Chapter 10

# Overview of Configuring Plug-Ins for Solaris Platforms

This chapter describes how to use SDX Configuration Editor and SDX Admin to configure plug-ins. It also shows how to configure internal, external, and state synchronization plug-ins.

You can also use the SRC CLI to configure a plug-ins on the C-series platform or on a Solaris platform. See *Chapter 9, Configuring Internal, External, and Synchronization Plug-Ins with the SRC CLI*.

Topics in this chapter include:



## Configuring Plug-Ins with SDX Configuration Editor

You can use SDX Configuration Editor to create a configuration object or modify an existing one. Before you can modify an existing object, you need to import the configuration objects from the directory into SDX Configuration Editor.

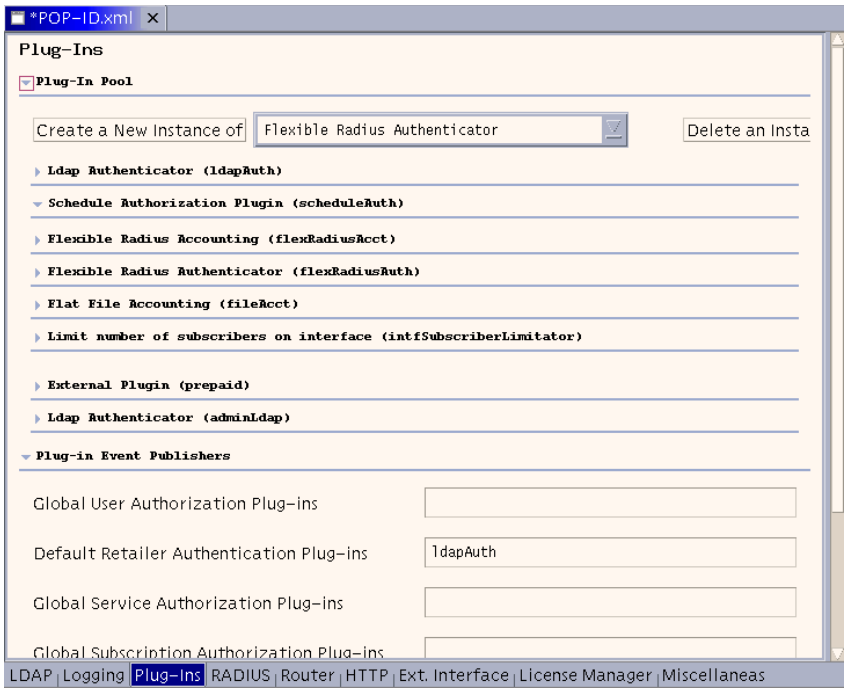
For information about how to use SDX Configuration Editor, see *SRC-PE Getting Started Guide, Chapter 39, Using SDX Configuration Editor*.

### *Accessing the Plug-In Configuration*

To access the plug-in pool and event publisher configuration:

1. In the navigation pane, select the SAE object for which you want to configure plug-ins.
2. Select the **Plug-Ins** tab.

The Plug-Ins pane appears. This screen shows the Plug-In Pool area and the Plug-in Event Publishers area.




- To expand a configuration, click the triangle to the left of the configuration that you want to expand. When the configuration is expanded, the triangle points down.
- To collapse a configuration, click the triangle to the left of the configuration. When the configuration is collapsed, the triangle points to the right.

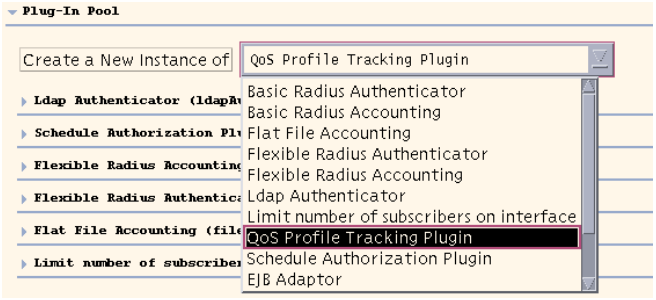
The Plug-In Pool area contains default plug-in instances that you can use as they are or modify. Instances are displayed by type of plug-in followed by the instance name in parentheses. For example, Ldap Authenticator (ldapAuth) is an LDAP authentication plug-in instance named ldapAuth.

The Plug-In Event Publishers area also contains several default plug-in instances.

### *Creating Plug-In Instances*

To create a plug-in instance:

1. In the plug-in pool, select the type of plug-in instance from the drop-down list, and click **Create a New Instance of**.




