	text	in this case, esmin.

esmin

Event type	Output	
Description	This event writes a record once a call opens in Service Management.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	name	In this case, use smout.
	string1	In this case, use ^.
	query	In this case, evuser in $axces.
	prompt	In this case, use nullsub(evuserseq in $axces, evsysseq in $axces).

gie

Event type	Input
Description	Both AssetCenter and ServiceCenter use the Generic Input Event (GIE).
Routine called	None

icma

Event type	Input
Description	This event adds or updates inventory items to the device file if filter criteria are satisfied.
Routine called	scauto.inventory

Parameters	Name	Description
	record	In this case, the value is **$axces**.
	prompt	In this case, use **evmap** in **$axces.register**.
	string1	In this case, use **device**.
	text	In this case, use **add**.
	query	In this case, use **network.name=5** in **$axces.parameters**.
	boolean1	In this case, the value is **true**.
	name	In this case, use **icma**.

icmd

Event type	Input	
Description	This event marks an inventory item for deletion if filter criteria are satisfied by placing **inactive** in the status field.	
Routine called	scauto.inventory	
Parameters	**Name**	**Description**
	record	In this case, the value is **$axces**.
	prompt	In this case, use **evmap** in **$axces.register**.
	string1	In this case, use **device**.
	text	In this case, use **delete**.
	query	To select the device using the network name, use nullsub("network.name=\""+1 in $axces.fields+"\" or logical.name=\""+1 in $axces.fields+"\"", "false".
	boolean1	In this case, the value is **true**.
	name	In this case, use **icmd**.

icmswa

Event type	Input	
Description	This event adds or updates inventory items (that ServerView or StationView discovers) to the device file if filter criteria are satisfied.	
Routine called	axces.pcfiles	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	text	In this case, use add.
	boolean1	In this case, the value is false.
	name	In this case, use pc.files.

icmswd

Event type	Input	
Description	This event marks an inventory item (that ServerView or StationView discovers) for deletion in the pcfiles file if filter criteria are satisfied.	
Routine called	axces.pcfiles	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	text	In this case, use delete.
	boolean1	In this case, the value is false.
	name	In this case, use pc.files.

icmu

Event type	Input
Description	This event updates inventory items if filter criteria are satisfied.

Routine called	scauto.inventory	
Parameter	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, evmap in $axces.register.
	string1	In this case, device.
	text	In this case, update.
	query	In this case, select the device using network name: network.name=5 in $axces.fields.
	boolean1	In this case, the value is true.
	name	In this case, use icmu.

mblcm3tc

Event type	Input	
Description	This provides Peregrine Mobile access to Change Management.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	string1	In this case use cm3t.
	record	In this case, the value is $axces.
	prompt	In this case, evmap in $axces.register.
	text	In this case, 3 in evlist in $axces.
	boolean1	In this case, the value is true.
	description	In this case, '00:10:00', in 10 minutes.

mblcm3tu

Event type	Input
Description	This provides Peregrine Mobile access to Change Management.

Routine called	axces.cm3	
Parameters	**Name**	**Description**
	string1	In this case use cm3t.
	record	In this case, the value is $axces.
	prompt	In this case, evmap in $axces.register.
	text	In this case, 3 in evlist in $axces.
	boolean1	In this case, the value is true.
	description	In this case, '00:10:00', in 10 minutes.

mblocmlc

Event type	Input	
Description	This provides Peregrine Mobile access to Request Management.	
Routine called	axces.cm3	
Parameters	**Name**	**Description**
	record	In this case use $axces.
	text	In this case, the value is 3 in evlist in $axces.
	prompt	In this case, evmap in $axces.register.
	query	In this case, number=1 in evlist in $axces.
	string1	In this case, the value is ocml.

mblocmlu

Event type	Input
Description	This provides Peregrine Mobile access to Request Management.
Routine called	axces.rm

Parameters	Name	Description
	record	In this case use $axces.
	text	In this case, 3 in evlist in $axces
	prompt	In this case, evmap in $axces.register.
	query	In this case, the value is number=1 in evlist in $axces.
	string1	In this case, the value is ocml.

mblpmc

Event type	Input	
Description	This event for Peregrine Mobile closes incidents.	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use close.
	query	In this case, $ax.query.passed.
	boolean1	In this case, nullsub(evstatus in $axces, "close")~#"error".

mblpmo

Event type	Input
Description	This event for Peregrine Mobile opens incidents.
Routine called	axces.apm

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use open.
	query	In this case, $ax.query.passed
	boolean1	In this case, nullsub(evstatus in $axces, "close")~#"error".
	cond.input	In this case, use $ax.open.flag.

mblpmu

Event type	Input
Description	This event for Peregrine Mobile updates incidents.
Routine called	axces.apm

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use update.
	query	In this case, $ax.query.passed
	boolean1	In this case, nullsub(evstatus in $axces, "update")~#"error".

opera

Event type	Input
Description	This event adds or updates a new user to ServiceCenter if filter criteria are satisfied.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use operator.
	string1	In this case, use operator.
	text	In this case, use add.
	query	To select the operator, name=1 in $axces.fields.
	boolean1	In this case, true.
	cond.input	In this case, false.
	name	In this case, operator.scauto.

