HP SCAuto SDK

for the UNIX® and Windows® operating systems

Software Version: 2.10

User's Guide

Document Release Date: September 2008

Software Release Date: September 2008

Legal Notices

Warranty

Restricted Rights Legend

Copyright Notices

Trademark Notices

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support web site at:

www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

1 Introduction

Welcome to HP ServiceCenter Automate (SCAuto) Software Development Kit (SDK). This product is part of the suite of SCAuto interface products that integrates HP Service Manager with premier network and systems management tools. SCAuto SDK is used in the development of custom interfaces to Service Manager. It provides a simple application program interface (API) to interact with current Service Manager Rational Application Developer (RAD) applications.

➤ Although this guide refers exclusively to HP Service Manager, it applies to HP ServiceCenter as well. If you are using ServiceCenter, just replace the term “Service Manager” with “ServiceCenter.”

Purpose

This guide describes how to implement SCAuto SDK for integration with Service Manager. For more information about SCAuto, see the *HP SCAuto Applications for Windows and UNIX User’s Guide*.

Audience

This guide is intended for system administrators who install and implement Service Manager.

Prerequisites

This guide assumes you have knowledge of the following:

- **Service Manager**

  Working knowledge of Service Manager applications, the Service Manager client-server environment, and the C programming language. For details, see the appropriate Service Manager documentation.

- **GUI environment**

  Working knowledge of a GUI or text-based environment.

- **Operating system**

  A thorough knowledge of the operating system where the product will be installed and implemented, as well as a basic understanding of Service Manager applications and Event Services.

➤ Service Manager installation requirements are specific to the machine where Service Manager is installed. For a list of requirements, see the respective installation guides.

2 SCAuto overview

HP ServiceCenter Automate (SCAuto) provides Event Management services through a collection of automation products. These automation products enable you to integrate external applications with HP Service Manager. Generic Event Management services enable you to create meaningful events from their environment and applications. For example, you can create tickets, send out notifications after tickets are closed, manage network inventory automatically, and manage email between disparate external email programs. The intent is to enable communication from any external application with any Service Manager RAD application, without predetermining what that communication requires.

Listener

A dedicated Java-based listener (`scautolistener`) manages all connections between SCAuto and Service Manager. You can start `scautolistener` from the command line or by using an entry in the `sm.cfg` file. For details, see the Service Manager online help.

For example, to start a listener on port 12690, you could use the following command:

```
sm -scautolistener:12690
```

After the listener starts, it accepts connections from SCAuto clients.

➤ While running, the listener appears with the user name `SCAutoListener-12690` from within Service Manager. You should stop it from there, if needed. You do not terminate existing clients when you stop `scautolistener`.

Internal Event Services

SCAuto products run on various supported platforms. Each product uses TCP/IP socket architecture to connect to a Listener that runs on the Service Manager server platform. After a connection is established with the Listener, a process on the server is created. The new process runs in a manner similar to that of an `sm` process in Service Manager. The new process reads and writes event records directly to the event files.

Service Manager events are written and read internally, using a set of RAD applications known as Event Services. Event Services provides functions such as event mapping and application scheduling within Service Manager. These services enable RAD applications to create events for external processing. They also enable external programs to schedule RAD applications asynchronously.

You can write or modify any RAD application to use Event Services to interact with external SCAuto applications.

External Event Services

Applications external to the Service Manager environment are provided with multiple options for interfacing with RAD applications.

File Monitor interface

The simplest option is a File Monitor interface, which is comprised of external event schedulers and event files. Using this option, Service Manager input events are written by the applications to an input event file. That file is read by an SCAuto external scheduler, and then forwarded to the Service Manager Event Manager. Service Manager output events are read from the Service Manager Event Manager by an SCAuto file monitor, and then placed in an application output events file. Each instance of an external scheduler is started for a different file, enabling you to batch events, create more parallelism, or have separation of external applications. Each file monitor can be started to process a different event or event list. This interface mode permits external programs to use standard file read-write privileges with a predefined record structure to communicate with Service Manager.

C function API

A second option is an application programming interface (API) and a run-time library that provide connection services and event services. Any program that can call a C function, and that can be called from a C environment, can utilize this API. Under this option, applications can connect to Service Manager with an `scauto_connect` call. You can then specify which events and Service Manager server your application requires.

After connecting to Service Manager successfully, you can create, retrieve, and delete Service Manager events by issuing `scauto_create`, `scauto_query`, and `scauto_delete` calls. Before termination, your application issues an `scauto_disconnect` call. Applications can have multiple sessions with one or more Service Manager servers.

SDK

The SCAuto Software Development Kit (SDK) is used in the development of custom interfaces to Service Manager. It provides a simple API to interact with current Service Manager RAD applications.

Platforms

The SCAuto SDK requires SCAuto Listener to be on the Service Manager server platform. Listener and an SDK application provide Event Management services, enabling external applications to be integrated with Service Manager. These generic services enable users to create meaningful events from their environment and legacy applications.

➤ To find the process for starting the listener for SCAuto, see the relevant Service Manager documentation.

SCAuto SDK is available for Microsoft Windows and most UNIX platforms. The API is packaged as a C library on UNIX and as DLLs on Windows. The Windows versions can also be used with Microsoft Visual Basic, Borland Delphi, or other products that can use DLL routines.

Protocols

SCAuto clients are not Service Manager clients. Rather, they use their own simpler network protocol to communicate with an Listener. The resulting applications are smaller. They do not need to be upgraded or recompiled to connect to newer Service Manager server versions.

The TCP/IP network protocol is used to connect SCAuto SDK to Listener. TCP/IP must be installed and configured correctly on the systems where SCAuto applications are to be run.

Components

The SCAuto SDK includes the following optional component files:

- **Libraries**

  The link libraries and run-time libraries are included for some systems.

- **Header file**

  The `scauto.h` C header file is included for applications written in C.

- **Test program**

  A test program ensures the proper installation and communication between SCAuto and Listener.

- **Source code**

  The C source code is included for a sample SCAuto application.

- **File Monitor program**

  The SCAuto File Monitor program reads and routes system events. It can be run in several instances simultaneously.

- **Submit Monitor program**

  The SCAuto Submit Monitor program responds to Service Manager requests to run external programs.

Architecture

The general architecture is composed of an SCAuto SDK application and the supplied API. SCAuto SDK connects to Listener, which in turn connects to Service Manager. The SCAuto SDK application can then create events that are processed by an accompanying RAD application. The RAD application can also create events that can be retrieved and processed by the external SCAuto SDK application.

This event-driven technique allows asynchronous running of the external applications. This standard interface is currently being employed by HP SCAuto products.

