Oracle® Communications IP Service Activator

SDK Service Cartridge Developer Guide

Release 7.5
F59549-01
September 2022

Contents

Preface

This guide explains how to use the Oracle Communications IP Service Activator Software Development Kit (SDK) to create service cartridges.

Audience

This guide is intended for system developers using the SDK toolset to develop service cartridges.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1 Overview

This chapter provides a brief overview of the concepts involved in creating service cartridges with the Oracle Communications IP Service Activator Software Development Kit (SDK).

Developing Service Cartridges with the SDK

A base cartridge provides a framework to allow the Network Processor to perform basic communication functions with a device. These functions include logging in and out of the device, sending commands or configlets, performing audits, and interpreting responses from the device as successes, warnings, or failures. Refer to *IP Service Activator SDK Base Cartridge Developer Guide* for details.

Base cartridges do not contain implementations of services. Additional services targeting specific vendor device types are added through integrated service cartridges.

Configuration policies are implemented in conjunction with supporting service cartridges. For additional information on creating configuration policies, refer to *IP Service Activator SDK Configuration Policy Extension Developer Guide*.

Core Cartridges

IP Service Activator legacy cartridges, known as core cartridges, included the functions provided by both a base and service cartridges all in the same package.

The base cartridge with separate related service cartridges is the preferred method of supporting new services to maximize scalability and flexibility.

Vendor Cartridges

A base or core cartridge can be combined with a number of service cartridges to create a vendor cartridge, which contains the functionality to connect to a specific device type, and apply the services provided by the service cartridges.

2

Building Service Cartridges

This chapter discusses service cartridge concepts and explains how to use the Oracle Communications IP Service Activator Software Development Kit (SDK) to create service cartridges, and integrate them with configuration policies.

References are made to sample service cartridge components packaged with the SDK.

Refer to the "Service Cartridge Generation Properties" for details on all properties in the properties file.

Table 2-1 lists the directory placeholders used in this guide.

Table 2-1 Directory Placeholders

Placeholder	Description
SDK_home	The directory in which the SDK is installed.
Service_Activator_home	The directory to which IP Service Activator is deployed. Typically **C:Program Files\Oracle Communications\IP Service Activator**

This guide assumes:

- That the required versions of additional third party tools to support the SDK are installed correctly.
- That you have set up the required environment variables to support the SDK functions.

For details on installing the SDK and the third party tool versions, refer to *IP Service Activator SDK Installation and Setup Guide*.

Building a Service Cartridge

This section lists the steps required to build an XQuery-based service cartridge and a Java-based service cartridge. A brief introduction to these steps follows. After that, each step is covered in detail.

The steps to build an XQuery-based service cartridge are:

- Creating a Service Cartridge Source Directory and Skeleton Properties File
- Defining the Service and Customizing the Skeleton Properties File
- Generating the Cartridge Source Files
- Customizing the Service Cartridge Source Files
- Compiling the Service Cartridge
- Performing Unit Tests
- Performing End-to-End Tests

The steps to build a Java-based service cartridge are:

- Copying one of the sets of sample source files to support your new development
- Customizing the Service Cartridge Source Files
- Compiling the Service Cartridge
- Performing Unit Tests
- Performing End-to-End Tests

Note:

Service cartridges which use java-based transforms do not make use of the **skeleton.properties** file, or the source code generation capabilities of the SDK. Instead, sample source code is provided for you to base your project on.

About Java-Based Service Cartridges

There are some special considerations to consider when creating service cartridges which use Java-based transforms as opposed to XQuery-based transforms. They do not make use of the **skeleton.properties** file, or the source code generation capabilities of the SDK. Instead, sample source code is provided for you to base your project on.

The parts of the procedures in this chapter dealing with XQuery, the **skeleton.properties** file, and generating and customizing XQuery cartridge source files in particular, do not directly apply to Java-based service cartridge development.

Creating a Service Cartridge Source Directory and Skeleton Properties File

To begin creating an XQuery-based service cartridge, you will need to establish a directory structure for the source files, and create the skeleton properties file that will be used to generate the starting source files.

Defining the Service and Customizing the Properties File

This step involves determining the specific details of the service to be created and specifying the information needed to apply the desired configuration to the device.

You will need to select at least one modeled service or configuration policy to be implemented by the service cartridge. As a starting point, you can specify one modeled service or one configuration policy as a subscription in the cartridge skeleton properties file. This file is then used by the SDK service cartridge generator tool to generate a set of source files as a starting point for your service cartridge. The generated source files can be customized to add any additional subscriptions if the service cartridge will implement more than one modeled service and/or more than one configuration policy.

Each cartridge you create with the SDK will have a skeleton properties file. In this file, you will customize the property values that control the generation of the cartridge source files. For complete details on all properties, refer to "Service Cartridge Generation Properties".

Generating the Cartridge Source Files

This step uses the SDK tools to read the skeleton properties file and create the skeleton cartridge source files.

Customizing the Cartridge Source Files

This step is where the majority of your development work takes place for XQuery-based service cartridges. Some of the key cartridge components you need to create are:

- Device Model (DM) schema definition
- Service Model (SM) to DM transform
- Annotated DM to CLI transform
- Message (success/warning/error) pattern definitions

There are many other source file components you may need to create or modify including files that support audit services, options, capabilities, and pre- and post-checks. These are described later in this chapter.

Compiling and Packaging the Cartridge

Use the included script to compile and package the cartridge.

Performing Unit Tests

Unit tests are provided as part of the generated source files.

Performing End-to-End Tests

Deploy the cartridge into a test IP Service Activator environment as an extension to a base or core cartridge to perform end-to-end testing.

About the Provided Sample Service Cartridge

The SDK includes XQuery-based sample service cartridges and Java-based sample service cartridges. This section describes the purpose and components of the sample service cartridges, and gives details about each sample cartridge.

Purpose of the Provided Service Cartridge Samples

You can use the samples in a variety of ways:

- You can inspect generated source files to see how a simple, working, service cartridge is constructed.
- You can complete, compile and package the pre-generated, customized sample source files into a working service cartridge and deploy it in a test system as an extension to the cisco base cartridge sample.

- You can take a copy of a provided skeleton properties file, relocate and rename it, and use it as the starting point to generate your own skeleton cartridge source files.

Components of the XQuery-Based Sample Service Cartridge

Included with each Cisco XQuery-based sample service cartridge are:

- A **skeleton.properties** file: used to generate the source files for its sample service cartridge. See "Generating the Cartridge Source Files".
- Pre-generated, customized sample service cartridge source files: used to demonstrate the edits required to the generated source files to produce a working service cartridge.