Important: The default query selects the operator and adds a new user with the minimum privileges to access ServiceCenter: No access to Problem, Change, Inventory, Request or Financial Management.

Note: Most organizations establish a template operator record for each class of users (for example, Incident Management) and modify their select query to the name defined for the template operator record.

For example, you can set up an operator record named standarduser with Execute Capabilities of Incident Management, Inventory Management, Change Request and Change Task, and OCML, OCMQ and OCMO. This allows non-administrative access to Incident, Inventory, Change and Request Management respectively. The query parameter changes from **name=1 in $axces.fields** to **name="standarduser"**.

operd

Event type	Input
Description	This event deletes a user from ServiceCenter if filter criteria are satisfied.
Routine called	axces.database

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use operator.
	string1	In this case, use operator.
	text	In this case, use de;ete.
	query	To select the operator, name=1 in $axces.fields.
	boolean1	In this case, true.
	cond.input	In this case, false.
	condition,1	In this case, true.
	name	In this case, operator.scauto.

operu

Event type	Input	
Description	This event updates items specified in a ServiceCenter file if filter criteria are satisfied.	
Routine called	axces.database	
Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use operator.
	string1	In this case, use operator.
	text	In this case, use update.
	query	To select the operator, name=1 in $axces.fields.
	boolean1	In this case, true.
	cond.input	In this case, true.
	name	In this case, operator.scauto.

outageend

Event type	Input	
Description	This event performs updates to outage records, which are part of the SLA module.	
Routine called	axces.outageend	
Parameters	**Name**	**Description**
	record	This parameter writes the unique record identifier (for example, problem number) to the eventin record. If you use a variable such as $axces, set it up using Format Control to create the intended result.

outagestart

Event type	Input	
Description	This event performs updates to outage records, which are part of the SLA module.	
Routine called	axces.outagestart	
Parameters	**Name**	**Description**
	record	This parameter writes the unique record identifier (for example, problem number) to the eventin record. If you use a variable such as $axces, set it up using Format Control to create the intended result.

page

Event type	Output
Description	This event registration allows ServiceCenter to create eventout records with the evtype=page.
Routine called	axces.write

pageclose

Event type	Input	
Description	This event uses a condition statement (evfiends in $axces)#"pm".	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use envmap in $axces.register.
	string1	In this case, use problem.
	text	In this case, use close.
	query	To select the operator, use "number=\""+substr(1 in $axces.fields, 3, lng(1 in $axces.fields) – 2)+"\"".
	boolean1	In this case, use false.

pageresp

Event type	Input	
Description	This event updates an incident with an acknowledgment or message received as response to a page. It uses a condition statement (evfiends in $axces)#"pm".	
Routine called	axces.apm	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use envmap in $axces.register.
	string1	In this case, use problem.
	text	In this case, use update.
	query	To select the operator, use "number=\""+1 in $axces.fields+"\"".
	boolean1	In this case, use false.

pcsoftware

Event type	Input	
Description	This event allows desktop inventory products to update ServiceCenter.	
Routine called	axces.database	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use pcsoftware.
	string1	In this case, use pcsoftware.
	text	In this case, use add.
	query	In this case, logical.name=20 in $axces.fields and license.number=2 in $axces.fields and application.name=1 in $axces.fields.
	boolean1	In this case, false.
	cond.input	In this case, true.

pmc

Event type	Input
Description	This event closes an incident if filter criteria are met. It uses the same path as manually closing the operation.
Routine called	axces.apm

Initialization expressions	- cleanup($ax.query.passed) - if (not null(3 in $axces.fields)) then ($ax.query.passed="number=\""+str(3 in $axces.fields)+"\"") else ($ax.query.passed="flag=true and network.name=\""+2 in $axces.fields+"\"") - if null($ax.query.passed) then if (not null(20 in $axces.fields)) then ($ax.query.passed="flag=true and reference.no=\""+str(20 in $axces.fields)+"\"") - if null($ax.query.passed) then ($ax.query.passed=nullsub("flag=true and network.name=\""+2 in $axces.fields+"\"", "false")) - if (index("NAPA", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+2 in $axces.fields+"\"", "false")) - if (index("IND", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false")) - $bypass.failed.validation=true - $axces.bypass.failed.validation=true

Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use envmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use close.
	query	In this case, use $axces.query.passed.
	boolean1	In this case, the value is nullsub(evstatus in $axces, "close")~#"error".

pmc

Event type	Output
Description	This event writes after an incident is closed.
Routine called	axces.write

Parameters	Name	Description
	record	In this case, the value is $axces.
	name	In this case, pmc.
	string1	The delimiter character is ^.
	prompt	In this case, nullsub(evusrseq in $axces, evsysseq in $axces).
	query	In this case, evuser in $axces.

pmo

Event type	Input
Description	This event opens an incident if filter criteria are met. It uses the same path as manually opening the incident.
Routine called	axces.apm
Initialization expressions	■ $ax.query.passed=nullsub("flag=true and network.name=\""+2 in $axces.fields+"\"", "false") ■ if (index("axmail", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false")) ■ if (index("NAPA", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false")) ■ $ax.open.flag=false ■ if (index("scnote", evuser in $axces)>0) then ($ax.open.flag=true) ■ $axces.lock.interval='00:00:30' ■ if (index("IND", evuser in $axces)>0) then ($ax.query.passed=nullsub("flag=true and logical.name=\""+1 in $axces.fields+"\"", "false");$ax.open.flag=false) ■ $bypass.failed.validation=true ■ $axces.bypass.failed.validation=true

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use envmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use open.
	query	In this case, use $axces.query.passed.
	boolean1	In this case, the value is nullsub(evstatus in $axces, "")~#"error".
	cond.input	In this case, the value is $ax.open.flag.

pmo

Event type	Output	
Description	This event writes after an incident is opened.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	name	In this case, pmo.
	string1	The delimiter character is ^.
	query	In this case, evuser in $axces.
	prompt	In this case, nullsub(enusrseq in $axces, evsysseq in $axces).

pmu

Event type	Input
Description	This event updates an incident if filter criteria are met. It uses the same path as manually updating the incident.
Routine called	axces.apm

Initialization expressions

- cleanup($ax.query.passed)
- if (not null(3 in $axces.fields)) then ($ax.query.passed="number=\""+str(3 in $axces.fields)+"\"") else ($ax.query.passed="flag=true and network.name=\""+2 in $axces.fields+"\"")
- if null($ax.query.passed) then if (not null(20 in $axces.fields)) then ($ax.query.passed="flag=true and reference.no=\""+str(20 in $axces.fields)+"\"")
- if null($ax.query.passed) then ($ax.query.passed="false")
- $bypass.failed.validation=true
- $axces.bypass.failed.validation=true

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use envmap in $axces.register.
	string1	In this case, use probsummary.
	text	In this case, use update.
	query	In this case, use $axces.query.passed.
	boolean1	In this case, the value is nullsub(evstatus in $axces, "update")~#"error".

pmu

Event type	Output
Description	This event writes after an incident is updated.
Routine called	axces.write

Parameters	Name	Description
	record	In this case, the value is $axces.
	name	In this case, pmu.
	string1	The delimiter character is ^.
	query	In this case, evuser in $axces.
	prompt	In this case, nullsub(evusrseq in $axces, evsysseq in $axces).

prgma

Event type	Input	
Description	This adds or updates a software inventory item to the pcfiles file that an external agent (other than ServerView or StationView) discovers if filter criteria are satisfied.	
Routine called	axces.software	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, use software.
	string1	In this case, use pcfiles.
	query	In this case, logical.name=1 in $axces.fields and description=9 in $axces.fields.
	name	In this case, pc.files.
	boolean1	In this case, false.

prgmd

Event type	Input
Description	This deletes a software inventory item from the pcfiles file that an external agent (other than ServerView or StationView) discovers if filter criteria are satisfied. **Note:** The default updates the estatus field as deleted rather than removing the record from the database.
Routine called	axces.software

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use pcfiles.
	string1	In this case, use software.
	query	In this case, logical.name=1 in $axces.fields and description=9 in $axces.fields.
	name	In this case, pc.files.
	boolean1	In this case, false.
	cond.input	In this case, true.
	condition,1	In this case, false.

prgmu

Event type	Input
Description	This updates an inventory item in the pcfiles file that an external agent (other than ServerView or StationView) discovers if filter criteria are satisfied.
Routine called	axces.software

Parameters	Name	Description
	record	In this case, the value is $axces.
	prompt	In this case, use pcfiles.
	string1	In this case, use software.
	query	In this case, logical.name=1 in $axces.fields and description=9 in $axces.fields.
	name	In this case, pc.files.
	boolean1	In this case, false.
	cond.input	In this case, false.