Event Manager

Service Manager includes an Event Manager application. This application consists of an input event scheduler, as well as a set of standard tables for registration, mapping, and event handling. The input event scheduler polls the input events tables. It then routes new events to RAD applications by using registration and mapping tables. For details, see the *ServiceCenter 6.1 Event Services Guide*.

The output event scheduler is called by any RAD application to create an event, and schedule it for external processing.

➤ In many cases, using these generic schedulers enables SCAuto implementation without affecting current production RAD applications. It also assists in new application development with a standard ready-made solution to many external interfacing requirements.

3 Installing SCAuto SDK

You can install SCAuto SDK on a Windows or Unix server. This section contains information about installation requirements and how to install the SCAuto SDK.

This chapter provides the instructions for installing the SCAuto SDK in the Windows and Unix environments.

Topics in this chapter include:

Windows Installation

The bulk of the installation is done automatically.

Installing SCAuto SDK on Windows systems

Follow these steps to install the SCAuto SDK:

1 Log in to the Windows server as a user with local administrator privileges.

2 Insert the SCAuto installation media into the appropriate drive of the server.

  If you are installing on a system that has auto-run enabled, the installation media browser starts automatically. If auto-run is disabled, follow these steps to start the installation media browser manually.

  a Navigate to the installation media folder.

  b Double-click `setupwin32.exe`.

  The HP SCAuto SDK InstallShield wizard opens.

3 Click **Next** to read and accept the licensing agreement.

4 Select the **I accept the terms in the License Agreement** option.

  The **Next** button becomes active.

5 Do one of the following:

  a Click **Next** to accept the default installation folder.

    The default installation folder is:
    `C:\Program Files\HP\HP_SCAuto_SDK_2.10`

  b Click **Browse** to choose a different installation location.

  You must install the SCAuto SDK in a folder containing only ASCII characters in the folder name. The SDK cannot start if installed in a folder with non-ASCII characters in the folder name.

  The installation process automatically adds a subfolder for the SCAuto SDK installed:

- `\bin` contains executable programs
- `\lib` contains the SCAuto SDK library and header files

- `\samples` contain sample source code

6 Click **Install** to begin copying the installation files.

  You can stop the installation by clicking **Cancel**.

  A summary page opens when the installation is complete.

7 Click **Finish** to exit the wizard.

Uninstalling SCAuto Applications from Windows systems

Follow these steps to uninstall SCAuto SDK:

1 From the Windows main menu, click **Start** > **Settings** > **Control Panel** > **Add/ Remove Programs**.

  The Add/Remove Programs dialog box opens.

2 Select HP SCAuto SDK and click **Change/Remove**.

  The HP SCAuto SDK InstallShield wizard opens.

3 Click **Next**.

4 Review the uninstallation information.

5 Click **Next**.

6 Click **Uninstall** to remove the indicated features.

  You can stop the installation by clicking **Cancel**.

  A summary page opens when the installation is complete.

7 Click **Finish** to exit the Setup wizard.

Unix Installation

The bulk of the installation is done automatically for you. .

➤ Case sensitivity may be dependent on the Unix system that you are using.

Installing SCAuto SDK on Unix systems

Follow these steps to install the SCAuto applications:

1. Log on to the server.
2. Insert the SCAuto installation media into the appropriate drive of the server.
3. Mount the installation media.
4. CD to the mount location.
5. Run the appropriate setup script. For example, you could run:

   ```
   <Path to Installation files>: setupHP11-parisc
   ```

   Alternatively, you can run the script in console mode by using the following command:

   ```
   setupHP11-parisc -console
   ```

   The HP SCAuto SDK InstallShield wizard starts.
6. Click **Next** to read the licensing agreement.

   Console mode users chose 1 for Next, and press Enter to read and accept the licensing agreement.
7. Select the **I accept the terms in the License Agreement** option.

   Click **Next** to accept the licensing agreement.

   Console mode users press 1 to accept the terms of the license agreement. and press 0 to proceed.

   The HP SCAuto SDK Install Location screen opens showing the Directory Name.
8. Do one of the following:
   - Click **Next** to accept the default installation directory.

- Click **Browse** to choose a different installation location.

‼ You must install the SCAuto SDK in a directory containing only ASCII characters in the directory name. The SDK cannot start if installed in a directory with non-ASCII characters in the directory name.

The installation process automatically adds a subdirectory for each SCAuto application installed:

- `/bin` contains executable programs
- `/lib` contains SCAuto SDK libraries and header files
- `/samples` contains sample source code

Console users, specify a directory or press Enter to accept the default directory, and then press 1 to proceed to the next screen.

The installation summary screen opens.

9. Click **Install** to start installing.

   You can stop the installation by clicking **Cancel**.

   Console users, press 1 to start installing.

   A summary page opens when the installation is complete.

10. Click **Finish** to exit the Setup wizard.

    Console users, press 3 to exit the wizard.

Uninstalling SCAuto SDK from Unix systems

Follow these steps to uninstall the SCAuto SDK:

1. CD to `<install_directory>/_uninst`
2. Run the uninstall program `./uninstall`.

   The HP SCAuto SDK InstallShield wizard opens.
3. Click **Next**.
4. Review the uninstallation information.
5. Click **Next**.
6. Click **Uninstall** to remove the indicated features.

You can stop the installation by clicking **Cancel**.

A summary page opens when the installation is complete.

7 Click **Finish** to exit the Setup wizard.

4 Utility programs

HP ServiceCenter Automatic (SCAuto) Software Development Kit (SDK) has two utility programs:

- File Monitor
- Submit Monitor

File Monitor

The SCAuto File Monitor utility communicates with HP Service Manager through normal text files. External applications can create Service Manager input events by appending events to a file. File Monitor reads the events, and then forwards them to Service Manager Event Manager.

File Monitor reads Service Manager output events from Service Manager Event Manager, and places them in a file for external applications to read. You can start each instance of File Monitor on a different file. Doing so enables you to batch events, increase parallelism, and separate external applications. You can also start each instance of File Monitor to process a different list of events.

External applications use standard read-write operations to access File Monitor files, using a predefined record structure to communicate with Service Manager.

To operate File Monitor, you need a general understanding of Service Manager Event Services. For details, see the *ServiceCenter 6.1 Event Services Guide*.

Components

The SCAuto File Monitor utility includes the following components:

`scfiled`

File Monitor executable. On Windows, the file is called `scfiled.exe`.

`scauto.msg`

Optional component. Messages printed by the File Monitor and other SCAuto tools. You can customize this component for use with languages other than English.

