Chapter 1

Integrating Third-Party Network Devices into the SRC Network with the SRC CLI

This chapter describes how to use the SRC CLI to integrate third-party network devices into the SRC network. You can use the CLI to configure the SRC software on a Solaris platform or on a C-series Controller.

You can also use the C-Web interface to integrate third-party devices. For more information, see *SRC-PE C-Web Interface Configuration Guide, Chapter 20, Integrating Third-Party Network Devices into the SRC Network with the C-Web Interface*.

Topics in this chapter include:

Overview of Integrating Network Devices into the SRC Network

You can integrate third-party routers and other network devices into your SRC network. The SAE provides a driver that you can use to integrate the SAE with a third-party device. This device driver uses the session store to store and replicate subscriber and service session data within a community of SAEs.

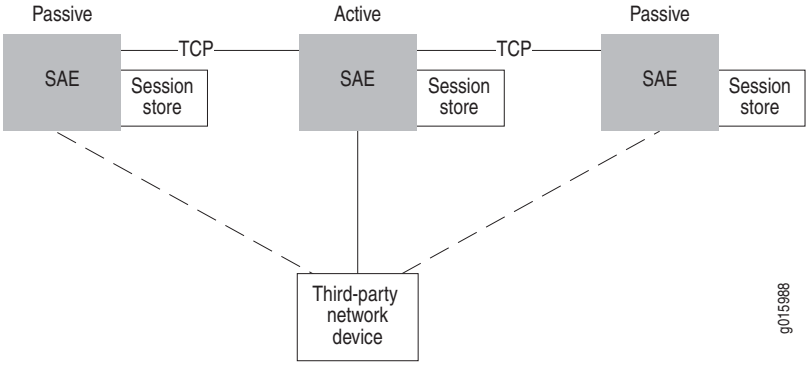
To log in subscribers to the SAE, you use assigned IP subscribers or event notification from an IP address manager.

To activate services and provision policies on the device, you use script services. You can also activate aggregate services for subscribers. However, you cannot activate normal services that require policies to be provisioned on the device.

SAE Communities

For SAE redundancy in an SRC network, you can have a community of two or more SAEs. SAEs in a community are given the role of either active SAE or passive SAE. The active SAE manages the connection to the network device and keeps session data up to date within the community. Figure 1 shows a typical SAE community.

Figure 1: SAE Community



When an SAE starts, it negotiates with other SAEs to determine which SAE controls the network device. The SAE community manager and members of the community select the active SAE.

A passive SAE needs to take over as active SAE in any of the following cases:

- The active SAE shuts down. In this case, the active SAE notifies the passive SAEs, and one of the passive SAEs takes over as active SAE.
- A passive SAE does not receive a keepalive message from the active SAE within the keepalive interval. In this case, the passive SAE attempts to become the active SAE.

Storing Session Data

To aid in recovering from an SAE failover, the SAE stores subscriber and service session data. When the SAE manages a network device, session data is stored in the SAE host's file system. The SRC component that controls the storage of session data on the SAE is called the session store. The session store queues data and then writes the data to session store files on the SAE host's disk. Once the data is written to disk, it can survive a server reboot.

For more information, see *Storing Subscriber and Service Session Data* in *SRC-PE Network Guide, Chapter 2, Configuring the SAE with the SRC CLI*.

Using Script Services to Provision Third-Party Devices

You use script services to activate services and provision policies on third-party network devices. A script service is a service into which you can insert or reference a script. You write a script that will activate services and provision policies on the third-party device, and then you insert the script into the script service or reference the script in the service. When the SAE activates a service, it runs the script. The script provisions policies on the device using a means that the device supports. You can also include an interface in the script that causes the SAE to send authentication and tracking events when it activates, modifies, or deactivates a script service session.

The SAE core API includes two interfaces for creating a script:

- ScriptService—Defines a service provider interface (SPI) that the script service must implement. The implementation of the ScriptService interface activates, modifies, or deactivates the service.

- ServiceSessionInfo—Provides a callback interface into the SAE and provides information about the service session to the script service.

For information about the ScriptService interface and the ServiceSessionInfo interface, see the script service documentation in the SRC software distribution in the folder *SDK/doc/sae* or in the SAE core API documentation on the Juniper Networks Web site at

http://www.juniper.net/techpubs/software/management/sdx/api-index.html

You can write the script in Java or Jython.

Logging In Subscribers and Creating Sessions

You can use two mechanisms to obtain subscriber address requests and other information and to set up a pseudointerface on the network device. (You must choose one mechanism; you cannot mix them.)

