Oracle® Communications IP Service Activator

SDK Configuration Policy Extension Developer Guide

Release 7.5
F59547-01
September 2022

ORACLE

Oracle Communications IP Service Activator SDK Configuration Policy Extension Developer Guide, Release 7.5

F59547-01

Contents

Preface

Audience v
Documentation Accessibility v
Diversity and Inclusion v

1 Overview

Developing Configuration Policies with the Software Development Kit 1-1
- Core Cartridges 1-1
- Vendor Cartridges 1-1

2 Building a Configuration Policy

Building a Configuration Policy 2-1
- Creating a Configuration Policy Source Directory and Skeleton Properties File 2-2
- Defining the Configuration Policy and Customizing the Properties File 2-2
- Generating the Configuration Policy Source Files 2-2
- Customizing the Configuration Policy Source Files 2-2
- Compiling and Packaging the Configuration Policy 2-2
- Performing End-to-End Tests 2-2

About the Provided Sample Configuration Policies 2-2
- Components of the Provided Sample Configuration Policies 2-3
- Completing the Sample 2-3
- Purpose of the Provided Sample Configuration Policies 2-3
- Sample Skeleton Properties File 2-3

Sample Configuration Policies 2-4
- Sample bannerSample Configuration Policy 2-4
- Sample staticrouteSample Configuration Policy 2-5

Creating a Configuration Policy Source Directory and Properties File 2-5
Defining the Configuration Policy and Customizing the Properties File 2-6
Generating the Configuration Policy Source Files 2-6
- Generating the bannerSample Configuration Policy Source Files 2-6
  - Result of the Generation Process 2-7

Generating Your Own Configuration Policy Source Files 2-7
Result of the Generation Process 2-8
Troubleshooting Configuration Policy Generation 2-8
Using an Alternate Directory Structure 2-8
Configuration Policy Generator Message Logging 2-8
Customizing the Configuration Policy Source Files 2-8
Completing the Configuration Policy Source Files 2-9
Completing the bannerSample Configuration Policy Schema 2-9
Completing the staticrouteSample Configuration Policy Schema 2-9
Completing Your Own Configuration Policy Schema 2-9
Creating an HTML GUI Form 2-9
GUI Extension Schema API 2-10
Compiling the Configuration Policy 2-11
Compiling the bannerSample Configuration Policy Source Files 2-12
Compiling the staticrouteSample Configuration Policy Source Files 2-12
Compiling Your Own Configuration Policy Source Files 2-12
Troubleshooting Configuration Policy Compiling 2-12
Manifest File 2-13
Creating Interface Management Properties 2-13
Deploying Configuration Policies 2-14
Configuration Policy Version 2-14
NpUpgrade 2-15
Uninstalling Configuration Policies 2-15

3 Working with the IP Service Activator YANG Import Tool

Setting Up the YANG Import Tool 3-1
About the Configuration Policy Property Files 3-2
Importing and Parsing YANG Model Configuration Policies 3-3
Example: Importing and Parsing a Custom Configuration Policy 3-8

A Configuration Policy Generation Properties

Preface

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

Audience

This guide is intended for system developers using IP Service Activator Software Development Kit (SDK) to develop configuration policies.

Before reading this guide, you should have read *IP Service Activator Concepts* and be familiar with IP Service Activator.

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 an overview of how to use the Oracle Communications IP Service Activator Software Development Kit (SDK) to develop configuration policies.

Developing Configuration Policies with the Software Development Kit

A base cartridge provides a framework to allow the Network Processor to perform basic communication functions with a device. For details on creating base cartridges, refer to *IP Service Activator SDK Base Cartridge Developer Guid*e.

Additional services targeting specific vendor device types are added through integrated service cartridges. Refer to *IP Service Activator SDK Service Cartridge Developer Guide* for details.

A configuration policy provides a GUI form and a schema to collect data for a service. Configuration policies require service cartridges to implement the service on specific devices.

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 a Configuration Policy

This chapter discusses configuration policy concepts and explains how to use the Oracle Communications IP Service Activator Software Development Kit (SDK) to create configuration policies. Configuration policies are implemented in a Service Activator installation using service cartridges. Refer to *IP Service Activator SDK Service Cartridge Developer Guide* for details about creating service cartridges.

A brief overview of the steps required to build configuration policies is given, followed by detailed sections explaining all the required activities. Included are steps to try out the procedures on the supplied bannerSample sample configuration policy source files.

> **Note:**
>
> For details about installing the SDK and the required third party tools, plus a detailed overview of all SDK concepts, a discussion of cartridge components, and an explanation of how cartridges integrate with the network processor, refer to *IP Service Activator SDK Developer Overview Guide*.

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 Configuration Policy

This section lists the steps required to build a configuration policy. Following a brief introduction to these steps, each of the steps and all the activities required to execute them are covered in detail.