rmlin

Event type	Input	
Description	This provides access to Request Management line items.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocml.
	text	In this case, use 3 in evlist in $axces.
	query	In this case, use number=1 in evlist in $axces.

rmoappr

Event type	Input	
Description	This provides access to Request Management order approval.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocmo.
	text	In this case, use 3 in evlist in $axces.
	query	In this case, use number=1 in evlist in $axces.

rmoin

Event type	Input	
Description	This provides access to Request Management order input.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocmo.
	text	In this case, use 3 in evlist in $axces.
	query	In this case, use number=1 in evlist in $axces.

rmqappr

Event type	Input	
Description	This provides access to Request Management quote approval.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocql.
	text	In this case, use 3 in evlist in $axces.
	query	In this case, use $L.approve.action.

rmqin

Event type	Input
Description	This provides access to Request Management quote input.

Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocmq.
	text	In this case, use 3 in evlist in $axces.
	query	In this case, use number=1 in evlist in $axces.

sapordl *(1)*

Event type	Output
Description	This event routes order information to SAP for processing.
Routine called	None

sapordl *(2)*

Event type	Input	
Description	This event routes order information to SAP for processing.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocml.
	text	In this case, use update.
	query	In this case, use number=12 in evlist in $axces.
	name	In this case, use falcon.

sapqtel *(1)*

Event type	Output
Description	This event is the Output quote line item component of the SAPQTE event. It uses this registration to identify which eventmap to use for message formatting.
Routine called	None

sapqtel *(2)*

Event type	Input	
Description	This event is the detail portion of the SAPQTE event.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocml.
	text	In this case, use update.
	query	In this case, use number=12 in evlist in $axces.
	name	In this case, use falcon.

saprecl *(1)*

Event type	Output
Description	This event is the Output goods receipt line item component of the SAPQTE event. It uses this registration to identify which eventmap to use for message formatting.
Routine called	None

sapreql *(1)*

Event type	Output
Description	This event is the Output request line item component of SAPREQ. It uses this registration to identify which eventmap to use for message formatting.
Routine called	None

sapreql *(2)*

Event type	Input	
Description	This event is the detail portion of the SAPREQ event.	
Routine called	axces.rm	
Parameters	**Name**	**Description**
	record	In this case, it passes the value of $axces.
	prompt	In this case, use evmap in $axces.register.
	string1	In this case, use ocml.
	text	In this case, use update.
	query	In this case, use number=20 in evlist in $axces.
	name	In this case, use falcon.

slaresponse

Event type	Output	
Description	This is a request from an external application to enter a response time metric against a device with an SLA.	
Routine called	axces.postresponse	
Parameters	**Name**	**Description**
	record	This parameter writes the unique record identifier (for example, problem number) to the eventin record. If you use a variable such as $axces, set it up using Format Control to create the intended result.

smin

Event type	Input	
Description	This event accesses Service Management incoming service requests or help issues.	
Routine called	axces.sm	
Initialization expressions	■ $ax.query.passed=nullsub("incident.id=\""+1 in $axces.fields+"\"", "false") ■ if (null(1 in $axces.fields) or 1 in $axces.fields="") then ($ax.query.passed="false")	
Parameters	**Name**	**Description**
	record	In this case, use the value of $axces.
	prompt	In this case, evmap in $axces.register.
	string1	In this case, incidents.
	query	In this case, $ax.query.passed.
	boolean1	In this case, true.

smout

Event type	Output	
Description	This event writes once an incoming service request or help issue enters the system.	
Routine called	axces.write	
Parameters	**Name**	**Description**
	record	In this case, the value is $axces.
	prompt	In this case, nullsub(evusrseq in $axces, evsysseq in $axces).
	name	In this case, smout.
	string1	The delimiter character is ^.
	query	In this case, evuser in $axces.

submit

Event type	Output
Description	This event submits a job for processing.
Routine called	axces.write

sysbull

Event type	Input
Description	This event adds a new System Bulletin to ServiceCenter if filter criteria are satisfied.
Routine called	axces.database

Parameters	Name	Description
	prompt	In this case, use bulletin.
	string1	In this case, use bulletin.
	text	In this case, use add.
	query	In this case, use date=date(val(str(1 in $axces.fields),3)).
	boolean1	In this case, the value is true.
	cond.input	In this case, the value is false.

Note: The system bulletin record is for **today's date.** For example, if today is New Year's Day, the bulletin is for 01/01/04 00:00, or the one with the default flag set to true.

Warning: Do not modify the application names or parameters unless you are completely familiar with RAD programming.

Event Services provides a standard interface for user-defined applications as well as those described in this section. You can call any RAD application that does not require user I/O as an event services application.

Index

June 18, 2004