

Customizing Service Implementations

Perform the following tasks to customize service implementations:

1. Writing Scripts for Script Services on page 1
2. Configuring Script Services (SRC CLI) on page 3

Writing Scripts for Script Services

The ScriptService service provider interface (SPI) provides a Java interface that a script service implements. For information about the ScriptService interface and the ServiceSessionInfo interface, see the script service documentation in the SAE core API documentation on the Juniper Networks Web site at <http://www.juniper.net/techpubs/software/management/src/api-index.html>.

The implementation of the ScriptService interface activates the service. The SAE sends authentication and tracking events when it activates, modifies, or deactivates a script service session.

The SAE supports script services written in Java or Jython. For scripts written in Java, you must compile and package the implemented ScriptService to make it available for use by the SAE. A Java implementation can include more than one Java archive (JAR) file.

The SAE synchronizes methods used by the same instance of the ScriptService class. You do not need to provide synchronized implementation of the methods.



NOTE: The script service implementation can be called by different threads at the same time. If your script uses resources that are shared between different service instances, you are responsible for synchronizing access to those resources.

To write a script to be used by a script service:

1. Create a class that provides a default constructor and that implements the ScriptService interface.
2. Manage activation and manipulation of the service session by implementing the following ScriptService methods:
 - activateSession()—Activates the script service session.
 - deactivateSession()—Deactivates the script service session and returns any final accounting data for the script service session.
 - modifySession()—If the counters were reset during the modification, modifies the script service session and returns any accounting data.

These methods are implemented by the script service. They perform the associated action (activate, deactivate, modify) when the SAE calls the method.

3. (Optional) Get information about service sessions by using methods on the `ServiceSessionInfo` interface.
4. (Optional) Provide accounting data, if used, by using the following `ScriptService` method:

`getAccountingData()`—Polls for current accounting data and returns any current accounting data.
5. (Optional) Provide service status information by using the following `ScriptService` method:

`getState()`—Returns session data to be stored persistently on the router. The SAE does not use this data but provides it to the script when a service session is restored after failover.
6. Manage the script service by using the following `ScriptService` methods:
 - `initState()` —Initializes a recovered script service session after a state synchronization.
 - `discarded()`—Provides notification that the service session has been discarded. Service sessions are discarded when the SAE loses connection to a router. A discarded service session continues to exist on the router and is restored after the connection to the router is reestablished by an SAE.

The script service session releases any resources associated with a discarded session, but must not take any action to disrupt the service session.

You can also use the `stopService()` method on the `ServiceSessionInfo` object to stop a service and remove the service from the SAE. For example, consider a script service that monitors a state that it creates outside the SAE. If the script detects that the service is not active, it can stop the service and remove it from the SAE. You could use this type of script service to start a daemon process and monitor the process to make sure that it is alive.



NOTE: The `ScriptService` SPI does not provide access to a router driver.

**Example: Using the
ScriptService SPI in
Jython**

The following example implements the `ScriptService` SPI in Jython.

```
from net.juniper.smgmt.sae.scriptservice import ScriptService
class SampleService(ScriptService):
    def initSessionInfo(self, ssi):
        self.ssi = ssi
    def activateSession(self):
        print "Activating ServiceName %s" % self.ssi.serviceName
    def deactivateSession(self):
        print "Deactivating ServiceName %s" % self.ssi.serviceName
        return None
    def modifySession(self, ssi):
        self.ssi = ssi
        print "Modifying ServiceName %s" % self.ssi.serviceName
        return None
```

```

def getAccountingData(self):
    print "Getting accounting data for ServiceName %s" % self.ssi.serviceName
    return None
def getState(self):
    return None
def initState(self, ssi, state):
    self.ssi = ssi
    pass
def discarded(self):
    pass

```

Example: Using the ScriptService SPI in Java

The following example implements the ScriptService SPI in Java.

```

class SampleService implements ScriptService {
    private ServiceSessionInfo ssi;
    public SampleService() { }
    public void initSessionInfo(ServiceSessionInfo ssi) {
        this.ssi = ssi;
    }
    public void activateSession() {
        System.out.println(" Activating ServiceName " +ssi.getServiceName());
    }
    public AccountingData deactivateSession() {
        System.out.println(" Deactivating ServiceName " +ssi.getServiceName());
        return null;
    }
    public AccountingData modifySessionSession(ServiceSessionInfo ssi) {
        this.ssi = ssi;
        System.out.println(" Modifying ServiceName " +ssi.getServiceName());
        return null;
    }
    public AccountingData getAccountingData() {
        System.out.println(" Getting accounting data for ServiceName "
+ssi.getServiceName());
        return null;
    }
    public byte[] getState() {
        return null;
    }
    public initState(ServiceSessionInfo ssi, byte[] state) {
        this.ssi = ssi;
    }
    public void discarded() {
    }
}

```

Configuring Script Services (SRC CLI)

Before you configure a script service, make sure that you know the location of the script file that the service will reference.

Use the following configuration statements to configure a script service in the global service scope:

```

services global service name script {
    script-type (url | python | java-class | java-archive);
    class-name class-name;
    file file ;
}

```

```

    filename filename;
}

```

Use the following configuration statements to configure a script service in a service scope:

```

services scope name service name script {
    script-type (url | python | java-class | java-archive);
    class-name class-name;
    file file ;
    filename filename;
}

```

To configure a script service:

1. Configure a normal service, but set the service type to **script**. See Adding a Normal Service (SRC CLI) .
2. From configuration mode, enter the service script configuration. In this sample procedure, the service called scriptService is configured in the scope configuration.

```

user@host# edit services scope script service scriptService script

```

3. Configure the type of script that the script service uses.

```

[edit services scope script service scriptService script]
user@host# set script-type (url | python | java-class | java-archive)

```

4. For Java class and Python script services, configure the name of the class that implements the script service.

```

[edit services scope script service scriptService script]
user@host# set class-name class-name

```

5. Script services of the url script type are written in Java and provided as a Java archive (.jar) file. Configure the URL that specifies the location of the *jar* file containing the script service implementation. For other script types, you must load the script service implementation.

```

[edit services scope script service scriptService script]
user@host# set file file

```

6. For Java class, and Java archive, and Python script services, load the script service implementation by specifying the filename of the local file containing the script service implementation. This file is the uncompiled Python source code or the compiled result of the Java script service (binary *.class* or *.jar* file).

```

[edit services scope script service scriptService script]
user@host# set filename filename

```

7. (Optional) Verify your configuration.

```

[edit services scope script service scriptService script]
user@host# show

```

```
script-type url;  
class-name net.juniper.smgmt.script.Service;  
file http://some-server/some-path/script-service.jar;
```

- Related Topics**
- Overview of SRC Script Services
 - Configuring Script Services (C-Web Interface)
 - Defining Attributes for Service Activation (SRC CLI)

Published: 2009-09-22