`scauto.dll`

Windows-only component. SCAuto run-time library.

Command-line syntax

You can start the SCAuto File Monitor utility with the following syntax:

```
scfiled -input infile -output outfile -evlist eventlist
-server scauto server -id execution id [...]
```

The following parameters are allowed:

`-input` <*infile*>

Input events file. Events in this file are written to Service Manager. You must specify at least one `-input` or `-output` parameter.

`-output` <*outfile*>

Output events file. Events from Service Manager are written to this file. You must specify at least one `-input` or `-output` parameter.

`-evlist` <*event list*>

List of event types to be retrieved from Service Manager (only if the `-output` parameter is provided).

You can specify the event list as follows:

`"type"`

Retrieves a single event type.

`"(type1,type2,...)"`

Retrieves a list of event types.

`"all"`

Default. Retrieves all event types.

`-user` <*user name list*>

List of events that match the user name list.

You can specify the user name list as follows:

`"type"`

Retrieves a single user name type.

`"(type1,type2,...)"`

Retrieves a list of user name types.

`"all"`

Default. Retrieves all user name types.

`-server <`*`scauto server`*`>`

Service name of the SCAuto server to which you want to connect. The default is `scauto`.

`-id <`*`execution id`*`>`

Execution ID for this Service Manager connection. This ID is used from within `status` in Service Manager to identify processes. This parameter is required. It has no default.

`-log <`*`log file`*`>`

Name of the file to which you want to log messages. The default is `scfiled.log`.

To log to the screen, use one of the following:

- *UNIX*

  `/dev/tty`

- *Windows*

  `con`

`-checkpoint <`*`file`*`>`

Name of the file to store run-time checkpoints. This parameter enables File Monitor to continue where it left off when it was restarted. The default is `scfiled.chk`.

`-nocheckpoint`

Prevents a checkpoint file from being used.

`-noloop`

Terminates File Monitor when all pending events have been processed. Normally, File Monitor sleeps and polls for new events.

`-sleep <`*`secs`*`>`

Number of seconds File Monitor sleeps before polling for new events. The default `10`.

`-delete`

Causes Service Manager output events to be deleted when successfully written to the output file. Normally, the events are retained, and the checkpoint file is used to resynchronize when the File Monitor is restarted.

`-debug`

Switches on additional log messages.

`-msgfile`

Name of the message file to use. The default is `scauto.msg`.

Event Records

SCAuto Event Records are comprised of comma-delimited header data, followed by the event fields, as shown in Table 1. For descriptions of these fields, see Event structure on page 44.

Table 1 SCAuto Event Records

Field Name	Max Length	Type
`evtype`	16	character
`evtime`	19	date
`evsysseq`	32	reserved
`evusrseq`	32	character
`evsysopt`	24	character
`evuser`	24	character
`evpswd`	24	character
`evsepchar`	1	character
`evfields`	32767	character

You can separate the header data and the event fields by a comma or a colon:

- **Comma**

```
evtype,evtime,evsysseq,evuserseq,evsysopt,evuser,evpswd,evsep
char,evfields
```

- **Colon**

```
evtype,evtime,evsysseq,evuserseq,evsysopt,evuser,evpswd,evsep
char:evfields
```

Fields that are too long are truncated. However, exceeding the maximum length of `evfields` can be inefficient.

Example:

```
pmo,06/11/1997 22:06:58,,866065020,,EventUser,,
^:logname^netname^866065020^^Testing Problem
Open^^^^^^^^^^^^^^^^^^
```

➤ You do not need to fill in all fields. Fields that are not filled in are indicated by double commas (`,,`). In the example, the empty `evsysseq`, `evsysopt`, and `evpswd` fields are indicated by double commas.

Submit Monitor

The SCAuto Submit Monitor utility enables Service Manager to execute external commands through the Event Manager. The commands are executed on the same machine as that of the Submit Monitor. This machine may be a different machine from that of the Service Manager. You can use this utility to automate and customize Service Manager applications.

The Submit Monitor uses `submit` events. You specify the command and arguments to execute, as well as the name of the host machine on which the command should execute. For the command to execute, the local host name must match that specified in the event.

Components

The SCAuto Submit Monitor utility includes the following components:

`scsubmit`

Submit Monitor executable. On Windows, the file is called `scsubmit.exe`.

`scauto.msg`

Optional component. Messages printed by the Submit Monitor and other SCAuto tools. You can customize this component for use with languages other than English.

`scauto.dll`

Windows-only component. SCAuto run-time library.

Command-line syntax

You can start the SCAuto Submit Monitor utility with the following syntax:

`scsubmit -server scauto server [...]`

The following parameters are allowed:

`-server <`*`scauto server`*`>`

Service name of the SCAuto server to which you want to connect. The default service name is `scauto`.

`-user <`*`user name list`*`>`

Only retrieve events that match user name.

You can specify the user name list as follows:

`"type"`

Retrieves a single user name type.

`"(type1,type2,...)"`

Retrieves a list of user name types.

`"all"`

Default. Retrieves all user name types.

`-log <`*`log file`*`>`

Name of file to which you want to log messages. The default is `scsubmit.log`.

`-sleep <`*`secs`*`>`

Number of seconds the Submit Monitor sleeps before polling for new events. The default is 10 seconds.

`-debug`

Switches on additional log messages.

`-msgfile`

Name of the message file to use. The default is `scauto.msg`.

Submitting events

With the SCAuto Submt Monitor utility, you can submit events with the following syntax:

`hostname^command^arg1^arg2^...`

The following parameter, command, and arguments are allowed:

`<Hostname>`

Name of the machine on which the command should be executed.

`<command><args>`

Command and arguments are joined together, separated by spaces, to compose the full command line to execute.

➤ There is no special formatting involved. Also, there is no requirement that you place each argument in a separate field.

No error checking is performed on the executed commands. Error checking is performed by the command itself.

If the command does not redirect its output, Submit Monitor tries to append the output of the command to its own log file:

- **UNIX**

  The executed command is placed in the background. An ampersand (`&`) is *not* added to the command.

- **Windows**

  The processing of `submit` events pauses until the executed command finishes. For this reason, you should write the command to execute quickly.

The executed command inherit the execution environment of Submit Monitor. Make sure that Submit Monitor has access to the commands you intend to execute from Service Manager, as well as the capabilities to execute them.

5 Developing applications

This chapter describes how to develop an HP ServiceCenter Automatic (SCAuto) Software Development Kit (SDK) application:

- Sample application
- Building applications

Sample application

This section includes a sample SCAuto SDK application with annotations. This sample is very similar to the sample application code that is included with SCAuto SDK.