The steps to build a configuration policy are:

- Creating a Configuration Policy Source Directory and Properties File

- Defining the Configuration Policy and Customizing the Properties File
- Generating the Configuration Policy Source Files
- Customizing the Configuration Policy Source Files
- Compiling the Configuration Policy
- Performing End-to-End Tests

Creating a Configuration Policy Source Directory and Skeleton Properties File

In order to create your own configuration policy, you need to establish a uniquely named directory structure for the source files, and create the skeleton properties file that will be used to generate the starting source files.

Defining the Configuration Policy and Customizing the Properties File

This step involves determining the specific details of the configuration policy to be created and specifying the information needed to apply the desired configuration to the device. Refer to "Configuration Policy Generation Properties" for complete details on the properties.

Generating the Configuration Policy Source Files

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

Customizing the Configuration Policy Source Files

This step is where most of your development effort will be spent. The key configuration policy source components include:

- Schema for configuration policy data: see "Customizing the Configuration Policy Source Files" for details.
- HTML GUI form: see "Creating an HTML GUI Form" for details.

Compiling and Packaging the Configuration Policy

This step uses the SDK tools to compile and package the configuration policy.

Performing End-to-End Tests

Use a service cartridge to deploy the configuration policy into a test Service Activator environment as an extension to a base or core cartridge. See *IP Service Activator SDK Service Cartridge Developer Guide* for information about creating service cartridges.

About the Provided Sample Configuration Policies

This section describes the sample configuration policies provided.

Components of the Provided Sample Configuration Policies

The SDK includes resources for two sample configuration policies: bannerSample and staticrouteSample.

Included with each sample are:

- A **skeleton.properties** file used to generate the source files for the sample configuration policy. For more information, see "Generating the Configuration Policy Source Files".
- A pre-edited sample schema used to demonstrate the edits required by the generated schema to produce a working sample configuration policy.

The provided sample schema is located in:

*SDK_home***samples\configPolicy\bannerSample\...**

The generated sample schema is placed in:

*SDK_home***configPolicies\bannerSample\...**

Completing the Sample

To complete the sample, you can either copy the provided files over their generated counterparts or edit the generated files.

Purpose of the Provided Sample Configuration Policies

The main values provided by the configuration policy sample are:

- You can inspect the generated schema to see how a simple, working, configuration policy is constructed.
- You can complete, compile and package the superceded generated sample source files into a working configuration policy and deploy it in a test system. The bannerSample configuration policy is implemented by the ciscoBanner service cartridge.
- You can take a copy of the provided skeleton properties file, relocate and rename it, and use it as the starting point to generate your own skeleton configuration policy source files.

> **Note:**
>
> Configuration policies are not supported on every router type/OS. For example, not every router supports the banner sub-command banner slip-ppp command. Users must ensure that the router supports the configuration policy commands that they are trying to implement.

Sample Skeleton Properties File

The sample skeleton properties file that is used to create the source files for the bannerSample sample configuration policy is called:

*SDK_home***samples\configPolicy\bannerSample\skeleton.properties**

This properties file is pre-populated with the information needed to construct the starting source files for the bannerSample sample configuration policy.

Some of the generated source files will require editing or you can overwrite them with the provided source files.

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

Refer to "Configuration Policy Generation Properties" for details on all the properties implemented in the sample properties files which create the source files for the sample.

Sample Configuration Policies

The configuration policy samples are:

- BannerSample
- StaticrouteSample

Sample bannerSample Configuration Policy

The bannerSample sample contains a schema for a consolidated banner Configuration Policy that allows you to configure a sequence of one or more of each possible banner command.

Key concepts illustrated by the sample include:

- Filling out the skeleton properties for a configuration policy
- Architecting the configuration policy schema
- Dealing with versioning and upgrades

It is assumed that only one occurrence of each banner type can exist on a device.

The banner types that can occur are:

- Exec: exec process creation banner
- Incoming: incoming terminal line banner
- Login: login message banner
- MOTD: message of the day banner
- Slip-ppp: slip-ppp message banner

Each banner is configured with a message which must begin and end with a delimiting character of your choice. The delimiting character must not otherwise appear in the message. The message may be single-line or multi-line. New line characters are allowed.

An example banner follows:

```
Type:    motd
Message: #Enter text for message of the day here.#
```

A configuration policy requires a service cartridge implementation. For details on the ciscoBanner service cartridge, refer to I*P Service Activator SDK Service Cartridge Developer Guide*.

Sample staticrouteSample Configuration Policy

The staticrouteSample sample contains a schema that allows you to configure one or more static routes. Static routes are routes that cause packets moving between a source and a destination to take a specified path. Such routes are useful for specifying a gateway of last resort to which all unroutable packets will be sent.