1. Assigned IP subscriber. The SAE learns about a subscriber through subscriber-initiated activities, such as activating a service through the portal or through the SRC SOAP Gateway (SRC-SG).

   With this method, you use the assigned IP subscriber login type along with the network interface collector (NIC) to map IP addresses to the SAE.

2. Event notification from an IP address manager. The SAE learns about subscribers through notifications from an external IP address manager, such as a DHCP server or a RADIUS server.

   With this method, you use the event notification application programming interface (API). The API provides an interface to the IP address manager, and lets the IP address manager notify the SAE of events such as IP address assignments.

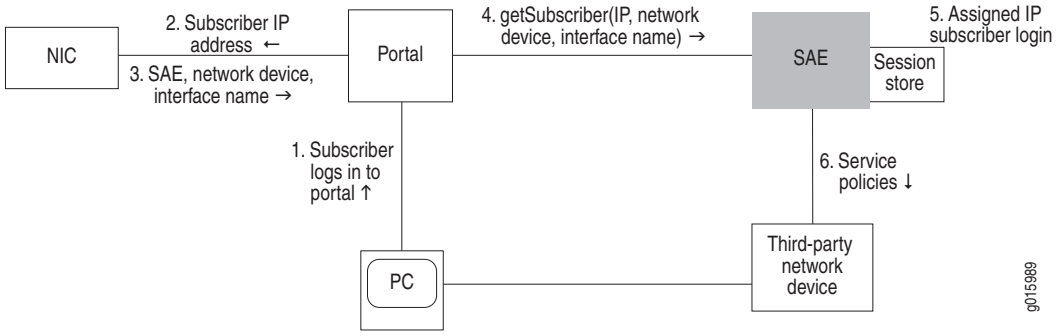
Assigned IP Subscribers

With the assigned IP subscriber method of logging in subscribers and creating sessions, the SRC software uses IP address pools along with network information collector (NIC) resolvers to provide mapping of IP addresses to SAEs. You configure the static address pools or dynamically discovered address pools in the virtual router configuration for a network device. These pools are published in the NIC. The NIC maps subscriber IP addresses in requests received through the portal or SRC-SG to the SAE that currently manages that network device.

Login Interactions with Assigned IP Subscribers

This section describes login interactions for assigned IP subscribers. In the example shown in Figure 2, the subscriber activates a service through a portal. You could also have the subscriber activate a service through the SRC-SG.

Figure 2: Login Interactions with Assigned IP Subscribers



The sequence of events for logging in and creating sessions for assigned IP subscribers is:

1. The subscriber logs in to the portal.
2. The portal sends the subscriber's IP address to the NIC.
3. Based on the IP address, the NIC looks up the subscriber's SAE, network device, and interface name, and returns this information to the portal.
4. The portal sends a getSubscriber message to the SAE. The message includes the subscriber's IP address, network device, and interface name.
5. The SAE creates an assigned IP subscriber and performs a subscriber login. Specifically, it:
    a. Runs the subscriber classification script with the IP address of the subscriber. (Use the ASSIGNEDIP login type in subscriber classification scripts.)
    b. Loads the subscriber profile.
    c. Runs the subscriber authorization plug-ins.

d. Runs the subscriber tracking plug-ins.

e. Creates a subscriber session and stores the session data in the session store file.

6. The SAE pushes service policies for the subscriber session to the network device.

Because the SAE is not notified when the subscriber logs out, the assigned IP idle timer begins when no service is active. The SAE removes the interface subscriber session when the timeout period ends.