For example, the source files provided with the ciscoBanner sample include:

- **...\audit\auditTemplate.xml**
- **...\messages\successMessages.xml**, **errorMessages.xml**, **warningMessags.xml**
- **...\schema\devicemodel.xsd**
- **...\test\TransformUnitTests.java**
- **...\transforms\sm2dm.xq**, **annotated-dm2cli.xq**, **dm-validation.xq**
- **...\xquerylib\ dm2cli-banner.xq, dm2cli-cisco.xq**, **dm-version.xq**, **sm2dm-banner.xq**

Note:

The other XQuery-based service cartridge sample files are organized in a similar fashion.

Sample XQuery-Based Service Cartridges

The XQuery-based sample service cartridges are:

- ciscoBanner
- ciscoMartini
- ciscoStaticRoute

Sample ciscoBanner Service Cartridge

The ciscoBanner sample cartridge illustrates the following concepts:

- Filling out the skeleton properties for a service cartridge
- Subscribing to configuration policies
- Managing basic capabilities for configuration policies
- Architecting the device model schema
- Creating the cartridge Extension registry

- Transformations (SM2DM and annotatedDM-CLI) that appropriately pass on the various IDs
- Device Model instance validation
- Providing data so the Network Processor can recognize and correctly act on router responses
- Providing command information to correctly enable NP audits for these services
- Use of XQuery modules to improve compartmentalization, maintainability and readability of XQuery code

Note:

These details apply in a similar fashion to the other XQuery-based samples (ciscoMartini and ciscoStaticRoute).

The ciscoBanner sample service cartridge implements the service modelled by the bannerSample sample configuration policy. The ciscoBanner sample contains a schema for a consolidated banner configuration policy that allows you to configure a sequence of one or more of five possible banner commands. It is assumed that only one occurrence of each banner type can exist on a device.

For details on the bannerSample configuration policy and its parameters and options, refer to *IP Service Activator SDK Configuration Policy Extension Developer Guide*.

Completing the Sample

To complete the sample:

1. Do one of the following:
   - Copy the provided files over their generated counterparts

     or
   - Edit the generated files.

You will need to create the source files for the corresponding configuration policies which implement the sample cisco services. See *IP Service Activator SDK Configuration Policy Extension Developer Guide* for details.

Sample Properties File

The sample skeleton properties file that is used to create the source files for the ciscoBanner sample service cartridge is called:

SDK_home\samples\serviceCartridge\ciscoBanner\skeleton.properties**

This properties file is pre-populated with the information needed to construct the source files for the ciscoBanner sample service cartridge.

Some of the generated source files will need to either be edited or overwritten with the provided source files.

As you read through the service cartridge creation steps, instructions are given on how to use the ciscoBanner sample to test some of the SDK tools and commands.

Refer to Service Cartridge Generation Properties for details on all the properties implemented in the sample properties files which create the source files for the samples.

Sample ciscoMartini Service Cartridge

The ciscoMartini sample service cartridge implements a Martini layer two VPN service.

This service cartridge is written using XQuery-based transforms and illustrates how to implement a modeled IP Service Activator service using a service cartridge.

The provided set of pre-generated source files in the sample are based on the output of the supplied skeleton.properties file, but are pre-configured with some of the specific service configuration knowledge required for the service.

Key concepts illustrated by the sample ciscoMartini service cartridge are:

- Filling out the skeleton.properties for a modeled service cartridge
- Subscribing to a modeled service
- Managing capabilities for a modeled service
- Transformations (SM2DM and annotatedDM-CLI) that appropriately pass on various IDs

Note:

This service cartridge does *not* implement a full-functional, or completed Martini service. It is intended only to demonstrate some of the concepts required when implementing such a service in a service cartridge.

Sample ciscoStaticRoute Service Cartridge

The ciscoStaticRoute implements the service modelled by the sample static route configuration policy. It allows you to configure one or more static routes.

The ciscoStaticRoute sample cartridge illustrates the same concepts listed in "Sample ciscoBanner Service Cartridge" plus the following concepts:

- Specification of options schema
- Use of options to influence transformations
- Use of type-1 pre-check (to ensure a new static route service does not have a pre-existing conflicting static route on the router)
- Use of type-2A pre-check (to ensure the next-hop address in the static route is reachable from the device - traps case when the network of next-hop is not reachable)
- Use of type-2B pre-check (to ensure the next-hop address in the static route is reachable from the device - traps case when subnet is not reachable)
- Use of post-check (to ensure that a provisioned static route has correctly updated the routing table)

For details on the ciscoStaticRoute configuration policy and its parameters and options, refer to *IP Service Activator SDK Configuration Policy Extension Developer Guide*.

Sample Java-Based Service Cartridges

The Java-based service cartridge sample is:

- ciscoBannerJava

Sample ciscoBannerJava Service Cartridge

The ciscoBannerJava sample service cartridge re-implements the bannerSample configuration policy for Cisco IOS and is similar to the ciscoBanner sample, except that it uses Java for model transformations instead of XQuery.

The provided sample is pre-compiled. Service cartridges with Java-based transforms do not make use of the generated XQuery source files, so no customization step for these files is needed.

The ciscoBannerJava sample implements the service modelled by the bannerSample configuration policy. The ciscoBannerJava sample contains a schema for a consolidated banner configuration policy that allows you to configure a sequence of one or more of five possible banner commands. It is assumed that only one occurrence of each banner type can exist on a device.

Key concepts illustrated by the ciscoBannerJava sample service cartridge are:

- File and directory structure for Java-based cartridges (since skeleton generation cannot be used)
- Ant build scripting
- Synonyms
- Extension registry
- Registry customization in the field
- Use of options in Java transforms
- Java-based SM-DM and annotatedDM-CLI transforms
- Device Model upgrade transformation
- Unit test harnesses for Java cartridges
- Java-based Device Model instance validation

For details on the bannerSample configuration policy and its parameters and options, refer to *IP Service Activator SDK Configuration Policy Extension Developer Guide*.

Creating a Service Cartridge Source Directory and Skeleton Properties File

To create your own XQuery-based service cartridge, you will need to establish a directory structure for the source files, and create a skeleton properties file to generate the starting source files. (For Java-based service cartridges, use one of the provides samples as a starting point and add your own Java code from there. None of the steps around creating and customizing XQuery files are required.)

Note:

When deciding upon a directory structure for a new base cartridge or service cartridge care must be taken to choose a unique base directory name. If the path of a file in the new cartridge is the same as the path of a file in a deployed cartridge, undesirable behavior could occur.

The simplest method is to copy one of the sample **skeleton.properties** files and then edit it for your own use.