Opening the session

When run, the application expects an `-s server` option on the command line:

```
/*
 * SCAuto SDK API sample program
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <errno.h>
```

Defining prototypes

The application must include `scauto.h`, which defines prototypes for all of the SCAuto functions, as well as for event structure and error codes:

```
#include “scauto.h”
```

The session identifier used throughout this program is defined as follows:

```
#define SESSID “SCAuto_test”

char server[256];/* SCAuto Listener to connect to */

void cleanup(int);
void print_event(char *header, struct scauto_event *event);
void parms_init(int argc, char *argv[]);
void show_error(int code);

int
main (int argc, char *argv[])
{
```

Passing the event structure to and from calls

The `event` variable passes an event structure to and from SCAuto calls:

```
struct scauto_event *event, the_event;
int status;

  event = &the_event;

  parms_init(argc, argv);/* parse command line */

#ifdef UNIX
  signal(SIGINT, cleanup);
  signal(SIGTERM, cleanup);
  signal(SIGQUIT, cleanup);
#endif
```

The `status` variable stores the return value of SCAuto calls.

Connecting to the remote SCAuto Listener

The application connects to the remote SCAuto Listener by using the scauto_connect function:

```
/* Connect to the remote SCAuto Listener */
  printf("<<<<<< scauto_connect to SCAuto Listener >>>>>>\n");
  status = scauto_connect(SESSID, server, "all", "all");
  if (status != 0)
    {
      printf("%s: scauto_connect() failure.\n", argv[0]);
      show_error(status);
      exit(4);
    }
  printf("scauto_connect() ok.\n\n");
```

The first two parameters specify the session identifier and server to which to connect. The last two all parameters indicate that the application lists events with any event type and from any user. The application also checks the return value from scauto_connect for errors. If it finds errors, it displays a diagnostic message.

Creating an event

To create an event, you must fill in an event structure:

```
/* Create an event */
  printf("<<<<<<< Create events processing >>>>>>\n");\

  strcpy(event->evtype, "email");
  strcpy(event->evtime, "");/* leave blank and SCAuto Listener will fill it in */
  strcpy(event->evsysseq, "");/* this will be ignored */
  strcpy(event->evusrseq, "12345678901234");
  strcpy(event->evsysopt, "");
  strcpy(event->evuser, "Testapi");
  strcpy(event->evpswd, "");
  event->evsepchar = '^';
  event->evfields = "falcon^SCAuto/sdk^^Test SCAuto^Hey, this worked!";

  /* utility routine to print out an event */
  print_event("Create event", event);
```

Checking the return status for errors

The application passes the event to scauto_create, and then checks the return status for errors:

```
status = scauto_create(SESSID, event);
if (status != 0)
  {
    printf("%s: scauto_create_event() failure.\n", argv[0]);
    show_error(status);
    scauto_disconnect(SESSID);/* be sure to clean up! */
    exit(4);
  }w
printf("scauto_create() ok.\n\n");
```

Because an SCAuto connection is open, the application calls scauto_disconnect before exiting.

Retrieving events with wait processing

To retrieve an event, the application calls one of the following:

- scauto_query

  Retrives the first event in the eventout file. The event argument contains the event, if there is one.

- scauto_query_ex

  Enables you to specify a query string in P4 format and look for event that match the szQuery string.

  Example:

  szQuery = "evusrseq=\"12345678901234\""

  To get the matched event, you need to call scauto_get_next.

Retrieving the first event

To retrieve the first event, you use `scauto_query` and `scauto_get_next`:

```
  printf("<<<<<<< Retrieve events with wait processing >>>>>>\n");
  status =  scauto_query(SESSID, "0000000000000000", "wait", event);
  if (status != 0)
    {
      if (status != NOEVFND)
        {
          printf("%s: scauto_query() failure.\n", argv[0]);
          show_error(status);
          scauto_disconnect(SESSID);
          exit(4);
        }
      else
        {
          printf("%s: No events found, exiting.\n", argv[0]);
          scauto_disconnect(SESSID);
          exit(4);
        }
    }
  print_event("Retrieved event ok", event);
```

In this instance, the `0000000000000000` parameter is the starting `evsysseq` value on which to query. This value causes `scauto_query` to retrieve the first event in the `eventout` file. The application tests for `NOEVFND` explicitly because this value does not indicate an error. Normally, applications do not need to check for `NOEVFND` after calling `scauto_query` with the `wait` option.

Retrieving the next event

To retrieve the second event, you use `scauto_get_next`:

```
printf("\n<<<<<<< Get next event processing >>>>>>\n");
  status = scauto_get_next(SESSID, event);
  if (status == 0)
    {
      print_event("Retrieved event ok", event);
    }
  else
    {
      if(status != NOEVFND)
        {
          printf("%s: scauto_get_next() failure.\n", argv[0]);
          show_error(status);
          scauto_disconnect(SESSID);
          exit(4);
        }
      else
        printf("No more events found.\n");
    }
}
```

You check for `NOEVFND` explicitly because it is a very common return code for `scauto_get_next`. If you want the application to continue and retrieve all events, you can put `scauto_get_next` into a loop. The application then checks the return status to determine when there are no more events to retrieve.

Finding a matching event

To find a matching event, you use `scauto_query_ex` and `scauto_get_next`:

```
// Query for the Event using the new Query API. Here we're doing a query
// for event that has a matching evusrseq number of 12345678901234

  status = scauto_query_ex(SESSID, "evusrseq=\"12345678901234\"","wait");

  if (status != 0)
  {
      if (status != NOEVFND)
  {
  printf("%s: scauto_query_ex() failure.\n", argv[0]);
  show_error(status);
  scauto_disconnect(SESSID);
  exit(4);
  }
      else
  {
  printf("%s: No events found, exiting.\n", argv[0]);
  scauto_disconnect(SESSID);
  exit(4);
  }
  }
```

In this instance, the question mark (`?`) query string is the following:

`evusrseq="12345678901234"`

If a match event is found, you call `scauto_get_next` to retrieve the event.