Key concepts illustrated by the sample include:

- Filling out the skeleton properties for a configuration policy
- Architecting the configuration policy schema
- Dealing with versioning and upgrades

Static route configuration parameters are:

- Destination Prefix: IP address for the destination network in dotted decimal notation
- Destination Mask: subnet mask for the destination address in dotted decimal notation
- Next Hop IP: next hop IP address in dotted decimal notation
- Exit Interface Name: exit interface name
- Distance Metric: metric value as the distance metric for this route

A configuration policy requires a service cartridge implementation. For details on the ciscoStaticRoute service cartridge and its parameters and options, refer to I*P Service Activator SDK Service Cartridge Developer Guide*.

Creating a Configuration Policy Source Directory and Properties File

To create your own configuration policy, you will need to establish a directory structure for the source files, and create a skeleton properties file to generate the starting source files.

The simplest method is to copy the sample **skeleton.properties** file and edit it for your own use.

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

1. Create a unique name that identifies the configuration policy. This name will be referred to as *this_config_policy*.
2. Create a new directory to hold your source files. For example:

   *SDK_home***configPolicies***this_config_policy*
3. Copy the sample **skeleton.properties** file into your directory:

   ```
   copy SDK_home\samples\configPolicy\bannerSample\skeleton.properties
   SDK_home\configPolicies\this_config_policy
   ```
4. Edit your **skeleton.properties** file and change bannerSample to the *this_config_policy* in the following entries:

   ```
   ## config policy name
   sdk_global_configPolicyName=this_config_policy
   . . .
   ## packaging structure
   ```

```
sdk_global_package=com.metasolv.serviceactivator.configpolicies.this_config_p
olicy
## configuration policy schema
sdk_schema_namespace=http://www.metasolv.com/serviceactivator/
this_config_policy
sdk_schema_namespaceAbbr=bn
sdk_schema_topLevelTag=this_config_policy
sdk_schema_topLevelType=this_config_policy

sdk_xmlbeans_package=com.metasolv.serviceactivator.this_config_policy
```

Defining the Configuration Policy and Customizing the Properties File

You must perform this step before creating your Configuration Policy. This involves determining the specific details of the configuration policy to be created and specifying the information needed to apply the desired configuration to devices.

The configuration policy may require application across multiple operating systems or device types. This can affect the definition of the configuration policy and/or require separate configuration policies to be defined to achieve the desired end-goal.

A single configuration policy to implement a generic type of service may be subscribed to by multiple service cartridges, each implementing it on a specific vendor's device.

Once the configuration policy is designed, use one of the sets of sample skeleton configuration policy source files as a starting point to creating a configuration policy.

Generating the Configuration Policy Source Files

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

Generating the bannerSample Configuration Policy Source Files

To generate the bannerSample configuration policy source files using the data from the sample skeleton properties file:

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

   ```
   set VERSION_STRING=1.0
   ```

2. Do one of the following:

   - In the SDK directory, run the cartridge generator script by running the included batch file:

     ```
     gencp samples\configPolicy\bannerSample\skeleton.properties
     ```

     or

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

     ```
     ant -DtemplateType=configPolicy -
     DpropFile=SDK_home\samples\configPolicy\bannerSample\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 Configuration Policy Source Directory and Properties File") has been extended using the *sdk_global_configPolicyName* value from the skeleton properties file. The source files generated under *SDK_home***configPolicies***sdk_global_configPolicyName*\ include:

- **build.xm**: ant build file to build the configuration policy.
- **src\...\schema\bannerSample.xsd**: contains the configuration policy schema that will be used to validate the data entered through the Service Activator client.
- **src\ConfigPolicyRegistry.xml**: contains registry information for the configuration policy used to integrate it with a service cartridge.

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 configuration policy, go to "Compiling the Configuration Policy".

Generating Your Own Configuration Policy Source Files

When you create your own configuration policy, the configuration policy name and the root folder for the generated source are based on the *sdk_global_configPolicyName* property value in the skeleton properties file. See "Creating a Configuration Policy Source Directory and Properties File" for details. The property value is incorporated into the source files in place of *sdk_global_ConfigPolicyName*.

To generate your own configuration policy source files using your customized skeleton properties file:

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

   ```
   set VERSION_STRING=1.0
   ```

2. Do one of the following:

   - In the SDK directory, run the cartridge generator script by running the included batch file:

     ```
     gencp configPolicies\<sdk_global_configPolicyName>\skeleton.properties
     ```

     or

   - Type in the command to run the generator script:

     ```
     ant -DtemplateType=configPolicy -
     DpropFile=SDK_home\configPolicies\sdk_global_configPolicyName\skeleton.properti
     es
     ```

> **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 is described in "Generating the Configuration Policy 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 Configuration Policy Generation

This section discusses where to find information to help you resolve configuration policy generation issues.

Using an Alternate Directory Structure

If you are not using the standard directory structure to lay out 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.

Configuration Policy Generator Message Logging

The logging level of the configuration policy 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 Configuration Policy Source Files

When creating a configuration policy, you will need to make appropriate edits to the schema to support the particular functionality you want to implement. You will also need to create an HTML GUI form.