Event Notification from an IP Address Manager

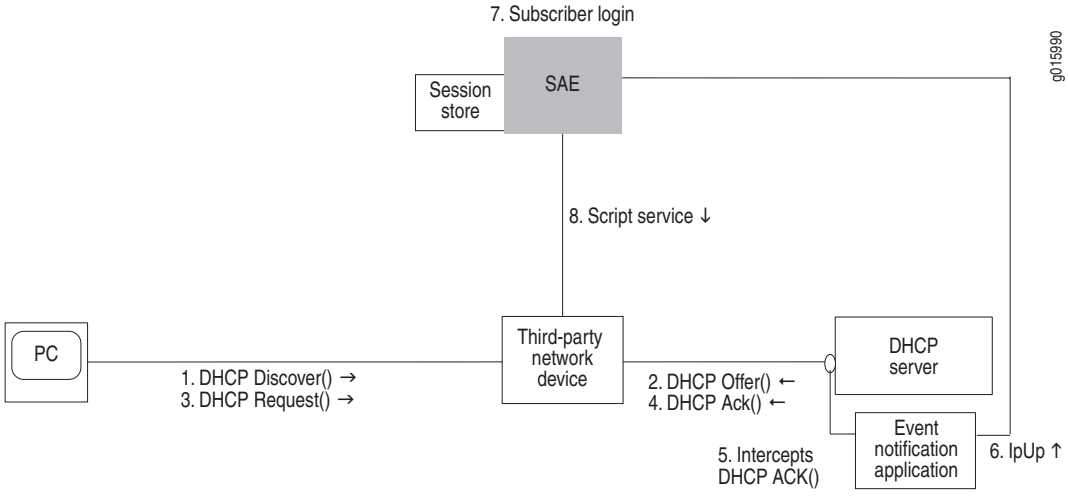
With the event notification method of logging in subscribers and creating subscriber sessions, the subscriber logs in to the network device and obtains an IP address through an address server, usually a DHCP server. The SAE receives notifications about the subscriber, such as the subscriber's IP address, from an event notification application that is installed on the DHCP server.

To use this method of logging in subscribers, you can use the event notification API to create the application that notifies the SAE when events occur between the DHCP server and the network device. You can also use Monitoring Agent, a sample application that was created with the event notification API and that monitors DHCP or RADIUS messages for DHCP or RADIUS servers. See the *SRC Sample Applications Guide*.

Login with Event Notification

This section describes login interactions by means of event notifications.

Figure 3: Login Interactions with Event Notification Application



The sequence of events for logging in subscribers and creating sessions is:

1. The DHCP client in the subscriber's computer sends a DHCP discover request to the DHCP server.
2. The DHCP server sends a DHCP offer to the subscriber's DHCP client.
3. The DHCP client sends a DHCP request to the DHCP server.
4. The DHCP server acknowledges the request by sending a DHCP Ack message to the DHCP client.
5. The event notification application that is running on the DHCP server intercepts the DHCP Ack message.
6. The event notification application sends an ipUp message to the SAE that notifies the SAE that an IP address is up.
7. The SAE performs a subscriber login. Specifically, it:
    a. Runs the subscriber classification script.
    b. Loads the subscriber profile.
    c. Runs the subscriber authorization plug-ins.
    d. Runs the subscriber tracking plug-ins.
    e. Creates a subscriber session and stores the session in the session store file.
8. The SAE can start script services.

The ipUp event should be sent with a timeout set to the DHCP lease time. The DHCP server sends an ipUp event for each Ack message sent to the client. The SAE restarts the timeout each time it receives an ipUp event.

If the client explicitly releases the DHCP address (that is, it sends a DHCP release event), the DHCP server sends an ipDown event. If the client does not renew the address, the lease expires on the DHCP server and the timeout expires on the SAE.

Configuration Tasks for Integrating Third-Party Network Devices

To integrate third-party devices into your SRC network, complete the following tasks:

- Write a script and add a script service that references the script.

  See *Setting Up Script Services* on page 10.

- Add objects for the devices.

  See *Adding Objects for Network Devices* on page 10.

- Configure an SAE community.

  See *Setting Up SAE Communities* on page 12.

- (Optional) Configure SAE properties for the Event Notification API if you are using the event notification method to log in subscribers.

  See *Configuring SAE Properties for the Event Notification API* on page 14.

- Configure the session store.

  See *Storing Subscriber and Service Session Data* in *SRC-PE Network Guide, Chapter 2, Configuring the SAE with the SRC CLI*.

- If you are using the event notification method to log in subscribers, integrate the SAE with an IP address manager. There are two ways to do so:

  - Use the event notification API to create an application that notifies the SAE when events occur between the DHCP server and the network device.

    See the event notification API documentation in the SRC software distribution in the folder *SDK/doc/sae* or in the SAE CORBA remote API documentation on the Juniper Networks Web site at

    http://www.juniper.net/techpubs/software/management/sdx/api-index.html

  - Use Monitoring Agent, a sample application that was created with the event notification API and that monitors DHCP or RADIUS messages for DHCP or RADIUS servers.

    See the *SRC Sample Applications Guide*.

Setting Up Script Services

To set up script services:

1. Write a script that implements the ScriptService interface, a service provider interface (SPI) for the SAE.

   See *SRC-PE Services and Policies Guide, Chapter 1, Managing Services with the SRC CLI*.