Retrieving a matching event

To retrieve a matching event from the `eventout` file, you call `scauto_get_next`:

```
// This gets the event out of the ServiceCenter EVENTOUT queue

  printf("\n<<<<<<< Get next event processing >>>>>>\n");
  status = scauto_get_next(SESSID, event);
  if (status == 0)
    {
      print_event("Retrieved event ok", event);
    }
  else
    {
      if(status != NOEVFND)
{
  printf("%s: scauto_get_next() failure.\n", argv[0]);
  show_error(status);
  scauto_disconnect(SESSID);
  exit(4);
}
      else
        printf("No more events found.\n");
    }

  print_event("Scauto_query_ex Retrieved event ok", event);
```

Closing the session

When the application is ready to exit, or when it is finished with the SCAuto session, you disconnect the session:

```
  /* Disconnect from SCAuto Listener */
  printf("\n<<<<<<< Disconnect SCAuto >>>>>>\n");
  status = scauto_disconnect(SESSID);
  if (status != 0)
    {
      printf("%s: scauto_disconnect failed.\n", argv[0]);
      show_error(status);
    }

  printf("\ntests completed ok\n\n");
  return (0);
}
```

Explicitly disconnecting enables the remote Listener to close the session gracefully and free up any used resources.

Printing an event

To print an event, you run the following utility routine:

```
void print_event(char *header, struct scauto_event *event)
{
  printf("%s: evtype=%s, evtime=%s, evsysseq=%s, evusrseq=%s, ", header,
         event->evtype, event->evtime, event->evsysseq, event->evusrseq);
  printf("evuser=%s, evpswd=%s, evsepchar=%c, ",
         event->evuser, event->evpswd, event->evsepchar);
  printf("evfields=%s\n", event->evfields);
}
```

Defining the server

To parse the command line for the `-s server` parameters, you use the following two routines:

```
void
usage(char *name)
{
  printf("usage: %s -s server\n", name);
  exit(0);
}
```

```
void
parms_init(int argc, char *argv[])
{
  /* Getopt would make this cleaner, but it’s not included  on many systems */
  int i;

  strcpy( server,””);

  for (i = 1; i < argc; i++)
    {
      if (strcmp(argv[i], “-s”) == 0 && argv[i+1])
        {
          i++;
          strcpy(server, argv[i]);
        }
      else
          usage(argv[0]);
    }

  /* verify that a server argument was given */
  if(!server[0])
    {
      printf(“no server specified.\n”);
      usage(argv[0]);
    }
}
```

Disconnecting from the server

When the application receives a signal (as specified during initialization), it calls the following routine:

```
void
cleanup(int ignored)
{
  scauto_disconnect(SESSID);
  printf(“terminated.\n”);
  exit(1);
}
```

This routine ensures that `scauto_disconnect` is called.

Printing a diagnostic

To print out an appropriate diagnostic when passed an SCAuto error code, you call the following routine:

```
void show_error(int code)
{
  if (errno)
    perror(scauto_error_string(code));
  else
    fprintf(stderr, “%s\n”, scauto_error_string(code));
}
```

The function `scauto_error` does most of the work.

Building applications

To build an SCAuto application, you must link it with the appropriate SCAuto library, as well as with any operating system–dependent libraries. How you link the application to these libraries depends to a large extent on the operating system and compiler being used.

This section describes sample makefiles for some systems. The sample makefiles that come with SCAuto SDK may be more detailed.

Windows applications

The SCAuto library includes two components for import libraries:

- `scauto.lib`

  Import library for Microsoft Visual C++ only.

- `scauto.dll`

  DLL file for other compilers. You use this DLL file to build an import library at run time.

  For example, you can create an `scauto.lib` that works with Borland C with the command:

  `implib scauto.lib scauto.dll`

scauto.lib

On Windows, the makefile that builds the SCAuto application might look like this:

```
# This is a makefile for building the sample SCAuto application.
!include <win32.mak>

CFLAGS = $(cflags) $(cvarsmt) $(cdebug) -I..\lib
LFLAGS = $(conlflags) $(ldebug)
LIBS = $(conlibsmt) ..\lib\scauto.lib wsock32.lib

sdktest.exe: sdktest.obj
$(link) $(LFLAGS) /OUT:sdktest.exe sdktest.obj $(LIBS)
```

scauto.dll

To run properly, SCAuto applications for Windows must access the SCAuto DLL files. These DLL files are in the `LIB` subdirectory, labeled `SCAUTO.DLL`.

When an SCAuto application runs, it searches for these DLL files in the following places:

- Same directory as the application executable file
- Windows system directory:

  `\Windows\system`

  `\windows\system32`
- All directories specified in the `PATH` environment variable

The same DLL files can be used by all SCAuto products. If you have previously installed an SCAuto product, these DLL files may already be set up correctly.

If you plan to use a single SCAuto application, simply copy these DLL files to the directory that contains your SCAuto application executable.

If you plan to develop multiple SCAuto applications, do one of the following:

- Add the `SCAUTO\LIB` directory to your `PATH` environment variable.
- Copy the DLL files to a directory names in the `PATH` variable.
- Copy the DLL files to the Windows system directory.

UNIX applications

By default, many UNIX compilers do not accept ANSI C code. For this reason, make sure that the correct flags are passed on the command line to switch on ANSI C mode. Some systems also require explicit links to network libraries. The SCAuto library is named `libscauto.a`.

On UNIX, a makefile might look like this:

```
# A sample makefile for HPUX UNIX.

# HPUX defines
#
CFLAGS= -Aa -I../lib
LDFLAGS=

sdktest: sdktest.o
$(CC) $(CFLAGS) sdktest.o ../lib/libscauto.a $(LDFLAGS) -o sdktest
```

6 C language API

The primary interface for HP ServiceCenter Automatic (SCAuto) Software Development Kit (SDK) consists of a small set of C functions, which are the same under all supported operating systems. These functions enable you to create, delete, and read HP Service Manager events. To use the C functions effectively, you should have at least a conceptual understanding of events and the Event Manager.

One important concept used by the C functions is a session. An SCAuto SDK session corresponds to a connection with SCAuto Listener. Each SCAuto SDK session appears as a distinct session from within Service Manager. When a session is created, it is given an identifying name. This name is then passed as a parameter to subsequent SCAuto SDK function calls.

Each primary SCAuto SDK function takes a session name as a parameter, along with other needed parameters. If it succeeds, each function returns the integer `0`. If it does not succeed, it returns a special error code. After each function call, you should check these error codes.

Each C program that uses SCAuto SDK function calls should include the C header file `scauto.h`. This header file declares prototypes for all SCAuto SDK functions. It also defines error codes that can be returned. The header file is set up for compilers that can process standard C (also known as ANSI or ISO C). Some compilers require that you specify a flag to switch on ANSI C compatibility mode. Other compilers use standard C by default.

Event structure

The structure definition for the scauto_event type looks like this:

```
struct scauto_event
{
char* evfields;
char  evtype[17];
char  evtime[20];
char  evsysseq[33];
char  evusrseq[33];
char  evsysopt[25];
char  evuser[25];
char  evpswd[25];
char  evsepchar;
};
```

This structure type is used for variables that hold an event definition. It is used to pass events to and retrieve events from SCAuto.