   The Create a New Instance dialog box appears.

2. Assign a name to the instance, and click **OK**.

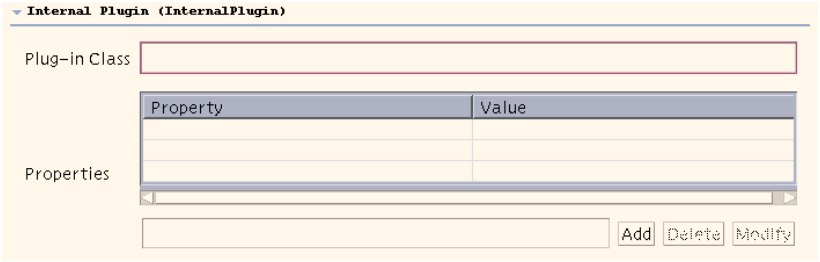
   The instance appears in the plug-in pool.

## Configuring Internal Plug-Ins

To configure an internal plug-in with SDX Configuration Editor:

1. In the Plug-In Pool area of the Plug-Ins pane, create an internal plug-in instance as described in *Creating Plug-In Instances* on page 143.

   The instance appears in the Plug-In Pool area.




2. Fill in the fields for the plug-in instance as described below.

***Plug-in Class***

- Class name of the plug-in.
- Value—Java class name of the plug-in
- Default—No value
- Property name—Class

***Properties***

- Properties that define the plug-in. Enter properties in the format:
  Plugin. < plug-in instance name > . < property name > = < expression >
- Configure the property table as follows:
  - To add a property, type the property definition in the field below the properties table, and click Add.
  - To modify a property, select the property, make your changes in the field below the property table, and click Modify.
  - To delete a property, select the property, and click Delete.
- Value—Property names and values that are available to the type of plug-in that you are configuring
- Default—No value

## Configuring the SAE for External Plug-Ins

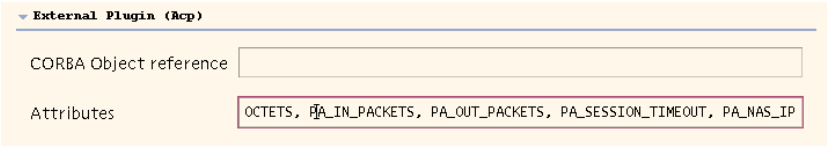
You need to configure SAE external plug-ins for SAE plug-in agents in the NIC, for Admission Control Plug-Ins, and for custom plug-ins developed in Common Object Request Broker Architecture (CORBA). For information about external plug-ins, see *SRC-PE Network Guide, Chapter 1, Overview of the SAE*.

When you use an external plug-in, you need to export its object reference to the SAE. When the SAE sends the first event to a registered plug-in, it resolves the object reference. In case of a failure, the SAE resolves the object reference again. In this case, if a plug-in restarts and instantiates a different object (that is, a different object reference), the SAE learns about the new object through the naming service or the file reference.

You can configure the SAE to resolve the object reference and specify which attributes to send to the external plug-in. To do so with SDX Configuration Editor:

1. In the Plug-In Pool area of the Plug-Ins pane, create an external plug-in instance as described in *Creating Plug-In Instances* on page 143.

   The instance appears in the Plug-In Pool area.




2. Fill in the fields for the plug-in instance as described below.

***CORBA Object reference***

- Object reference of the external plug-in that is exported to the SAE. When the SAE sends the first event to a registered external plug-in, it resolves the object reference.
- Value—Supply the object reference in one of the following forms:
  - The absolute path to the interoperable object reference (IOR) file in the form file:// < absolutePath >
  - The corbaloc URL in the format
    corbaloc:: < host > : < portNumber > / < path >
    - < host > —Name or IP address of the host that supports the plug-in
    - < portNumber > —TCP/IP port number
    - < path > —Absolute path to plug-in
  - Common Object Services (COS) naming service in the form:

    corbaname:: < host > [: < port > ][/NameService]# < key >

    - < key > —Provided by the publisher of the IOR to the COSnaming service.
  - The actual IOR in the form IOR: < objectReference >
- Default—No value
- Examples
  - Absolute path—file:///var/acp/acp.ior
  - corbaloc URL—corbaloc:boston:8801/acp
  - Actual IOR—
    IOR:000000000000002438444C3A736D67742E6A756E697...
- Property name—objectref

***Attributes***

- Attributes that are sent to the external plug-in.