To copy and edit the sample **skeleton.properties** files:

1. Create a meaningful name for your new service cartridge. This name will be referred to as *this_service_name*.
2. Create a new directory for your cartridge source files:

   *SDK_home***\serviceCartridges***this_service_name*
3. Copy a skeleton.properties file from one of the sample service cartridges into the new directory:

   ```
   copy
   <SDKHome>\samples\serviceCartridge\ciscoBanner\skeleton.properties
   <SDKHome>\serviceCartridges\this_service_name
   ```
4. Edit your skeleton.properties file to change the sample cartridge name to *target_service_name* in the following entries:

   ```
   ## service cartridge name
     sdk_global_cartridgeName=this_service_name
   . . .
   ## packaging structure

   sdk_global_package=com.metasolv.serviceactivator.cartridges.this_service_name
   ```

Defining the Service and Customizing the Skeleton Properties File

This step involves determining the specific details of the service to be created and specifying the information needed to apply the desired configuration to the device.

The service may require application across multiple operating systems or device types. In this case, you will need to create multiple service cartridges.

This can affect the definition of the service and/or require separate services to be defined to achieve the desired end goal.

If your service requires an HTML-based GUI input form, you will also need to create a corresponding configuration policy. A single configuration policy to implement a generic type of service may be subscribed to by multiple service cartridges, each implementing the service on a specific vendor's device.

You must edit the properties file to match the requirements of your service cartridge. Assuming you have used one of the supplied sample **skeleton.properties** files as a starting point, you will have to edit or remove properties that are not applicable to your service cartridge, and otherwise supply appropriate values for properties for your particular needs. Refer to Service Cartridge Generation Properties for details on the properties in the skeleton properties file.

Generating the Cartridge Source Files

The SDK provides a tool for generating the service cartridge source files from the skeleton properties file. Once the source files are generated, you will need to edit them to complete your service cartridge.

Note:

Ensure that you save copies of any cartridge source files you alter prior to re-generating from the **skeleton.properties** file to ensure that you do not lose customization work. Alternatively, modify the **skeleton.properties** file so that a new target directory name is used. In either case, you will need to manually merge any alterations you made in the previous iteration if you want those changes to persist.

Generating the Sample ciscoBanner Service Cartridge Source Files

The file *SDK_home***\samples\serviceCartridge\ciscoBanner\skeleton.properties** is fully populated and can be used to construct the skeleton source files for the sample ciscoBanner service cartridge. You can use these files for reference, or as a starting point for your own service cartridge. The generated source files do not contain all the necessary modifications to complete the service. See the provided sample source files for the changes needed.

For the ciscoMartini service cartridge, you also have to modify the **Extension.xml** to point to the **default_caps.xml** file instead of the **empty_caps.xml** file.

To generate the ciscoBanner sample service cartridge source files using the data from the skeleton properties file:

1. Set the cartridge version string variable. For example, if the cartridge version is 1.0, on a Windows host, type the command:

   ```
   set VERSION_STRING=1.0
   ```

2. In the SDK directory, do one of the following:

   - Run the included batch file to run the cartridge generator script:

     ```
     gensc samples\serviceCartridge\ciscoBanner\skeleton.properties
     ```

     or

   - Type in the command to run the cartridge generator script:

     ```
     ant -DtemplateType=serviceCartridge -
     DpropFile=SDK_home\samples\serviceCartridge\ciscoBanner\skeleton.properties
     ```

Note:

To use the batch file, you must first add *SDK_home***bin** to your PATH variable where *SDK_home* is the SDK directory.

Result of the Generation Process

The directory structure you created previously (see "Creating a Service Cartridge Source Directory and Skeleton Properties File") has been extended by using the *sdk_global_cartridgeName* value from the skeleton properties file. The cartridge source files generated under *SDK_home***serviceCartridges***sdk_global_cartridgeName*\ include:

- **build.xml**: ant build file to build the service cartridge
- **src\synonyms.xml**: used by the audit process
- **src\...\audit\auditTemplate.xml**: stub file for audit commands
- **src\...\capabilities\empty_caps.xml**: stub file for capabilities information
- **src\...\messages**: contains .xml files with success, error and warning message patterns
- **src\...\options\options.xsd**: stub .xsd file for cartridge options
- **src\...\schema\devicemodel.xsd**: contains the stub service cartridge device model schema
- **src\...\test**: resources for testing the service cartridge
- **src\...\transforms**: transforms including pre- and post-check, SM to DM, annotated DM to CLI, and DM validation
- **src\...\xquerylib**: additional XQueries for DM to CLI, migration, validation, and version checking
- **src\...\cisco\Extension.xml**: identifies the service cartridge instance
- **src\...\cisco\Customization.xml**: can be used to override **Extension.xml**

The skeleton source file generation process also creates a log file is within the logs directory at *SDK_home***logs\generator.log**.

To continue working with the sample ciscoBanner service cartridge, go to "Completing the Sample Service Cartridge Source Files".

Generating Your Own Service Cartridge Source Files

When you create your own service cartridge, the cartridge name and the root folder for the generated source are based on the *sdk_global_cartridgeName* property value in the service cartridge skeleton properties file. It is incorporated into the cartridge source files in place of *sdk_global_cartridgeName* as shown below.

To generate your own sample service cartridge source files using the your customized skeleton properties file:

1. Set the cartridge version string variable. For example, if the cartridge version is 1.0, on a Windows host, type the command:

```
set VERSION_STRING=1.0
```

2. In the SDK directory, do one of the following:
   - Run the included batch file to run the cartridge generator script:

     ```
     gensc \serviceCartridge\sdk_global_cartridgeName\skeleton.properties
     ```

     or
   - Type in the command to run the cartridge generator script:

     ```
     ant -DtemplateType=serviceCartridge -
     DpropFile=SDK_home\serviceCartridge\sdk_global_cartridgeName\skeleton.propertie
     s
     ```

Note:

To use the batch file, you must first add *SDK_home***\bin** to your PATH variable where *SDK_home* is the SDK directory.

Result of the Generation Process

This extends the SDK directory structure in a similar manner to what was described in "Generating the Sample ciscoBanner Service Cartridge Source Files".

Note:

It is possible to use a different name for the skeleton properties file. If you choose to do this, supply the new name instead of skeleton.properties in the ant commands.

Troubleshooting the Service Cartridge Generation

This section discusses where to find information to help you resolve service cartridge generation issues.

Access to jar Files

The service cartridge needs to have access to the XMLbeans jar files. It must get these from the **ipsaSDK/3rdparty/lib** as shown in the setting of **cartridge.build.path** in **ipsaSDK/build/cartridge_build_imports.xml**. The use of this is shown in the ciscoBannerJava sample service cartridge's **build.xml** file.

In addition, if the service cartridge implements an SDK generated configuration policy, then the service cartridge will need access to the configuration policy's jar file as well. As an example, see how the **build.xml** file for ciscoBannerJava references the bannerSample configuration policy jar.

Using an Alternate Directory Structure

If you are not using the standard directory structure to layout all the configuration policies, base cartridges and service cartridges being developed using the SDK, then you must modify the Java sample **build.xml** file to ensure that all instances of **sdkDir** are replaced with valid paths to the respective files. The preferred way to do this is to set **sdkDir** to the top level directory for all the SDK-based artifacts.