The fields in the structure definition correspond to fields of the same name within Service Manager Event Services. Except for evsepchar, all strings are standard null-terminated C strings.

The fields are defined as follows:

evtype

Type of event (for example, pmo for problem open, e-mail for electronic mail, and so on). For a list of standard event types, see the *HP Service Manager Event Services Guide*. You can also create custom event types.

evtime

Time the event occurred. The time should be in the format that the event scheduler expects. This format varies, depending on the operator record of the event scheduler, as well as the defaults of Service Manager. Normally, the event scheduler uses UTC time (also known as Greenwich Mean Time) and the format MM/DD/YYYY HH:MM:SS. If the field is empty while scauto_create is called, it is filled in automatically.

evsysseq

Reserved sequence number for the event. This number is a unique keyed field. You can save the field and then use it in scauto_query to start the query after the last retrieved event. Because the number is unique, scauto_create ignores any values placed in the field and creates its own.

`evusrseq`

User-specified sequence number. This number may be used for any purpose the application needs. In particular, it may be used to track an input event to an output event. All standard events that create an output event copy the `evusrseq` field from the input event.

`evsysopt`

Reserved field.

`evuser`

User or application for which the event is scheduled.

`evpswd`

Application password, if required.

`evsepchar`

Application separator character to use for parsing `evfields`. Normally, this field is set to a caret (`^`) character. It should keep that value, if possible, for consistency.

`evfields`

All of the application-specific data. For the standard event types, all sub-fields within this field are separated by the character specified in `evsepchar`. The filed is not a fixed-length string. Rather, it is a pointer to an allocated string of any length. The program that allocates this value is also responsible for freeing the string.

Error codes

If they succeed, SCAuto SDK functions return `0`. If they fail, they return an error code. Symbolic names for these error codes are defined in the `scauto.h` C header file.

Symbolic error codes

Each function description in Functions on page 48 lists several possible error codes that it may return. SCAuto SDK functions `scauto_error_string` and `scauto_error` can provide additional error information.

SCAuto SDK functions use the following symbolic error codes:

`NOCONN (800)`

No connection could be established.

`INVID (801)`

An invalid session identifier was passed as a parameter.

`LICENCE (803)`

License check failed.

`STORERR (807)`

Error while allocating memory.

`DUPEID (808)`

Session identifier was not unique.

`NOEVFND (809)`

No more events were found.

`NOQUERY (810)`

Function `scauto_query` was not called before the function `scauto_get_next`.

`SYNTAX (812)`

Network protocol error occurred.

`DELERR (813)`

Service Manager could not delete an event.

`INSERR (814)`

Service Manager could not create an event.

`BADREQ (816)`

Command was not understood by Listener. It is possible that the server needs to be upgraded.

Networking error codes

SCAuto SDK uses codes to indicate networking errors. For these errors, additional information is available from the `scauto_error` function.

SCAuto SDK uses the following networking error codes:

`NETERR_SOCKINIT (101)`

Error while initializing networking libraries. Make sure that networking libraries are installed.

`NETERR_GETHOST (102)`

Host name lookup failed. Make sure that the host name passed to `scauto_connect` is correct, and that it exists in the system `hosts` file.

`NETERR_GETSERV (103)`

Service name lookup failed. Make sure that the service name passed to `scauto_connect` is correct, and that it exists in the system `services` file.

`NETERR_SOCKET (104)`

Could not create a network socket.

`NETERR_CONNECT (105)`

Connection to a remote Listener failed. Make sure that the server is running, and that the network is up.

`NETERR_BIND (106)`

Could not bind to a network address.

`NETERR_SOCKSEND (107)`

Error while sending data over the network.

`NETERR_SOCKRECV (108)`

Error while reading data from the network.

`NETERR_TIMEOUT (109)`

Timeout while waiting for server response. Timeouts occur if the SCAuto client tries to connect to a Service Manager server instead of an SCAuto server.

`NETERR_LOSTCONN (110)`

Remote connection closed or server process died.

Functions

This section lists SCAuto SDK functions, as well as the possible error codes for each function.

scauto_connect

NAME

`scauto_connect()` - connect to Service Manager.

SYNOPSIS

```
#include "scauto.h"
int scauto_connect(char *id, char *server, char *events, char
*users);
```

DESCRIPTION

The `scauto_connect` function connects the application to Service Manager. This function must be performed before any other SCAuto function that takes a session identifier as a parameter. Use `scauto_disconnect` to close and abandon all sessions opened with `scauto_connect`.

`id`

Name of the SCAuto session that is created. This name is used to uniquely identify the session in subsequent SCAuto function calls.

`server`

Host name and service name of the Listener to which you want to connect. Separate the host name and service name with a single period (`.`). If you specify a service name without a host name, the local machine is assumed. For example, using `joe.scauto1` with this parameter means that Listener is sought on the machine `joe` by using the service name `scauto1`.

`events`

Event type or list of event types to retrieve when calling subsequent `scauto_query` and `scauto_get_next` functions. A value of `all` causes queries to fetch all event types. You can specify multiple event types in the format `(type1,type2,...)`.

`user`

User name or list of user names to retrieve on subsequent queries. A value of `all` causes events owned by any user to be retrieved. You can specify multiple user names in the format `(user1,user2,...)`.

RETURN VALUE

If it competes successfully, the `scauto_connect` function returns `0`. Otherwise, it returns an error code. To get extended error information, call `scauto_error`.

ERRORS

The `scauto_connect` function can return the following symbolic error codes:

`DUPEID`

ID passed is in use by another session.

`NOCONN`

Connection could not be created.

`STORERR`

Problem allocating memory.

`NETERR_CONNECT`

Problem connecting to a remote server. Make sure that the server is running, and that the network is up.

`NETERR_TIMEOUT`

Timeout waiting for a server response. Make sure that `server` refers to a SCAuto server, not a Service Manager server.

scauto_disconnect

NAME

`scauto_disconnect()` - disconnect from Service Manager.

SYNOPSIS

```
#include "scauto.h"
int scauto_disconnect(char *id);
```

DESCRIPTION

The `scauto_disconnect` function closes a connection to Service Manager, and de-allocates any resources used by the session.

`id`

Identifier for the session to be closed. This identifier must be the same as that passed to `scauto_connect`.

RETURN VALUE

If it completes successfully, the `scauto_disconnect` function returns `0`. Otherwise, it returns an error code.

ERRORS

The `scauto_disconnect` function can return the following error codes:

`NOCONN`

No previous connection was established.

`INVID`

No session with the name `id` was found.

scauto_query

NAME

`scauto_query()` - retrieve a Service Manager event.