**NOTE:** Configure only the attributes required. If you do not specify attributes, all attributes are sent. Specifying fewer attributes improves the performance of the SRC network.

- Value—Comma-separated list of plug-in attributes. For a list of attributes and descriptions, see the documentation for the sspPlugin.idl file in the SRC software distribution at */SDK/doc/idl/sspPlugin/html/index.html* or on the Juniper Networks Web site at

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

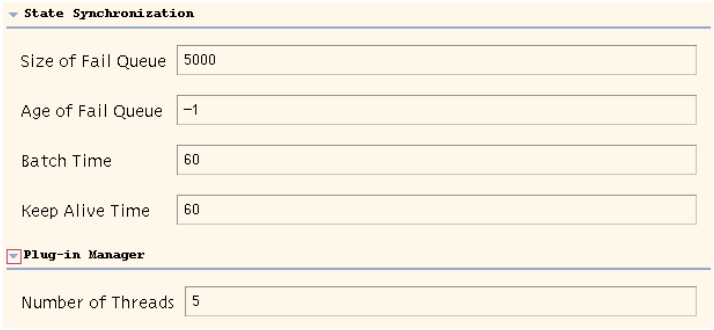
- Default—Comma-separated list of all possible attributes
- Property name—attr

## Configuring the State Synchronization Plug-In Interface

Some external plug-ins, such as the Admission Control Plug-In (ACP) application and the SAE plug-in agent for the NIC, support state synchronization with the SAE. The state synchronization plug-in interface allows external plug-ins to maintain the state of active subscriber, service, and interface sessions without having to store intermediate versions of the state locally.

To use SDX Configuration Editor to configure the state synchronization plug-in interface:

1. Access the plug-in configuration as described in *Accessing the Plug-In Configuration* on page 141.




2. Using the field descriptions below, fill in the fields in the State Synchronization and Plug-in Manager areas of the Plug-Ins pane.

***Size of Fail Queue***

- Maximum number of plug-in events that are stored while the communication with a state synchronization plug-in is interrupted.
- Value—Integer in the range 0-2147483647; –1 means unlimited
- Default—5000
- Property name—SyncPlugin.failQueue.maxSize

***Age of Fail Queue***

- Maximum time that plug-in events are stored while the communication with a state synchronization plug-in is interrupted.
- Value—Number of seconds in the range 0-2147483647; –1 means unlimited
- Default— –1
- Property name—SyncPlugin.failQueue.maxTime

***Batch Time***

- Time that the SAE waits for other plug-ins to become ready before starting a synchronization sequence.
- Value—Number of seconds in the range 0-2147483647
- Default—60
- Property name—SyncPlugin.batchTime

***Keep Alive Time***

- Time that the SAE waits after an event before sending a ping to the remote plug-in.
- Value—Number of seconds in the range 0-2147483647
- Default—60
- Property name—SyncPlugin.keepAliveTime

***Number of Threads***

- Number of threads that the SAE maintains for plug-in synchronization.
- Value—Integer in the range 0-2147483647
- Default—5
- Property name—PluginManager.threads

## Configuring Plug-Ins with SDX Admin

This section provides guidelines for configuring plug-ins in the SAE property file with SDX Admin or a text editor. See *Modifying the SAE Property File* on page 57 for information about accessing the property file.

### *Configuring External Plug-Ins*

There are two properties that you define for external plug-ins: objectref and attr. You must define both of these properties. Use the syntax:

```
Plugin.<plug-in instance name>.objectref = <object reference>
Plugin.<plug-in instance name>.attr = <attribute>
```

- plug-in instance name—Name that you choose to identify a particular plug-in instance.
- object reference—Specifies the object reference of the plug-in. You can define the object reference by specifying the absolute path to the IOR file, the corbaloc URL, the COS naming service, or the actual IOR.

  The following example identifies the object reference by its absolute path to the IOR file:

  ```
  Plugin.admissionControl.objectref = file:///var/acp/acp.ior
  ```

- attribute—Comma-separated list of attributes that the SAE sends to the plug-in. See *Fields* on page 154 for a list of attributes.

**NOTE:** Configure only the attributes required. Specifying fewer attributes improves the performance of the SRC network.

### *Configuring Internal and Hosted Plug-Ins*

To define plug-in instances for internal and hosted plug-ins, use the syntax:

```
Plugin.<plug-in instance name>.<property name> = <expression>
```

- plug-in instance name—Name that you choose to identify a particular plug-in instance.
- property name—Each plug-in type has a list of properties that you can define. Use those names to configure properties in the file. Property names are case sensitive. For information about the properties that you can assign, see the section that describes the associated plug-in.
- expression—Sets a value for the property name. For information about the valid values that you can assign to each property, see the section that describes the associated plug-in.