Completing the Configuration Policy Source Files

You will need to make appropriate edits to the schema to complete the configuration policy source files.

> **Note:**
>
> The provided schema is located in *SDK_home***samples\configPolicy\bannerSample\src\...\schema\schema.xsd** and the generated schema is located in *SDK_home***configPolicies\bannerSample\src\...\schema\bannerSample.xsd**.

Completing the bannerSample Configuration Policy Schema

To complete the bannerSample configuration policy schema:

1. Do one of the following:
   - Copy the provided schema over the generated schema.

     or
   - Edit the generated schema to complete its content development.

     The provided schema demonstrates the edits required to complete the generated schema to produce a working sample configuration policy.

Completing the staticrouteSample Configuration Policy Schema

To complete the staticrouteSample configuration policy schema:

1. Do one of the following:
   - Copy the provided schema over the generated schema.

     or
   - Edit the generated schema to complete its content development.

     The provided schema demonstrates the edits required to complete the generated schema to produce a working sample configuration policy.

Completing Your Own Configuration Policy Schema

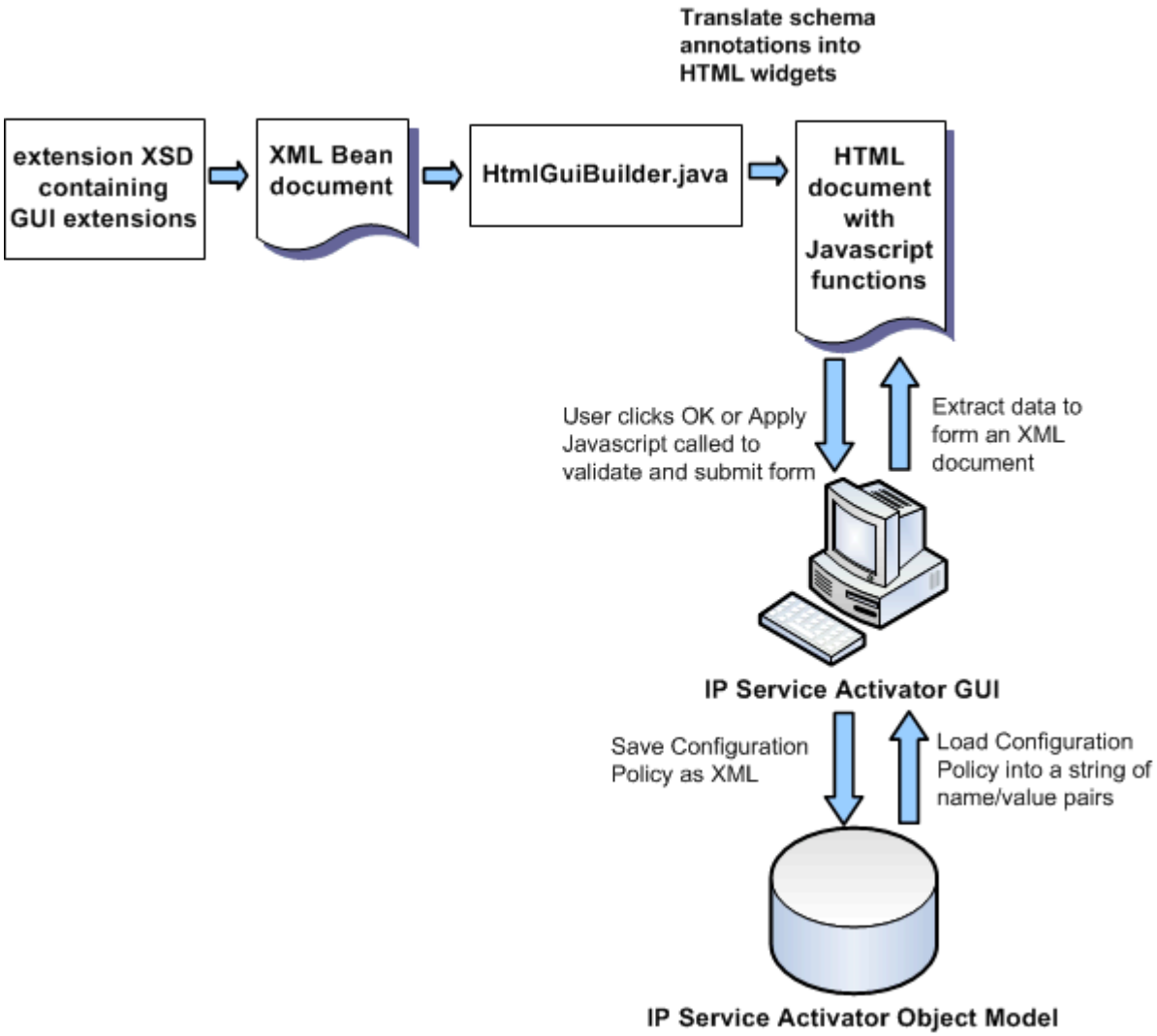
Once you have created your configuration policy schema using the process described in "Generating Your Own Configuration Policy Source Files", edit your schema to add the elements to describe the details of your Configuration Policy.