SYNOPSIS

```
#include "scauto.h"
int scauto_query(char *id, char *startseq, char *options,
struct scauto_event *result);
```

DESCRIPTION

The `scauto_query` function retrieves an event, starting with the sequence number specified by the `startseq` parameter. Only events that match the event types and user names specified in the `scauto_connect` call are retrieved. Only one event is retrieved with this function. To retrieve subsequent events, use `scauto_get_next`.

`id`

SCAuto session identifier.

`startseq`

Starting sequence number for queried events. Only events with higher sequence numbers (the `evsysseq` field) are retrieved. This number can be saved and used when restarting an application to resume processing events from where it left off.

`options`

A value of `wait` causes the query to wait until an event is available for retrieving. If no events are available, `scauto_query` immediately returns an error code of `NOEVFND`.

`result`

When `scauto_query` returns, the retrieved `scauto_event` structure as defined in `scauto.h`.

The `wait` option is ignored if used on Microsoft Windows 3.1.

RETURN VALUES

If it completes successfully, the `scauto_query` function returns `0`, and the result points to a valid `scauto_event` structure. Otherwise, it returns a symbolic error code.

ERRORS

The `scauto_query` function can return the following error codes:

`NOEVFND`

No matching event was found.

`INVID`

Invalid session identifier was passed.

`STORERR`

Problem allocating memory.

scauto_query_ex

NAME

`scauto_query_ex()` - retrieve a Service Manager event based on the `szQuery` option.

SYNOPSIS

```
#include "scauto.h"
scauto_query_ex(char *id, char *szQuery, char *options );
```

DESCRIPTION

The `scauto_query_ex` function queries for an event by looking for the event that matches the `szQuery` string.

To retrieve the event and any subsequent events, use `scauto_get_next`.

`id`

SCAuto session identifier.

`szQuery`

Query string (for example, `"evusreq=\"12345678901234\""`).

`options`

A value of `wait` causes the query to wait until an event is available for retrieving. If no events are available, `scauto_query_ex` immediately return an error code of `NOEVFND`.

RETURN VALUES

If the `scauto_query_ex` function completes successfully, it return `0`. Otherwise, it returns a symbolic error code.

ERRORS

The `scauto_query_ex` function can return the following error codes:

`NOEVFND`

No matching event was found.

`INVID`

Invalid session identifier was passed.

`STORERR`

Problem allocating memory.

scauto_get_next

NAME

`scauto_get_next()` - retrieve next Service Manager event.

SYNOPSIS

```
#include "scauto.h"
int scauto_get_next(char *id, struct scauto_event *result);
```

DESCRIPTION

The `scauto_get_next` function retrieves an event after an `scauto_query` or `scauto_query_ex` function call. For example, `scauto_query` is called to retrieve the first event, and `scauto_get_next` can then be repeatedly called to retrieve the remaining matching events. By contrast, `scauto_query_ex` is called to query for events, `scauto_get_next` is called to retrieve the first event, and then `scauto_get_next` can be repeatedly called to retrieve the remaining matching events. Typical usage is to call `scauto_get_next` repeatedly in a loop until it returns `NOEVFND`.

id

SCAuto session identifier.

result

Pointer to a retrieved event if `scauto_get_next` is successful.

RETURN VALUES

If the `scauto_get_next` function completes successfully, it returns `0`, and `result` points to a valid `scauto_event` structure. Otherwise, it returns a symbolic error code.

ERRORS

The `scauto_get_next` function can return the following error codes:

`NOEVFND`

No matching event was found.

`INVID`

Invalid session identifier was passed.

`STORERR`

Problem allocating memory.

scauto_create

NAME

`scauto_create()` - create a Service Manager event.

SYNOPSIS

```
#include "scauto.h"
int scauto_create(char *id, struct scauto_event *event);
```

DESCRIPTION

The `scauto_create` function creates a new Service Manager event. The Event Manager in Service Manager processes the event, and executes the requested RAD applications.

`id`

Identifier for this session.

`event`

Pointer to an `scauto_event`. Unused fields should be initialized to null strings. The `evsysseq` field is ignored.

RETURN VALUE

If the `scauto_create` function completes successfully, it returns `0`. Otherwise, it returns a symbolic error code.

ERRORS

The `scauto_create` function can return the following error codes:

`INVID`

Invalid session ID was passed.

`INSERR`

Service Manager returned an error when trying to create the event. Check the Service Manager log file for possible error messages.

scauto_delete

NAME

`scauto_delete()` - delete an event in Service Manager.

SYNOPSIS

```
#include "scauto.h"
int scauto_delete(char *id, struct scauto_event *event);
```

DESCRIPTION

The `scauto_delete` function deletes the event that matches the passed `event` parameter. The event is deleted from the Event Manager output events, not from the input events.

The `event` passed should be an event that was retrieved previously by using `scauto_query` or `scauto_get_next`. Because other applications may be using this event at the same time as your application, make sure that yours is the only application using this event record before deleting it.

`id`

SCAuto session ID.

`event`

Pointer to the event to delete.

RETURN VALUE

If the `scauto_delete` function completes successfully, it returns `0`. Otherwise, it returns a symbolic error code.

ERRORS

The `scauto_delete` function can return the following error codes:

`INVID`

Invalid session ID was passed.

`NOEVFND`

Event to delete was not found.

`DELERR`

Service Manager returned an error while trying to delete the event. Check the Service Manager log file for possible error messages.

scauto_error

NAME

`scauto_error()` - return OS- or network-specific error code.

SYNOPSIS

```
#include "scauto.h"
int scauto_error(char * id);
```

DESCRIPTION

Sometimes, an SCAuto error occurs because of error indications in the operating system or network interface. These errors are saved for each SCAuto session, and can be retrieved with the `scauto_error` function. These codes can provide more detailed information than the error codes returned from SCAuto functions. The codes are invaluable when developing and debugging applications.

On UNIX operating systems, the `errno` value is saved. On Windows operating systems, the `GetLastError()` value is saved.

`id`

Identifier for the SCAuto session from which the error is to be retrieved.

RETURN VALUE

The appropriate saved error code, if any, is returned. If the previous SCAuto function did not have an error, or if there was no appropriate error code to save, then `0` is returned. If the passed session ID is invalid, then `-1` is returned.

ERRORS

The specific error codes returned are specific to the operating system used.

scauto_error_string

NAME

`scauto_error_string()` - return a descriptive string for an SCAuto error. code.