For internal and hosted plug-ins, you must define the class property, which identifies the Java class name of the plug-in. The following example identifies the Java class name for plug-in instance ldapAuth:

```
Plugin.ldapAuth.class = net.juniper.smgt.sae.plugin.LdapAuthenticator
```

For the Java class names of tracking plug-ins, see Table 16 on page 152. For the Java class names of authorization plug-ins, see Table 17 on page 161.

### Defining RADIUS Packets

To create templates that define RADIUS packets in flexible RADIUS accounting and authentication plug-ins, use the syntax:

```
RadiusPacket.<template instance name>. <packet-type>.<id>[.type] =
<expression>
```

- template instance name—Name that you choose to identify the template.
- packet-type—Assign one of the values described in Table 18 on page 177.
- id[.type]—Identifies a RADIUS attribute; use as described in *Property* on page 179.
- expression—Assigns a value to the RADIUS attribute; use in the same way as described in *Value* on page 180.

### Setting Up the Plug-In Instance to Use a Template

To set up a RADIUS plug-in to use a template, define the template property as follows:

```
Plugin.<plug-in instance name>.template = RadiusPacket.<template instance
name>
```

For example, to use the stdAuth template in the flexRadiusAuth plugin instance:

```
Plugin.flexRadiusAuth.template = RadiusPacket.stdAuth
```

### Configuring Event Publishers

To configure global and default retailer event publishers, use the following syntax:

```
<event publisher>=<list of plug-in instances>
```

- Event publisher—Name of property that identifies the event publisher. See *Configuring Global and Default Retailer Event Publishers* on page 185 for the property names of global and default retailer publishers.
- List of plug-in instances—Comma-separated list of plug-in instances to which you want the publisher to send events.

The following is the default event publisher configuration. It sets the global subscriber tracking and global service tracking publishers to send events to the fileAcct plug-in instance, and sets the default retailer publisher to send events to ldapAuth.

```
#global plug-ins
User.auth.plugins =
User.tracking.plugins = fileAcct
Service.auth.plugins =
Service.tracking.plugins = fileAcct
Subscription.auth.plugins =
# default user authentication
Retailer.auth.plugins = ldapAuth
Interface.tracking.plugins =
# default dhcp authentication
Retailer.dhcpauth.plugins =
```

### *Example: LDAP Authentication Plug-In*

The following LDAP authentication plug-in searches for objects of class inetOrgPerson, where the username is stored as the common name (cn):

```
Plugin.ldapAuthFoo.class =\ com.junipernetworks.ssc.plugin.LdapAuthenticator
Plugin.ldapAuthFoo.method = search
Plugin.ldapAuthFoo.host = 10.1.2.3
Plugin.ldapAuthFoo.bindDN = cn=admin
Plugin.ldapAuthFoo.bindPW = {BASE64}c3Nw
Plugin.ldapAuthFoo.filter = (objectclass=inetOrgPerson)
Plugin.ldapAuthFoo.nameAttr = cn
Plugin.ldapAuthFoo.pwdAttr = userPassword
```

### *Example: Basic RADIUS Accounting Plug-In*

The following example configures the basic RADIUS accounting plug-in. The name of the plug-in instance is radiusAcct-1. It communicates with two peers: peer 0 over port 1813 at address 10.1.2.3 and peer 1 over port 1813 at 10.1.2.4. Load-balancing is set to failover. The RADIUS Calling-Station-Id is not sent to the plug-in.

```
Plugin.radiusAcct-1.class = net.juniper.smgt.sae.plugin.\
RadiusTrackingPluginEventListener
Plugin.radiusAcct-1.loadBalancingMode = failover
Plugin.radiusAcct-1.local.timeout = 10000
Plugin.radiusAcct-1.CallingStationId = no
Plugin.radiusAcct-1.peer.0.remote.address = 10.1.2.3
Plugin.radiusAcct-1.peer.0.remote.port = 1813
Plugin.radiusAcct-1.peer.0.remote.password = secret
Plugin.radiusAcct-1.peer.1.remote.password = {BASE64}c2Vjc
Plugin.radiusAcct-1.peer.1.remote.address = 10.1.2.4
Plugin.radiusAcct-1.peer.1.remote.port = 1813
Plugin.radiusAcct-1.peer.1.remote.password = secret
```