Creating an HTML GUI Form

The SDK includes a Configuration Policy HTML GUI builder tool. It builds an HTML file from the XML Bean of a configuration policy that can be used in the Service Activator client to create or edit an instance of that configuration policy.

Figure 2-1 illustrates the flow of data between the GUI extensions specified in the extension XSD file (i.e. *SDK_home***schema\cartridge.xsd**)and the object model.

Figure 2-1 Data Flow Between GUI Extensions and Object Model



GUI Extension Schema API

A service model extension schema embeds GUI extension information for all the elements that have an HTML representation. HtmlGuiBuilder relies on the GUI extension information to translate a schema element into an HTML widget, a text area, an input box, a drop-down, a JavaScript import, or other output.

The GUI extension information is defined in **guiextensions.xsd**. The following example shows how to add GUI extension information to any schema element.

```
<xs:schema targetNamespace="http://www.metasolv.com/serviceactivator/banner"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:bn="http://www.metasolv.com/serviceactivator/banner" xmlns:lib="http://
www.metasolv.com/serviceactivator/devicemodel"
xmlns:app="http://www.metasolv.com/serviceactivator/guiextensions"
```

```
xmlns:ipsa="http://www.metasolv.com/serviceactivator/documentation"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="banner" type="bn:Banner" minOccurs="0" maxOccurs="unbounded">
  <xs:annotation>
    <xs:appinfo>
      <app:gui>
        <!-- displayName become the label on the field -->
          <app:displayName>Banner</app:displayName>
          … …
      </app:gui>
    </xs:appinfo>
  </xs:annotation>
</xs:element>
```

Table 2-2 lists the possible schema types and their HTML widget equivalents.

Table 2-2 GUI Extension Schema Element Details

Schema Type	HTML Widget
<app:displayName>	Field label
<app:defaultValue>	Field default value
<xs:enumeration>	Dropdown menu label and value
<app:textarea>	text area
<app:size>	Field size
<app:optional>	Reserved
<xs:choice>	Not supported

Note:

There are three points to note with regard to the schema elements:

- A field can be an input box or a drop-down list.
- If an element has minOccurs="0", and it is a drop-down list, an empty item will be added to the list and selected as the default.
- If an object has multiple elements, an Add button and a Delete will be added to the HTML file, to create or delete an instance of the object.

Compiling the Configuration Policy

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

Note:

A 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 window or where the SDK is being used.

Compiling the bannerSample Configuration Policy Source Files

To compile the bannerSample configuration policy source files:

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

   ```
   set VERSION_STRING=1.0
   ```

2. Compile the bannerSample configuration policy source files with the command:

   ```
   ant package -fSDK_Home\configPolicies\bannerSample\build.xml
   ```

Compiling the staticrouteSample Configuration Policy Source Files

To compile the bannerSample configuration policy source files:

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

   ```
   set VERSION_STRING=1.0
   ```

2. Compile the bannerSample configuration policy source files with the command:

   ```
   ant package -fSDK_Home\configPolicies\staticrouteSample\build.xml
   ```

Compiling Your Own Configuration Policy Source Files

To compile your own configuration policy source files when you are done editing them:

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

   ```
   set VERSION_STRING=1.0
   ```

2. Compile the configuration policy source files with the command:

   ```
   ant package -fSDK_Home\configPolicies\sdk_global_configPolicyName\build.xml
   ```

This will result in the following additions to the Configuration Policy directory structure:

```
SDK_home\configPolicies\sdk_global_configPolicyName
build.xml
beansrc
classes
html
lib
  sdk_global_configPolicyName.jar
package
  sdk_global_configPolicyName-configPolicy-${env.VERSION_STRING}.zip
```

Troubleshooting Configuration Policy Compiling

Compilation problems will be caused by schema 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 Configuration Policy is built, a manifest file is created listing all of the files that are packaged into the Configuration Policy zip file. Installation of the Configuration Policy places the manifest in the uninstall directory of the IP Service Activator installation.

Creating Interface Management Properties

In IP Service Activator the following has been implemented for configuration policies related to interface or sub-interface creation and decoration.

Whenever an interface or sub-interface is created or decorated through IP Service Activator, if the user specifies the IP address and netmask in the HTML page, the user can see the IP address and netmask in the interface properties immediately. Both IPv4 and IPv6 address formats are supported.

For configuration policies shipped with IP Service Activator, the location of ipaddress and ipnetmask is fixed in the **subinterface.xsd**.

The path of the elements is as follows:

- ipaddress: **root Node/primaryAddress/ipAddress**
- ipNetMask: **root Node/primaryAddress/ipNetMask**

If you are using the SDK to create your own configuration policies related to interface or sub-interface creation and decoration, you may wish to change the element names (other than ipaddress and ipnetmask).