Service Cartridge Generator Message Logging

The logging level of the cartridge generator can be controlled by editing the settings in the *SDK_home***\config\logging.properties** file.

The default is to log debug level messages. Output is sent to both stdout and a logging file: *SDK_home***\logs\generator.log**.

For details on troubleshooting property file attributes, see *IP Service Activator SDK Base Cartridge Developer Guide*.

Customizing the Service Cartridge Source Files

When creating your service cartridges, you will need to make appropriate edits to the skeleton source files to support the particular functionality you want to implement for your service.

The key cartridge source components you need to create and/or modify include:

- Device Model (DM) schema definition
- Service Model (SM) to Device Model transform
- Device Model validation
- Annotated Device Model to CLI transform
- Message (success/warning/error) pattern definitions

Device Model Schema Definitions

The service cartridge device model (**deviceModel.xsd**) extends the Network Processor's device model (DM) schema. It adds validation rules so that the new service(s) which the cartridge enables can be fully described between it and the Network Processor's Device Model.

Service Model to Device Model Transform

This XQuery or java source transform transforms the device-independent service model to a device-specific device model.

It is essential that the service model Definition IDs, which identify policy definitions, and Association IDs, which identify the links between defined policies and their target objects and their representative concretes in IP Service Activator, flow through from the service model to the device model.

Device Model Validation

If the cartridge entry <dmValidation> contains a dmValidation entry, the Network Processor will invoke this function to validate the transformed device model. This would capture logical faults as opposed to syntax faults that would be caught by the device model validation using **deviceMode.xsd**.

Annotated DM to CLI Transform

The Network Processor compares the target device model with the last device model that was persisted to the database after the last successful push to the device. The Network Processor annotates the target device model. For each policy object, the annotation includes the smId, a dmId that is generated by the Network Processor and a changeType which indicates whether configuration is being added, deleted or modified on the device.

The data from the annotated device model is transformed into a CLI document which contains the required device specific commands.

Message Pattern Definitions

Success, warning and error message pattern files can be defined for service cartridges in the same way as for base cartridges. For example, device responses for commands sent by a service cartridge are analyzed and patterns are created in the message pattern files for that service cartridge. The difference is that for a base cartridge, the messages files are referenced from the **Registry.xml** file, and for a service cartridge, the messages files are reference from the **Extension.xml** file. For an overview of the concepts behind message files, see *IP Service Activator SDK Developer Overview Guide*.

For complete details on defining message patterns, see *IP Service Activator SDK Base Cartridge Developer Guide*.

Customizing the Service Cartridge Registry

The service cartridge registry identifies the service model definitions to which the service cartridge subscribes and defines the transforms, audits, capabilities, options and messages of the service cartridge.

When the service cartridge is deployed, the service cartridge jar file will be deployed into the serviceCartridges sub-directory, the parent of which is the directory in which a base or core cartridge is already installed. Aspects of the service cartridge registry can be overridden after deployment by using a customization registry file.

The layout of the service cartridge registry file, **Extension.xml**, is defined by the schema **cartridge.xsd**. The service cartridge registry file is the first service cartridge file read by the Network Processor. It defines all of the important entry points into the service cartridge.

When you create a service cartridge, you must customize **Extension.xml** appropriately for your implementation.

The parameters specified in the service cartridge registry include:

- Name: the name of the service cartridge. This must be unique across all service cartridges.
- SmToDm: the service model to device model transform file
- DmValidation: the device model validation file
- DmToCli: the device model to CLI transform file
- DmMigration: the device model migration file. This is used to perform device model migration.
- Success: the success message patterns file. These patterns are used to identify success messages sent by a device.

- Warning: the warning message patterns file. These patterns are used to identify warning messages sent by a device.
- Error: the error message patterns file. These patterns are used to identify error messages sent by a device.
- AuditTemplateFile: the audit template file. It is used by the audit process for devices provisioned using a CLI.
- AuditQueryFile: the audit query file. This file is used by the audit process for devices provisioned using an XML interface. Different audit template files can be specified for different device types and OS versions.
- OptionsEntry: the options file. This file specifies the options file that will be passed to both the SM2DM and DM2CLI XQuery transforms. Different option files can be specified for different device types and OS versions.
- CapsEntry: the capability file. This file specifies the capabilities that are supported by the service cartridge. Different capability files can be specified for different device types and OS versions. Capabilities for different service cartridges are ORed together by the network processor during a policy server caps fetch.
- Subscriptions: specify the parts of the service model that the service cartridge is interested in.

For details on registry operations, refer to *IP Service Activator SDK Developer Overview Guide*.

For information on customizing the base cartridge registry, see *IP Service Activator SDK Base Cartridge Developer Guide*.

Completing the Sample Service Cartridge Source Files

To complete the sample service cartridge source files:

1. Do one of the following:
   - Copy the files provided in *SDK_home*\samples\serviceCartridge\CiscoBanner\ over their counterparts in the generated source directory (*SDK_home***\serviceCartridge\ciscoBanner\**)

     or
   - Edit the generated sample source files to complete their content development.

     The provided files demonstrate the modifications required to complete their content development to produce a working sample service cartridge.

     You can examine the contents of the sample files and by highlighting in some manner (change bars, etc.), you can observe what was added, or modified to complete the sample.

The files to be copied or edited are:

- *SDK_home***\samples\serviceCartridge\ciscoBanner\src\...\audit\auditTemplate.xml**
- *SDK_home***\samples\serviceCartridge\ciscoBanner\src\...\messages\***
- *SDK_home***\samples\serviceCartridge\ciscoBanner\src\...\schema\devicemodel.xsd**
- *SDK_home***\samples\serviceCartridge\ciscoBanner\src\...\transforms\***

Using Options in the CiscoStaticRoute Sample

Given a cartridge which supports a particular service on Cisco devices, here is an example of how to add an option to support a variation in a configuration command for certain device type(s) and OS version(s).

Options.xsd File

In this case, the example adds an option for the ciscostaticRoute sample service cartridge supplied with the SDK to add the parameter permanent to the static route command for certain devices which require this.

To define support for a new boolean option type called **cartridge.ciscoStaticRoute.permanentOption**:

1. Edit the options schema file **options.xsd** based on the generated sample cartridge source file.
2. Add the following statement:

   ```
   . . .
   <xs:element name="cartridge.ciscoStaticRoute.permanentOption"
   type="opt:BooleanValue" minOccurs="0" default="false"/>
   . . .
   ```

   The default value for this new option is false.

Options.xml File

To specify which options apply for the device types and device OS variants that this cartridge supports:

1. Create the **options.xml** file.
2. Create an entry using the data type **cartridge.ciscoStaticRoute.permanentOption**, which was defined in the **options.xsd** file.