   See the script service documentation in the SRC software distribution in the folder *SDK/doc/sae* or in the SAE core API documentation on the Juniper Networks Web site at

   http://www.juniper.net/techpubs/software/management/sdx/api-index.html

2. Add a script service that references the script.

   See *SRC-PE Services and Policies Guide, Chapter 1, Managing Services with the SRC CLI*.

Adding Objects for Network Devices

For each network device that the SAE manages, add a router object and virtual router object.

Use the following configuration statements to add a router object:

```
shared network device name {
    description description;
    management-address management-address;
    device-type (junose| junos| pcmm| proxy);
    qos-profile [qos-profile...];
}
```

To add a router object:

1. From configuration mode, access the configuration statements that configure network devices. This sample procedure uses proxy_device as the name of the router.

   ```
   user@host# edit shared network device proxy_device
   ```

2. (Optional) Add a description for the router object.

   ```
   [edit shared network device proxy_device]
   user@host# set description description
   ```

3. (Optional) Add the IP address of the router object.

   ```
   [edit shared network device proxy_device]
   user@host# set management-address management-address
   ```

4. Set the type of device that you are adding to proxy.

   [edit shared network device proxy_device]
   user@host# **set device-type proxy**

5. (Optional) Verify your configuration.

```
[edit shared network device proxy_device]
user@host# show
description "Third-party router";
management-address 192.168.9.25;
device-type proxy;
interface-classifier {
  rule rule-0 {
    script #;
  }
}
```

Adding Virtual Router Objects

Use the following configuration statements to add a virtual router:

shared network device name virtual-router *name* {
    sae-connection [*sae-connection*...];
    snmp-read-community *snmp-read-community*;
    snmp-write-community *snmp-write-community*;
    scope [*scope*...];
    tracking-plug-in [*tracking-plug-in*...];
}

To add a virtual router:

1. From configuration mode, access the configuration statements for virtual routers. This sample procedure uses proxy_device as the name of the router object. For third-party devices, use the name default for the virtual router.

   user@host# **edit shared network device proxy_device virtual-router default**

2. Specify the addresses of SAEs that can manage this router. This step is required for the SAE to work with the router.

   [edit shared network device proxy_device virtual-router default]
   user@host# **set sae-connection** [*sae-connection*...]

3. (Optional) Specify an SNMP community name for SNMP read-only operations for this virtual router.

   [edit shared network device proxy_device virtual-router default]
   user@host# **set snmp-read-community** *snmp-read-community*

4. (Optional) Specify an SNMP community name for SNMP write operations for this virtual router.

   [edit shared network device proxy_device virtual-router default]
   user@host# **set snmp-write-community** *snmp-write-community*

5. (Optional) Specify service scopes assigned to this virtual router. The scopes are available for subscribers connected to this virtual router for selecting customized versions of services.

   [edit shared network device proxy_device virtual-router default]
   user@host# **set scope** [*scope*...]

6. (Optional) Specify the plug-ins that track interfaces that the SAE manages on this virtual router.

   [edit shared network device proxy_device virtual-router default]
   user@host# **set tracking-plug-in** [*tracking-plug-in*...]

7. (Optional) Verify your configuration.

```
[edit shared network device proxy_device virtual-router default]
user@host# show
sae-connection 10.8.221.45;
snmp-read-community ********;
snmp-write-community ********;
scope POP-Toronto;
tracking-plug-in flexRadius;
```

Setting Up SAE Communities

Tasks to configure SAE communities are:

- *Adding Virtual Router Objects* on page 11.
- *Configuring the SAE Community Manager* on page 12.
- *Specifying the Community Manager in the SAE Device Driver* on page 14.
- If there is a firewall in the network, configuring the firewall to allow SAE messages through.

Configuring the SAE Community Manager

Use the following configuration statements to configure the SAE community manager that manages third-party network device communities:

```
shared sae configuration external-interface-features name CommunityManager {
    keepalive-interval keepalive-interval;
    threads threads;
    acquire-timeout acquire-timeout;
    blackout-time blackout-time;
}
```

To configure the community manager:

1. From configuration mode, access the configuration statements for the community manager. In this sample procedure, sae_mgr is the name of the community manager.

   user@host# **edit shared sae configuration external-interface-features sae_mgr CommunityManager**

2. Specify the interval between keepalive messages sent from the active SAE to the passive members of the community.

   [edit shared sae configuration external-interface-features sae_mgr CommunityManager]
   user@host# **set keepalive-interval** *keepalive-interval*

3. Specify the number of threads that are allocated to manage the community. You generally do not need to change this value.