To change the element names:

1. Implement a new mappings tag.

   The tag has the following constraints:

   - The tag must be present in your XSD file.
   - The tag is always related to a target. The only valid target is **Interface**.

The layout of the tag is:

```
 <xs:annotation>
        <xs:appinfo>
            <model:mappings  target="Interface">
                <model:mapping direction="export">

<model:source_path>primaryAddress/ipAddress</model:source_path>

<model:target_attribute>ipAddress</model:target_attribute>
                </model:mapping>
                <model:mapping direction="export">

<model:source_path>primaryAddress/ipNetMask</model:source_path>

<model:target_attribute>ipNetMask</model:target_attribute>
                </model:mapping>
            </model:mappings>
        </xs:appinfo>
    </xs:annotation>
```

Under mappings, you can specify a list of mappings. Each mapping contains a direction attribute. The supported direction is **export**.

Each mapping has source_path and target_attribute as sub-tags. The source_path can tell the path of the attribute from the root.

The target_attribute is fixed. The only supported target attributes are ipAddress and ipNetMask.

Deploying Configuration Policies

The generated SDK configuration policy zip file needs to be deployed on a Service Activator Network Processor host as well as on a Service Activator client host.

To deploy the configuration policy to a Network Processor host:

1. Unzip the *sdk_global_configPolicyName***-configPolicy-${env.VERSION_STRING}.zip** file to the runtime environment of the Network Processor in the *Service_Activator_home* directory. The configuration policy schema, for example **bannerSample.xsd**, is installed in *Service_Activator_home***\Config\schema**.
2. If the Network Processor is already running, restarted it to load *sdk_global_configPolicyName*.jar.

To deploy the configuration policy to the policy server host:

1. Unzip the *sdk_global_configPolicyName***-configPolicy-${env.VERSION_STRING}.zip** file to the *Service_Activator_Home* directory.
2. Using the IP Service Activator client, right-click the **Domain** object and select **Properties**.

   The Properties dialog appears.
3. Select **Setup**.
4. Click **Browse** and select the *sdk_global_configPolicyName***.policy** file.
5. Click **OK**.
6. Click **Load** and commit.

The configuration policy is now deployed.

Configuration Policy Version

Once the configuration policy is deployed, any newer version of the configuration policy which includes schema edits should be built with another version number.

To up-version the configuration policy:

1. Modify the version value in **ConfigPolicyRegistry.xml** to supply a service model XQuery transform referenced from the <upgrade> tag in **ConfigPolicyRegistry.xml**.
2. Rebuild the package.

For example, with version 1.0 of a configuration policy deployed, a configuration policy concrete may be created. Now version 2.0 is built and deployed and the Network Processor is restarted. At the time that modifications to the configuration policy

concrete are committed, the Network Processor will compare the currently registered version 2.0 with version 1.0 in the previously saved service model and detect a version mismatch. An upgrade exception will be thrown causing a fault to be raised and the device state to change to Intervention Required. NpUpgrade should be used by an administrator to perform the required service model upgrade transforms.

NpUpgrade

The SDK includes the file **config_policy_registration.xs**d under *SDK_home***schema**. NpUpgrade calls the XQuery pointed to by the <upgrade> tag in the **ConfigPolicyRegistration.xml** file. The purpose of the XQuery is to upgrade the configuration policy data from the old version to the new version inside the service model document in the database. An XQuery is provided with each sample (e.g. **bannerSample_upgrade.xq**).

The **ConfigPolicyUpgradeTest.java** file runs the **XXX_upgrade.xq** XQuery on the **XXX_invalid.xml** and **XXX_valid.xml** files. These files can be modified and run to test variations of the xquery and xml input. You can build and run these tests using ant unitTests from the **ipsaSDK/configPolicies/XXX** folder. The result of running the test goes into file reports/TEST---.xml underneath this folder.

The sample XQuery for banners (**bannerSample_upgrade.xq**) does the following:

- Replaces any <bannerType> elements that have the text `tmotd` with the text `motd`
- Removes any <bogus> or <obsolete> elements
- Keeps all other elements as is

The sample XQuery for static routes (**staticrouteSample_upgrade.xq**) does the following:

- Replaces any <ipa> element found with an <ip> element with the same content
- Removes any <bogus> or <obsolete> elements
- Keeps all other elements as is

Uninstalling Configuration Policies

Configuration policies 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 configuration policy files, as a parameter, and uses its contents to uninstall the configuration policy. (See "Troubleshooting Configuration Policy Compiling".)

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.

> **Note:**
>
> you can use a relative path to specify the manifest file, but it must be relative to the current directory (i.e. 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 configuration policies:

1. Use the following command:

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

   Use the -k option to leave empty directories. The -v (verbose) option produces extra output from the script.
2. Once uninstallation is complete, 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.