The content for the options.xml file is:

```
<?xml version="1.0" encoding="UTF-8"?>
<base:options
xsi:type="CartridgeOptions"
xmlns="http://www.metasolv.com/serviceactivator/cisco/staticroute/options"
xmlns:base="http://www.metasolv.com/serviceactivator/options"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<!-- options for ciscoStaticRoute service cartridge -->
  <cartridge.ciscoStaticRoute.permanentOption>true
</cartridge.ciscoStaticRoute.permanentOption>
<!-- additional options could go here -->
</base:options>
```

Xquerylib\dm2cli-staticroute.xq File

To implement the newly-defined option:

1. Edit the annotated DM to CLI transform as follows:

   ```
     import module namespace options = "options-common-functions" at "resource://
   metasolvcom/metasolv/serviceactivator/networkprocessor/xquerylib/options-
   common.xq";
   . . .
   ```

```
  if
(options:getBooleanOption("cartridge.ciscoStaticRoute.permanentOption",false(
)) = true()) then
    " permanent"
  else ())
. . .
```

The first statement imports a standard library file of common XQuery functions (**options-common.xq**) to provide the ability to recognize and process 'if' clauses.

The if statement triggers the option evaluation and action.

When the annotated DM to CLI transform is run, an XQuery command to read the boolean option **cartridge.ciscoStaticRoute.permanentOption** from the loaded **Extension.xml** file is run.

A check is made to see if there is an options file specified that matches the device type and OS variant for the target device the static route is being configured on.

In this case, there is an entry in the **Extension.xml** file to match devices of type 'Cisco.*' and OS '*.'. The file **options.xml** is specified for devices matching these characteristics. Because of the wild cards, the target device for the static route matches.

Next, the specified options file (**options.xml**) is searched for an entry matching the passed option (**cartridge.ciscoStaticRoute.permanentOption**). The entry exists, and the value associated with it is true.

This true value is returned back up to the if statement in the transform XQuery which initiated the option lookup:

```
  if
(options:getBooleanOption("cartridge.ciscoStaticRoute.permanentOption",false())
= true()) then
    " permanent"
```

and the string " permanent" is added to the output CLI. The static route command for this target device, which is handled by this cartridge, will have **permanent** appended to it.

Extension.xml File

To specify that the cartridge uses options and to give the name of the options file:

1. Edit the Extension.xml file for the cartridge.
2. Supply regular expressions to specify the device types and OS versions that this options file applies to. In this case, it applies to all Cisco device types, and all OS versions.

For example:

```
<options>
  <optionsEntry>
  <optionsFile>com/metasolv/serviceactivator/cartridges/ciscostaticroute/options/
options.xml</optionsFile>
  <appliesTo>
    <deviceTypes useRegex="true">Cisco.*</deviceTypes>
    <osVersions useRegex="true">.*</osVersions>
    </appliesTo>
  </optionsEntry>
</options>
```

Compiling the Service Cartridge

Service cartridge source files are compiled using ant. The compilation process creates the required XML beans for the cartridge and packages them into a .zip file.

> **Note:**
>
> An existing CLASSPATH environment variable may interfere with the CLASSPATH required by the SDK. It is therefore recommended that the CLASSPATH environment variable be unset in the session where the SDK is being used. For example:
>
> ```
> set CLASSPATH=
> ```

Compiling the CiscoBanner Sample Source Files

To compile the ciscoBanner sample service cartridge source files:

1. Set the cartridge version string variable. For example, if the service cartridge version is 1.0, on a Windows host, type the command:

   ```
   set VERSION_STRING=1.0
   ```

2. Enter the following command:

   ```
   ant package -fSDK_Home\serviceCartridges\ciscoBanner\build.xml
   ```

Compiling Your Custom Service Cartridge Source Files

To compile your custom service cartridge source files, once you are done editing them:

1. Set the cartridge version string variable. For example, if the service cartridge version is 1.0, on a Windows host, type the command:

   ```
   set VERSION_STRING=1.0
   ```

2. Enter the following command:

   ```
   ant package -fSDK_Home\serviceCartridges\sdk_global_cartridgeName\build.xml
   ```

This results in the following additions to the service cartridge directory structure:

```
SDK_home\serviceCartridges\sdk_global_cartridgeName\
build.xml
AuditTrailsReports
beansrc
classes
lib
  sdk_global_cartridgeName.jar
  sdk_global_cartridgeNametests.jar
package
  sdk_global_cartridgeName-serviceCartridge-${env.VERSION_STRING}.zip
    sdk_global_cartridgeName-serviceCartridge-${env.VERSION_STRING}.manifest
```

Troubleshooting Service Cartridge Compilation

Compilation problems will be caused by schema or XQuery errors. To debug these problems, load the schema into an XML schema aware editor. This will make it much easier to find and correct problems in the schema.

Manifest File

When a service cartridge is built, a manifest file is created listing all of the files that are packaged into the service cartridge zip file. Installation of the service cartridge places the manifest in the uninstall directory of the IP Service Activator installation.

Implementing pre- and post-checks

Pre- and post-checks provide the ability verify information on a device when the annotated DM to CLI transform executes, before the general configuration is sent. This allows you to confirm that pre-requisites to the configuration are met.

After configuration is sent, you have the opportunity to have a post-check invoked to verify some aspect of the commands that were sent to the device.

For further information on pre- and post-checks, see *IP Service Activator SDK Developer Overview Guide*.

Testing in a Standalone Environment

Test scripts are created as part of the cartridge skeleton generation process. (See "Generating the Cartridge Source Files").

Performing Unit Tests

The unit test is generated with the skeleton service cartridge source files.

To run unit tests:

1. After you have compiled the service cartridge, enter the following command:

   ```
   ant unittests -
   f=SDK_home\serviceCartridges\sdk_global_cartridgeName\build.xml
   ```

This runs tests which are intended to prove that the main transform stages of the cartridge (i.e. service model to device model and annotated device model to CLI) will generate the output documents correctly.

Deploying Service Cartridges

Service cartridges are deployed as extensions to either base or core cartridges.

Core and base cartridges are deployed as jar files, along with their **MIPSA_registry.xml** / **Registry.xml** files, to the following subdirectory:

*Service_Activator_home***\lib\java-lib\cartridges\***vendor*

Service cartridges are placed below this:

*Service_Activator_home***\lib\java-lib\cartridges\vendor\ServiceCartridges**

The following file instructs the Network Processor to either add a specific service cartridge or not:

*Service_Activator_home***\Config\networkProcessor\com\metasolv\serviceactivator\networkprocessor\ServiceCartridges.properties**

If the file does not exist, or if the service cartridge entry is not present in the file, the service cartridge will be added by default.

The **Extension.xml** file of each service cartridge must be placed at the root level in the service cartridge .jar file.