   [edit shared sae configuration external-interface-features sae_mgr CommunityManager]
   user@host# **set threads** *threads*

4. Specify the amount of time an SAE waits for a remote member of the community when it is acquiring a distributed lock. You generally do not need to change this value.

   [edit shared sae configuration external-interface-features sae_mgr CommunityManager]
   user@host# **set acquire-timeout** *acquire-timeout*

5. Specify the amount of time that an active SAE must wait after it shuts down before it can try to become the active SAE of the community again.

   [edit shared sae configuration external-interface-features sae_mgr CommunityManager]
   user@host# **set blackout-time** *blackout-time*

6. (Optional) Verify the configuration of the SAE community manager.

```
[edit shared sae configuration external-interface-features sae_mgr
CommunityManager]
user@host# show
CommunityManager {
  keepalive-interval 30;
  threads 5;
  acquire-timeout 15;
  blackout-time 30;
}
```

Specifying the Community Manager in the SAE Device Driver

Use the following configuration statements to specify the community manager in the SAE device driver.

```
shared sae configuration driver third-party {
    sae-community-manager sae-community-manager;
}
```

To specify the community manager:

1. From configuration mode, access the configuration statements for the third-party device driver.

   ```
   user@host# edit shared sae configuration driver third-party
   ```

2. Specify the name of the community manager.

   ```
   [edit shared sae configuration driver third-party]
   user@host# set sae-community-manager sae-community-manager
   ```

3. (Optional) Verify the configuration of the third-party device driver.

   ```
   [edit shared sae configuration driver third-party]
   user@host# show
   sae-community-manager sae_mgr;
   ```

Configuring SAE Properties for the Event Notification API

Use the following configuration statements to configure properties for the event notification API:

```
shared sae configuration external-interface-features name EventAPI {
    retry-time retry-time;
    retry-limit retry-limit;
    threads threads;
}
```

To configure properties for the event notification API:

1. From configuration mode, access the configuration statements for the event notification API. In this sample procedure, event_api is the name of the Event API configuration.

   ```
   user@host# edit shared sae configuration external-interface-features event_api
   EventAPI
   ```

2. Specify the amount of time between attempts to send events that could not be delivered.

   ```
   [edit shared sae configuration external-interface-features event_api EventAPI]
   user@host# set retry-time retry-time
   ```

3. Specify the number of times an event fails to be delivered before the event is discarded.

   [edit shared sae configuration external-interface-features event_api EventAPI]
   user@host# **set retry-limit** *retry-limit*

4. Specify the number of threads allocated to process events.

   [edit shared sae configuration external-interface-features event_api EventAPI]
   user@host# **set threads** *threads*

5. (Optional) Verify the configuration of the event notification API properties.

```
[edit shared sae configuration external-interface-features event_api
EventAPI]
user@host# show
EventAPI {
  retry-time 300;
  retry-limit 5;
  threads 5;
}
```

Developing Initialization Scripts for Network Devices

When the SAE establishes a connection with a network device, it can run a script to customize the setup of the connection. These scripts are run when the connection between a network device and the SAE is established and again when the connection is dropped.

We provide the IorPublisher script in the */opt/UMC/sae/lib* folder. The IorPublisher script publishes the interoperable object reference (IOR) of the SAE in the directory so that a NIC can associate a router with an SAE.

Interface Object Fields

Scripts for network devices interact with the SAE through an interface object called Ssp. The SAE exports a number of fields through the interface object to the script and expects the script to provide the entry point to the SAE.

Table 4 describes the fields that the SAE exports.

Table 4: Exported Fields

Ssp Attribute	Description
Ssp.properties	System properties object (class: java.util.Properties)—The properties should be treated as read-only by the script.
Ssp.errorLog	Error logger—Use the SsperrorLog.printIn (message) to send error messages to the log.
Ssp.infoLog	Info logger—Use the Ssp.infoLog.printIn (message) to send informational messages to the log.
Ssp.debugLog	Debug logger—Use the Ssp.debugLog.printIn (message) to send debug messages to the log.