SYNOPSIS

```
#include "scauto.h"
char *scauto_error_string(int error);
```

DESCRIPTION

The `scauto_error_string` function takes an SCAuto error code as a parameter, and returns a short descriptive string for that error. This function is intended for debugging convenience, as well as for providing descriptive error messages to end users. No locale is used. Currently, only English-language messages are provided in the library.

`error`

Code for which a description is returned.

RETURN VALUE

A null-terminated character string with no embedded carriage returns.

ERRORS

None.

scauto_version

NAME

`scauto_version()` - return string describing current SCAuto version.

SYNOPSIS

```
#include “scauto.”
char *scauto_version(void);
```

DESCRIPTION

The `scauto_versio` function returns a string that describes the current SCAuto version as well as the date the SCAuto library or DLL was built. It is intended to be used for debugging or for presenting descriptive text on application startup.

RETURN VALUE

A null-terminated character string constant containing the version.

ERRORS

None.

A Troubleshooting

This appendix provides tips for troubleshooting HP ServiceCenter Automatic (SCAuto) Software Development Kit (SDK).

Contacting HP Support

If you need assistance in the resolution of your problem, make sure you have the following data available before calling HP Support:

- Core files
- Error logs produced by your program or SCAuto application with the `-debug` option (`scsubmit`, `scfiled`).
- Input and output files for `scfiled` errors
- Standard output and error files for manually executed `scsubmit` commands
- Service Manager log and Scheduler logs (if specified)
- Service Manager `msglog` for affected applications or services
- Service Manager print of agent status (Event Services menu)
- Unloaded Event records relevant to the errors
- Unloaded filter specifications (if relevant)

Listener-to-application connection fails

If the connection between SCAuto Listener and the SCAuto application fails, make sure that Listener is running, that the TCP/IP specifications are correct, and that there is connectivity between the server and client.

Problem

The connection between Listener and the SCAuto application (for example, `sdktest`) fails.

Solution

Do the following:

1. Make sure Listener is running on the Service Manager server platform.

   To check, run the Service Manager `status` command:

   - If Listener is *not* running, start it from the status option `start schedulers`, and then select `scauto.startup`.

     If the start fails, the Listener daemon writes log messages to the `sm.log` file. Check the log for any error messages. Also, check the TCP/IP specifications for your platform. A service name other than that specified for Service Manager server is required. The Listener keyword specified in the file must match this name.

   - If Listener is running, check the service name specified in the file on the Service Manager platform for the Listener keyword.

     If the keyword is not specified, Listener defaults to the service name `scauto`. This name is useful when checking TCP/IP specifications.

2. Make sure the TCP/IP specifications are correct on the SCAuto platform.

   Make sure the host and service are specified, and that they match the host and service name of Listener.

   To verify the specifications, you can use either of the following:

   - *Local files*

     On UNIX, these files are located in the following directories:

     `/etc/hosts`

     `/etc/services`

   - *Name server*

To get specifications, ask your network administrator.

Also, check your specification in the program, configuration file, or passed parameters. Make sure they all match.

Example:

```
scauto_connect (id, server,...), scfiled -server host.service
```

3 Make sure there is connectivity between the Listener (server) and the SCAuto application (client).

Ping the Listener platform from the SCAuto application (client) platform. If the ping fails, and all TCP/IP specifications are correct, contact your network administrator for assistance.

Internal event: RAD application not invoked

If an internal event occurs, but no RAD application is invoked, check Event Scheduler and the review schedule record specifications.

Problem

You have an event in the `event.in` file, but no RAD application is invoked. for example, you have an `pmo` event, but the problem is not opened.

Solution

Do the following:

1 Make sure the Event Scheduler is running on the Service Manager server platform.

To check, run the Service Manager `status` command:

- If the Event Scheduler is *not* running, start it from the status option `start schedulers`, and then select `event.startup`.

  If the start fails, review error messages and retry. If errors persist, call HP Support. For instructions, see Contacting HP Support on page 61.

- If the Event Scheduler is running but *not* processing, stop the Event Scheduler, build a new schedule record, and restart.

2 Review schedule record specifications.

For details, see the troubleshooting section of the *HP Service Manager Event Services Guide*.

External event: SCAuto application not invoked

If an external event occurs, but the external SCAuto application is not invoked, check SCAuto Listener and the SCAuto application.

Problem

You have an event in the `event.out` file, but the external SCAuto application is not invoked. For example, you have an `scfiled` event, but no event in output file.

Solution

Check the following:

1. Make sure there is connectivity to Listener.

   For instructions, see Listener-to-application connection fails on page 62.

2. Make sure the SCAuto application (for example, `scfiled`) is active and retrieving the event:

   If the SCAuto application is *not* running, check the `scfiled.log` for errors.

   Correct errors and restart, using appropriate command with the debug option, if available.

   The debug option (`-debug`) enables you to determine if the event was retrieved, or whether the retrieval failed:

   - If the event was retrieved, it is an SCAuto application failure.
   - If event was *not* retrieved, check the SCAuto application for connect and sequence number specification:

     Example:

     ```
     scauto_connect (...,events, user), scauto_query (id,
     startseq,.), scfiled -evlist -user -checkpoint
     ```

     If you do *not* specify these fields correctly, events are `not` retrieved. The sequence number of the event to be retrieved must be equal to or greater than your specification. Also, you must specify the events you will service on your connect statement or `scfiled` parameter. If you specify anything other than `all` for your event and user specifications, make sure the event in the `event.out` file matches your criteria.

No internal event: SCAuto application event occurs

If an internal event does *not* occur, but an external SCAuto application event occurs, check SCAuto Listener and the SCAuto application.

Problem

You do *not* have an event in the `event.in` file, but an external SCAuto application event occurs. For example, a `pmo` event is sent, but is not in Service Manager.

Solution

Check the following:

1. Make sure there is connectivity to Listener.

   For instructions, see Listener-to-application connection fails on page 62.

2. Make sure the SCAuto application (for example, `scfiled`) is active.

   If the SCAuto application is *not* running, check the `scfiled.log` for errors. Correct the errors and restart, using appropriate command with the debug option, if available.

   The debug option enables you to determine if the failure is an SCAuto application failure or a connection failure. Common application failures are invalid event structures in the `scfiled input <`*`infile`*`>` or invalid specifications on the `scauto_create (id,...)` statement. Your application should check all return codes and log any errors.

No external event: RAD application invoked

If an external event does *not* occur, but the RAD application is invoked, see the *HP Service Manager Event Services Guide*.

Problem

You do *not* have an event in the `event.out` file, but the RAD application is invoked. For example, an email is sent, but there is no `email` event for it.

Solution

Review the troubleshooting section of the *HP Service Manager Event Services Guide*.

Index