When an ant build is successfully completed from the service cartridge skeleton directory a service cartridge .zip file is produced in the skeleton package directory. The service cartridge can then be installed into IP Service Activator by unzipping it to the *Service_Activator_home* directory. Upon restart of the Network Processor, the service cartridge is loaded. A notification appears in the IP Service Activator client fault pane to confirm that the service cartridge has been loaded.

To deploy the service cartridge in an IP Service Activator environment:

1. Unzip the cartridge file *sdk_global_cartridgeName***-serviceCartridge-${env.VERSION_STRING}.zip** to the runtime environment of the Network Processor (*NetworkProcessor_home*).
2. Restart the Network Processor to load *sdk_global_cartridgeName***.jar**

To observe cartridge loading operation, see the following files:

- *Service_Activator_home***\logs\networkProcessor.log**
- *Service_Activator_home***\AuditTrails\np***sdk_global_cartridgeName***.log**

Verification of Deployment

Once the Network Processor has started, it will raise information faults in the system indicating each cartridge registered. The new service cartridge should be indicated. If this does not happen, check the Network Processor log; it will contain the details on why the cartridge was not loaded. If the log does not indicate the problem, check that the cartridge was deployed to the correct location.

Deploying the Sample Service Cartridge

Once you have compiled the ciscoBanner sample service cartridge, it can be deployed along with the sample configuration policy **bannerSample**. (See *IP Service Activator SDK Configuration Policy Extension Developer Guide* for details on creating the bannerSample sample configuration policy.)

To deploy the sample service cartridge with the sample configuration policy:

1. Deploy the service cartridge as described in "Deploying Service Cartridges".
2. Create and deploy the bannerSample configuration policy following the procedure in *IP Service Activator SDK Configuration Policy Extension Developer Guide*.

Audit Trail Logging

Audit trail logging records the commands sent to devices by the base cartridge, and any service cartridges that extend the services of the base cartridge.

Each Network Processor maintains one current **networkprocessor.log** file and one current audit trail log file per base or core cartridge. Audit trail logging for a service cartridge is written to the audit trail log file for the base or core cartridge that the service cartridge extends. The network processor logging facilities are based on the log4j utility.

For more information on network processor logging, see *IP Service Activator System Administrator's Guide*.

For details on setting audit trail logging properties, see *IP Service Activator SDK Base Cartridge Developer Guide*.

Device Model Upgrades

Once a cartridge is constructed and deployed, it will carry with it a device model version identifier, such as 1.0. If a subsequent release of the cartridge is constructed which involves a non-trivial device model change, then the device model version would be incremented to, for example, 2.0, to distinguish it from the predecessor cartridge.

For further details on device model upgrades, see *IP Service Activator SDK Base Cartridge Developer Guide*.

Audit

Audit functionality is controlled by an audit template and an audit query file. The names of these files are specified in the **Extension.xml** file.

The audit template file is used by the audit process for devices provisioned using a command line interface (CLI).

The audit query file is used by the audit process for devices provisioned using an XML interface. Different audit template and audit query files can be specified for different device types and OS versions.

For complete details on audit, see *IP Service Activator SDK Developer Overview Guide*.

Uninstalling Service Cartridges

Service cartridges are uninstalled using the **uninstallCartridge.sh** script, which resides in the bin directory of the IP Service Activator installation. This script takes the name of the manifest file, which contains a list of all installed service cartridge files, as a parameter., and uses its contents to uninstall the service cartridge. (See "Manifest File".)

You can include the base directory or the IP Service Activator installation as a parameter to the script. If you do not, the script queries the ORCHcore package to locate the base directory of the IP Service Activator installation.

The **uninstallCartridge.sh** script sorts the manifest file in reverse order, then deletes files, and then directories. Only empty directories are removed; this ensures that the script will not remove directories used by other cartridges.

You can use a relative path to specify the manifest file, but it must be relative to the current directory (where you are running the uninstall script from). You can also use an absolute path. To verify that the manifest file is in the directory, use the command "ls<manifest>" using the same value that is provided to the script.

To uninstall a service cartridge:

1. Enter the following command:

   ```
   uninstallCartridge <manifest_file> [<ServiceActivatorHome>] [-k | -v]
   ```

   Use the -k option to leave empty directories. The -v (verbose) option produces extra output from the script.
2. After the service cartridge is uninstalled, restart the Network Processor.

Note:

Uninstalling a cartridge or configuration policy developed using the SDK does not remove the Network Processor's device model entries that reference this cartridge or configuration policy. This information is maintained because it is unknown whether you are uninstalling the cartridge or configuration policy to remove it or to upgrade it.

Removing a Generated Service Cartridge from the SDK

To remove a generated service cartridge from the SDK installation:

1. Delete all contents under and including *SDK_home***\serviceCartridges***sdk_global_cartridgeName*

Uninstalling the SDK

To uninstall the SDK:

1. Delete all contents under *SDK_home*.

A

Service Cartridge Generation Properties

This appendix provides details on the parameters you can configure in the **skeleton.properties** file used to generate service cartridge source files.

This file contains a number of properties that customize the generated service cartridge source.

Property names are of the form sdk_*context*_*type* and are composed of three parts:

- sdk: indicates an SDK variable
- *context*: describes of the context in which the variable applies
- *type*: indicates how the variable is being used, and may imply a restriction on the possible values:
  - If **supported** appears in the *type*, a boolean value should be entered.
  - If **pattern** appears in the *type*, a regular expression (regex) pattern should be entered.
  - If **prompt** appears in the *type*, a device response should be entered in the form of a regex pattern.
  - If **cmd** appears in the *type*, a device specific command should be entered.

Boolean variables are validated to ensure that the values conform to boolean values (**true** or **false**).

Regex patterns are validated to ensure that they can be compiled.

Note:

For certain regular expressions in the **skeleton.properties** file, it maybe necessary to use an escape character to precede certain special characters in order for them to be translated to the generated source code correctly. This is dependent on whether you are using XQuery or Java based transforms.

Table A-1 shows the Audit properties.

Table A-1 Audit Properties

Property	Description	Example
sdk_audit_supported	Command sent to the device to determine if device is supported. This property is optional.	true

Table A-2 shows the naming and packaging properties.

Table A-2   Naming and Packaging Properties

Property	Description	Example
sdk_global_cartridgeName	This is the cartridge name. This variable is used throughout the cartridge code in generating file names and source code variable names. This property is mandatory.	`ciscoBanner`
sdk_global_baseCartridgeName	The base or core cartridge that this service cartridge extends to provide support for a specific service for a specific vendor. This property is mandatory.	`cisco`
sdk_global_cartridgeVersion	This is the cartridge version that is being developed. It is used at run time to verify that a device model is still valid in the event of an upgrade of the cartridge. This property is optional.	`1.0`
sdk_global_package	This is the cartridge path in dotted notation used for packaging. Its value is translated to a directory structure for source code path generation. The value is used in build scripts, java source code and support files. The generated files are placed under *SDK_home***\serviceCartridges***sdk_global_cartridgeName***\src***sdk_global_package* This property is mandatory.	`com.metasolv.serviceactivator.ciscoBanner would become com\metasolv\serviceactivator\ciscoBanner`

Table A-3 shows the device type identification properties used in the JUnit test environment.