The script must set the field Ssp.routerInit to a factory function that instantiates a router initialization object:

- <VRName>—Name of the virtual router object that has been configured for the network device in the format: virtualRouterName@RouterName
- <virtualIp>—Virtual IP address of the SAE (string, dotted decimal; for example: 192.168.254.1)
- <realIp>—Real IP address of the SAE (string, dotted decimal; for example, 192.168.1.20)
- <VRIp>—IP address of the virtual router (string, dotted decimal)
- <transportVR>—Name of the virtual router

The factory function must implement the following interface:

```
Ssp.routerInit(VRName,
virtualIp,
realIp,
VRIp,
transportVR)
```

The factory function returns an interface object that is used to set up and tear down a connection. A common case of a factory function is the constructor of a class.

The factory function is called directly after a connection is established. In case of problems, an exception should be raised that leads to the termination of the connection.

Required Methods

Instances of the interface object must implement the following methods:

- *setup()*—Is called when the connection is established and is operational. In case of problems, an exception should be raised that leads to the termination of the connection.
- *shutdown()*—Is called when the connection is terminated to the virtual router. This method should not raise any exceptions in case of problems.

Example: Initialization Script

The following script defines a router initialization class named *SillyRouterInit*. The interface class does not implement any useful functionality. The interface class just writes messages to the infoLog when the router connection is created or terminated.

```
class SillyRouterInit:
    def __init__(self, vrName, virtualIp, realIp, vrIp, transportVr):
        """ initialize router initialization object """
        self.vrName = vrName
        Ssp.infoLog.printin("SillyRouterInit created")
```

```
    def setup(self):
        """ initialize connection to router """
        Ssp.infoLog.printin("Setup connection to VR %(vrName)s" %
                    vars(self))

    def shutdown(self):
        """ shutdown connection to router """
        Ssp.infoLog.printin("Shutdown connection to VR %(vrName)s" %
                    vars(self))

#
# publish interface object to Ssp core
#
Ssp.routerInit = SillyRouterInit
```

Copying Initialization Scripts to the C-series Controller

If you use a script that is not provided with the SRC software, you need to use the file copy command to copy your script to the C-series Controller. For example:

user@host> **file copy ftp://user@myserver/routerinit.py /opt/UMC/sae/lib**
Password:

Specifying Initialization Scripts on the SAE

Use the following configuration statements to specify initialization scripts for third-party devices:

shared sae configuration driver scripts {
    extension-path *extension-path*;
    general *general*;
}

To configure initialization scripts for third-party devices:

1. From configuration mode, access the configuration statements that configure initialization scripts.

   user@host# **edit shared sae configuration driver scripts**

2. Specify the initialization script for third-party devices.

   [edit shared sae configuration driver scripts]
   user@host# **set general** *general*

3. Configure a path to scripts that are not in the default location, */opt/UMC/sae/lib*.

   [edit shared sae configuration driver scripts]
   user@host# **set extension-path** *extension-path*

4. (Optional) Verify your initialization script configuration.

   ```
   [edit shared sae configuration driver scripts]
   user@host# show
   ```

Using SNMP to Retrieve Information from Network Devices

You can use SNMP to retrieve information from a network device. For example, if you create a script that uses SNMP, specify the SNMP communities that are on the network device.

We recommend that you specify SNMP communities for each virtual router object. (See *Adding Virtual Router Objects* on page 11.) You can also configure global default SNMP communities.

Configuring Global SNMP Communities in the SRC Software

You can configure global default SNMP communities that are used if a VR does not exist on the router or the community strings have not been configured for the VR.

Use the following configuration statements to configure global default SNMP communities:

```
shared sae configuration driver snmp {
    read-only-community-string read-only-community-string;
    read-write-community-string read-write-community-string;
}
```

To configure global default SNMP communities:

1. From configuration mode, access the configuration statements that configure default SNMP communities.

   user@host# **edit shared sae configuration driver snmp**

2. Configure the default SNMP community string used for read access to the router.

   [edit shared sae configuration driver snmp]
   user@host# **set read-only-community-string** *read-only-community-string*

3. Configure the default SNMP community string used for write access to the router.

   [edit shared sae configuration driver snmp]
   user@host# **set read-write-community-string** *read-write-community-string*

Using the NIC Resolver

If you are using the assigned IP subscriber method of logging in subscribers, and you are using the NIC to determine the subscriber's SAE, you need to configure a resolver on the NIC. The OnePopDynamicIp sample configuration data supports this scenario. The OnePopDynamicIp configuration supports one point of presence (POP) and provides no redundancy. The realm for this configuration accommodates the situation in which IP pools are configured locally on each virtual router object.

You can access the OnePopDynamicIp configuration in the SRC CLI. See *SRC-PE Network Guide, Chapter 10, Configuring NIC with the SRC CLI* for information about configuring NIC scenarios.