To uninstall a configuration policy HTML GUI form from a Service Activator GUI host:

1. Delete the configuration policy HTML file from the *Service_Activator_home* directory.
2. Edit *Service_Activator_home***\Config\ ConfigurationPolicy.cfg** to delete the policytypeReference name and reference to the configuration policy html file.

To remove a generated configuration policy from the SDK installation:

1. Delete all contents under *SDK_home***\configPolicies***sdk_global_configPolicyName*

To uninstall the SDK:

1. Delete all contents under *SDK_home*.

3

Working with the IP Service Activator YANG Import Tool

YANG is a data modeling language used to model configuration and state data manipulated by the Network Configuration Protocol (NETCONF). The IP Service Activator YANG import tool enables you to import configuration policies that use a device-specific YANG model. The IP Service Activator YANG import tool parses the YANG model and requires no coding in IP Service Activation.

Setting Up the YANG Import Tool

To set up the IP Service Activator YANG import tool:

1. Install IP Service Activator version 7.3.4.2.0 or later.

   When installing IP Service Activator, on the Installation Type page, select to install **All Components**, which is the default value. If you do not intend to install all components, ensure that you select:

   - The YANG component
   - The NETCONF Base Cartridge

   See *IP Service Activator Installation Guide* for more information about installing IP Service Activator.

2. Launch IP Service Activator.
3. Enable NETCONF on the appropriate devices.

   To enable NETCONF on a device:

   a. From the **Topology** tab, right-click a device and select **Properties**.

   b. Select **Security**.

      The Security page appears.

   c. In the **Access Style** field, select **NETCONF**.

   d. Click **OK**.

      NETCONF creates a secure SSH v2 connection to the device and sends XML messages that you can review in the **AuditTrails** folder.

      For example:

```
2016-12-15 20:55:19|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?><hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<capabilities>
<capability>urn:ietf:params:netconf:base:1.0</capability>
<capability>urn:ietf:params:netconf:capability:candidate:1.0</capability>
</capabilities>
</hello>]]>]]>
2016-12-15 20:55:19|10.156.68.176|
<rpc>
```

```
<get-configuration format="text">
</get-configuration>
</rpc>]]>]]>
2016-12-15 20:55:19|10.156.68.176|
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
<close-session/>
</rpc>]]>]]>
```

4. If the device was previously defined with the Access Style **SSH** or **TACACS+**, do the following:
   a. Right-click the device and select Unmanage Device.
   b. Commit the change.
   c. Rediscover the device.
   d. On the Device Properties Management page, click Manage.
   e. Commit the change.

About the Configuration Policy Property Files

You use the configuration policy property files to define how to access the information in the YANG model. A configuration policy property file must specify the path to the firewall policer section:

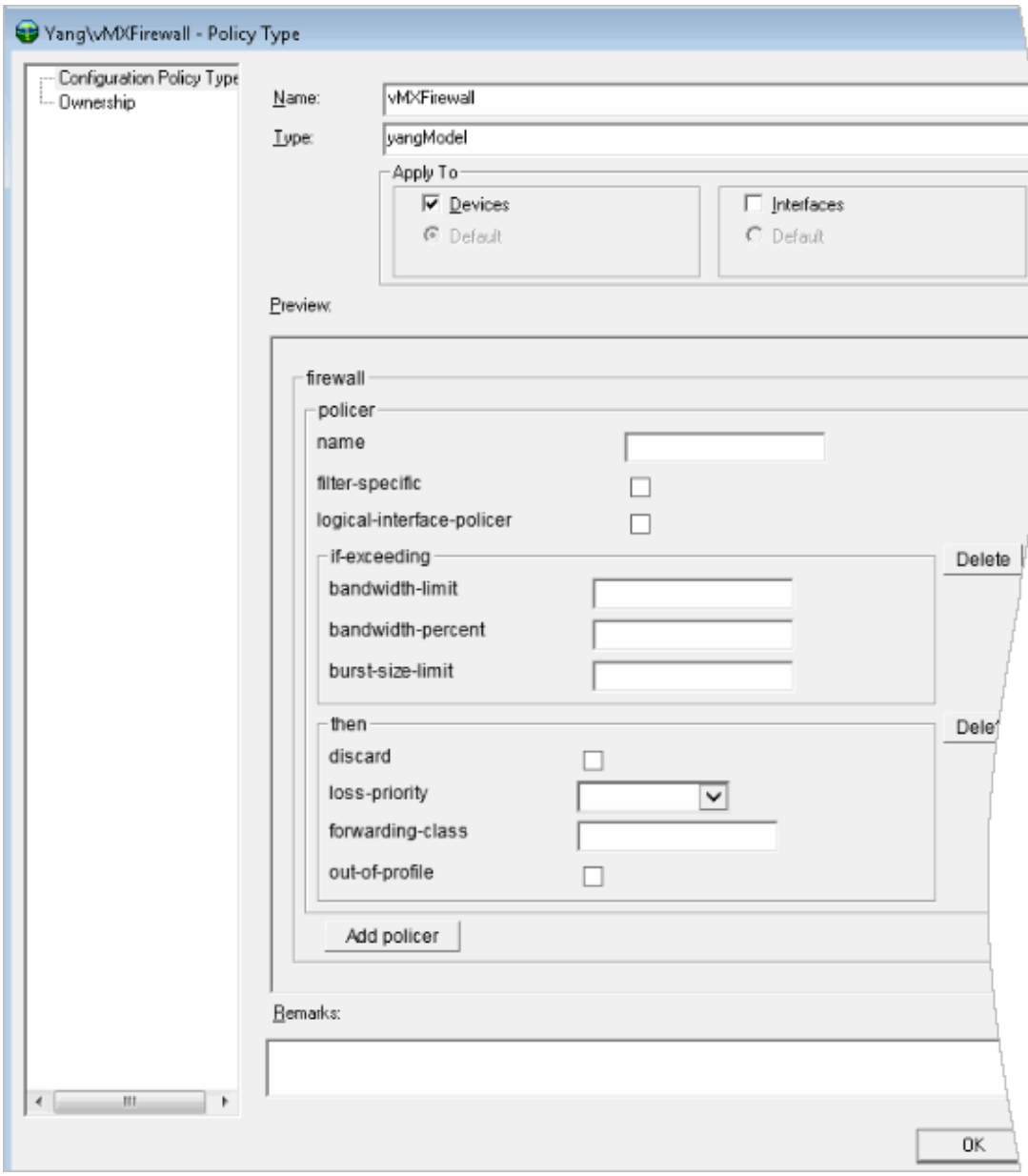
```
container firewall {
   description "Define a firewall configuration";
   uses apply-advanced;
   container family {}
...
  list policer {
    key name;
    ordered-by user;
    description "Policer template definition";
    uses firewall_policer;
  }
  list flexible-match {
```