Table A-3   Device Type Identification Properties

Property	Description	Example
sdk_global_deviceName	Specify the device name that the base cartridge, that this service cartridge extends, was constructed for. This is used in the sample service model used by the junit tests for this service cartridge. This property is mandatory.	`Cisco`
sdk_global_deviceDescription	Device description. This is used in the sample service model used by the junit tests for this service cartridge. This property is optional.	`Cisco Internetwork Operating System Software IOS (tm) RSP Software (RSP-PV-M), Version 12.2(8)T, RELEASE SOFTWARE (fc2) TAC`
sdk_global_deviceModel	Device model. This is used in the sample service model used by the junit tests for this service cartridge. This property is optional.	`2611`
sdk_global_deviceVersion	Device version. This is used in the sample service model by the junit tests for this service cartridge. This property is optional.	`12.2(11)T8`

Table A-4 shows the Device Model schema properties.

Table A-4 Device Model Schema Properties

Property	Description	Example
sdk_deviceModel_namespace	Target namespace of the device model schema for this service cartridge. This property is mandatory.	--
sdk_deviceModel_namespaceAbbr	Abbreviation of the target namespace of the device model schema for this service cartridge. This is used as a namespace prefix. This property is mandatory.	`dmbanner`
sdk_deviceModel_prefix	A complex type with the name *sdk_deviceModel_prefix*Device which extends BaseDevice will be generated in the deviceModel schema for this service cartridge This property is mandatory.	`CiscoBanner` becomes `CiscoBannerDevice`

Table A-5 shows the subscription properties. Subscription properties manage subscription to a configuration policy or modeled service definition.

Although a service cartridge can subscribe to more than one modeled service definition type and/or more than one configuration policy, the skeleton generator allows you to specify only one subscription. More subscriptions can be added by editing the generated file **Extension.xml** after the file generation step.

Note:

Only one of *sdk_subscription_configPolicy_supported* and *sdk_subscription_serviceDefinition_supported* can be set to **true**.

Table A-5 Subscription Properties

Property	Description	Example
sdk_subscription_configPolicy_supported	Boolean value to indicate whether or not this service cartridge implements a configuration policy. When set to true, the value of *sdk_subscription_configPolicyName* identifies the configuration policy. This property is mandatory.	`True`
sdk_subscription_configPolicyName	Configuration policy subscription. Specify either a core configuration policy content type, or, for configuration policies created using the SDK, specify the configuration policy name in the GUI when the new configuration policy type is created. See the online Help for more detailed information. This property is optional.	`bannerSample`

Table A-5 (Cont.) Subscription Properties

Property	Description	Example
sdk_subscription_serviceDefinition_supported	Boolean value to indicate whether or not this service cartridge implements a modeled service definition. When set to true, the value of *sdk_subscription_serviceDefinitionName* identifies the modeled service definition. This property is mandatory.	`false`
sdk_subscription_serviceDefinitionName	Modeled service definition subscription. Specify a core definition type from: • ParameterSetDefinitionType • MqcDefinitionType • AccessRuleDefinitionType • GenericRuleDefinitionType • ServiceRuleDefinitionType • PolicingRuleDefinitionType • PhbGroupType • IBgpBaseDefinitionType • IBgpNeighbourDefinitionType • VrfTableDefinitionType • MartiniDefinitionType • CccDefinitionType • L2InterfaceCreationDefinitionType • TlsDefinitionType • SAADefinitionType	`ParameterSetDefinitionType`

Valid core configuration policy content types for the *sdk_subscription_configPolicyName* property include the following:

- atmSubInterfaceData
- multicastInterface
- frSubInterfaceData
- multicastVrf
- multicastDevice
- plSerialInterfaceData
- plPosInterfaceData
- multicastBootstrapRouter
- dslInterfaceData
- multicastAutoRp
- loopbackInterfaceData
- vrfRoutePolicy
- ciscoUniversalInterface
- bgpRoutePolicy
- multilinkInterface

- userData
- pppMultilink
- userAuth
- virtualTemplateInterface
- dialerList
- dialerInterface
- ipsecmodule
- ipPools
- stm1Controller
- prefixListEntries
- t3Controller
- staticRoutes
- e1Controller
- banners
- e3Controller
- collectorParameters
- t1Controller
- staticNats
- stm1ChannelizedSerialInterface
- snmpCommunities
- t3ChannelizedSerialInterface
- snmpHosts
- e1ChannelizedSerialInterface
- saveConfig
- e3ChannelizedSerialInterface
- qosCosAttachment
- t1ChannelizedSerialInterface
- juniperQosCosAttachment
- basicRateInterfaceData
- ciscoQosPfcTxPortQueues
- sgbp
- vlanAccessPort
- schedule
- vlanTrunkPort
- atmPvcVcClass
- vlanDefinitions
- atmVcClass

- vlanInterface
- bgpCE
- portCharacteristics
- extendedAcl
- publicIPsec
- customerIPsec
- backUpInterfacePolicy
- vrfCustomNaming
- vrfExportRouteFilter
- rate-limit
- dlswDevice
- hsrp
- dlswEthernetInterface
- vlanSubInterfaceData
- dlswTokenRingInterface
- keyChains

Table A-6 shows the options schema properties.

Table A-6 Options Schema Properties

Property	Description	Example
sdk_options_namespace	Target namespace of the options schema for this service cartridge. This propert is mandatory.	--
sdk_options_namespaceAbbr	Abbreviation of the target namespace of the options schema for this service cartridge. This is used as a namespace prefix. This property is mandatory.	`cisopt`

B

Generated Skeleton Service Cartridge Source Files

This appendix describes generated service cartridge source files.

About the Generated Skeleton Service Cartridge Source Files

The file *SDK_home***\samples\serviceCartridge\ciscoBanner\skeleton.properties** is fully populated and can be used to construct the skeleton source files for the sample ciscoBanner service cartridge. You can use these files for reference, or as a starting point for your own service cartridge. The generated source files do not contain all the necessary modifications to complete the service. See the provided sample source files for the changes needed.

To generate the ciscoBanner sample service cartridge source files using the data from the skeleton properties file:

1. In the *SDK_home* directory, do one of the following:
   - Run the cartridge generator script using the included batch file:

     ```
     gensc samples\serviceCartridge\ciscoBanner\skeleton.properties
     ```

     or
   - Type in the command to run the cartridge generator script:

     ```
     ant -DtemplateType=serviceCartridge -
     DpropFile=SDK_home\samples\serviceCartridge\ciscoBanner\skeleton.properties
     ```

Note:

To use the batch file, you must first add *SDK_home***\bin** to your PATH variable where *SDK_home* is the SDK directory.

This results in the following directory structure, which is a skeleton ciscoBanner service cartridge:

```
SDK_home
  logs
    generator.log
  serviceCartridges
    <sdk_global_cartridgeName>
      build.xml
      src
        synonyms.xml
        <sdk_global_package>(com)
          <sdk_global_package>(.metasolv)
            <sdk_global_package>(..serviceactivator)
              <sdk_global_package>(...cartridges)
                <sdk_global_package>(....ciscobanner)
```

```
audit
  auditTemplate.xml
capabilities
  empty_caps.xml
messages
  errorMessages.xml
  successMessages.xml
  warningMessages.xml
options
  options.xsd
schema
  devicemodel.xsd
test
  models
    upgradeFrom
      sampleDeviceModel.xml
    sampleServiceModel.xml
  DmUpgradeTests.java
  TransformUnitTests.java
transforms
  annotated-dm2cli.xq
  dm2cli-postcheck.xq
  dm2cli-precheck.xq
  dm-validation.xq
  sm2dm.xq
xquerylib
  dm-migration.xq
  DmUpgrade.xq
  dm-validation.xq
  dm-version.xq
Customization.xml
Extension.xml
```

Table B-1 describes the functionality of the component files of the skeleton service cartridge.

Table B-1 Generated Skeleton Service Cartridge Source Files Details

Component File	Description
synonyms.xml	This file can be used to specify audit synonyms for commands delivered by this service cartridge. Audit synonyms can improve the success rate of a device audit, for devices that display some commands differently than how Oracle Communications IP Service Activator sent them.
auditTemplate.xml	This file is an audit template. Audit templates define filter patterns to be applied to commands to identify configuration of interest, to affect their inclusion in the audit report, and to set attributes on the command results, which, when viewed using a style sheet will affect how they are displayed to the IP Service Activator user.

Table B-1 (Cont.) Generated Skeleton Service Cartridge Source Files Details

Component File	Description
Extension.xml	This file is the service cartridge registry file. The service cartridge registry is defined by the schema **cartridge.xsd**. The service cartridge registry file is the first service cartridge file read by the Network Processor. It defines all of the important entry points for the service cartridge. The parameters controlled by the service cartridge registry are: • **name**: defines the name of the service cartridge. This must be unique across all service cartridges. • **smToDmQuery**: defines the service model to device model transform XQuery • **dmValidation**: defines the device model validation XQuery • **dmToCliQuery**: defines the device model to CLI transform XQuery • **dmMigration**: defines the device model migration XQuery. This is used for device model migration. • **success**: defines success messages sent by a device • **warning**: defines warning messages sent by a device • **error**: defines error messages sent by a device • **auditTemplateFile**: used by the audit process for devices provisioned using a CLI. Different audit template files can be specified for different device types and OS versions. • **subscriptions**: specifies the parts of the service model that the service cartridge is interested in. • **auditQueryFile**: used by the audit process for devices provisioned using an XML interface. Different audit template files can be specified for different device types and OS versions. • **optionsEntry**: specifies the options file that is passed to both the SM2DM and DM2CLI XQuery transforms. Different option files can be specified for different device types and OS versions. • **capsEntry**: specifies the capabilities that are supported by the service cartridge. Different capability files can be specified for different device types and OS versions.
Customization.xml	This file is used to customize the registry file. When packaged in the zip file, it will be named *sdk_global_cartridgeName.xml*.
empty_caps.xml	This file is a sample capabilities file. Capabilities provide privileges to the device and its subordinate interfaces to support various policies. The sample, being empty, will provide no capabilities to the device and it subordinate interfaces. The user needs to provide capability entries in order to provide privileges to the device during the IP Service Activator device discovery process.
errorMessages.xml	This file contains error patterns for commands generated by the service cartridge. If the response from the device matches one of the known error patterns, then a fault (Error) is raised against the device itself, all the concretes affected by that transaction are rejected and the partially implemented configuration is rolled back.
warningMessages.xml	This file contains warning patterns (blocking or non-blocking) for commands generated by the service cartridge. If the response from the device matches a non-blocking warning pattern, a fault (Warning) is raised. If the response from the device matches a blocking warning pattern, a fault is raised, and all concretes affected by that transaction are rejected and the partially implemented configuration is rolled back.
successMessages.xml	This file contains success patterns for commands generated by the service cartridge. If the device response (to sending a command) matches a success pattern, or there is no response at all (only a prompt), then the command is considered successful.

Table B-1 (Cont.) Generated Skeleton Service Cartridge Source Files Details

Component File	Description
options.xml	This file is an XML schema file containing the configuration options that are required to define the cartridge jar file.
deviceModel.xsd	This file is used to validate the device model created during the service model to device model transformation. The base_devicemodel is owned by the network processor framework. The deviceModel is owned by the cartridge and can be extended as needed to support various polices and commands.
sampleDeviceModel.xml	This file is a sample device model used by **DMUpgradeTests.java**.
sampleServiceModel.xml	This file is a sample service model used for JUnit testing by **TransformUnitTesting.java**.
TransformUnitTests.java	• method **testBasicServiceModelToDeviceModelTransform**: tests the ability to transform the sampleServiceModel to a proper deviceModel • method **testBasicDeviceModelToCommandDocumentAddTransform**: tests the ability to transform the deviceModel to a proper cliDocument which is adding cmds to the device • method **testBasicDeviceModelToCommandDocumentDeleteTransform**: tests the ability to transform the deviceModel to a proper cliDocument which is deleting cmds from the device
DMUpgradeTests.java	This file is used for testing cartridge upgrade scenarios.
sm2dm.xq	This file contains the XQuery source code that transforms a service model to a device model.
annotated-dm2cli.xq	This file contains the XQuery source code that transforms a device model to a CLI document.
dm2cli-postcheck.xq	This file contains the XQuery source code that performs the post-check functionality which is used by the **annotatedDM2Cli.xq**.
dm2cli-precheck.xq	This file contains the XQuery source code that performs pre-check functionality, which is used by the **annotatedDM2Cli.xq**.
dm-validation.xq	This file contains the XQuery source code providing the ability to raise fault to the system console.
dm-migration.xq	This file contains the XQuery source code used to support device model upgrades.
DmUpgrade.xq	This file contains the XQuery source code used to support executing a DM upgrade if cartridge DM has been enhanced.
dm-version.xq	This file contains the XQuery source code used to identify which cartridge version is in use.
dm-validation.xq	This file contains the XQuery source code used for additional validation of the device model.