The configuration policy property file must also define the policer in the `firewall_policer` section:

```
grouping firewall_policer {
description "Define a policer";
leaf name {
description "Policer name";
type string {
junos:posix-pattern "!^((__.*)|(.{65,}))$";
junos:pattern-message "Must be a non-reserved string of 64 characters or less";
}
}  ...additional configuration...
```

This example code produces a configuration policy that contains the contents of Policer section:

Figure 3-1 Configuration Policy properties - Configuration Policy page



Importing and Parsing YANG Model Configuration Policies

You can use the YANG import tool to import and parse your own custom configuration policies.

> **Note:**
>
> To import a configuration policy, you should have a general understanding of the YANG model and an understanding of the filter used in the property file to parse the configuration policy.
>
> There is a 450K size limit on the configuration policy.

You can use the **sample.properties** file as a starting point to import and parse an **snmp** configuration policy.

To parse a YANG model configuration policy:

1. Change to the **yangImporter** directory.

   The **yangImporter** directory is in the following location:

   /opt/installs/OracleCommunications/ServiceActivator/bin/yangImporter

2. Run the following command:

   /yangImporter.sh localhost *port username* /opt/installs/configvSRX.yang testsnmp sample.properties

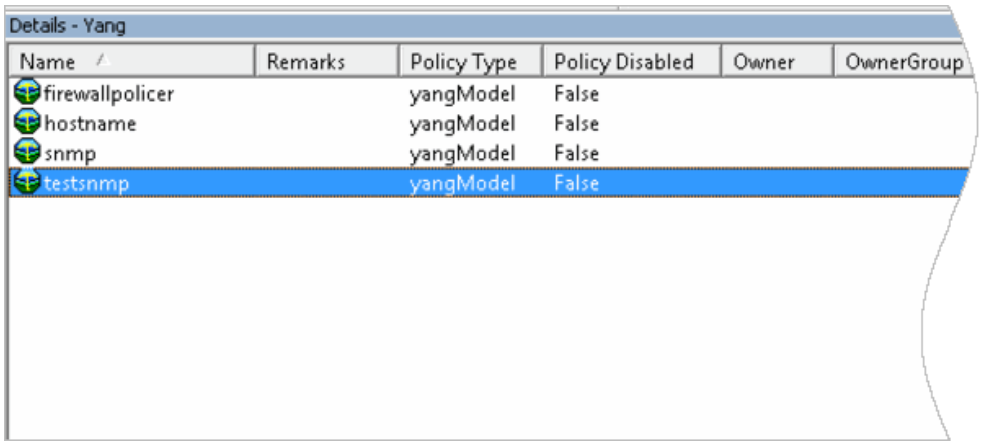
   For example:

   ./yangImporter.sh localhost 2809 rwalter /opt/installs/configvSRX.yang testsnmp sample.properties

3. When prompted, enter the password for the user.

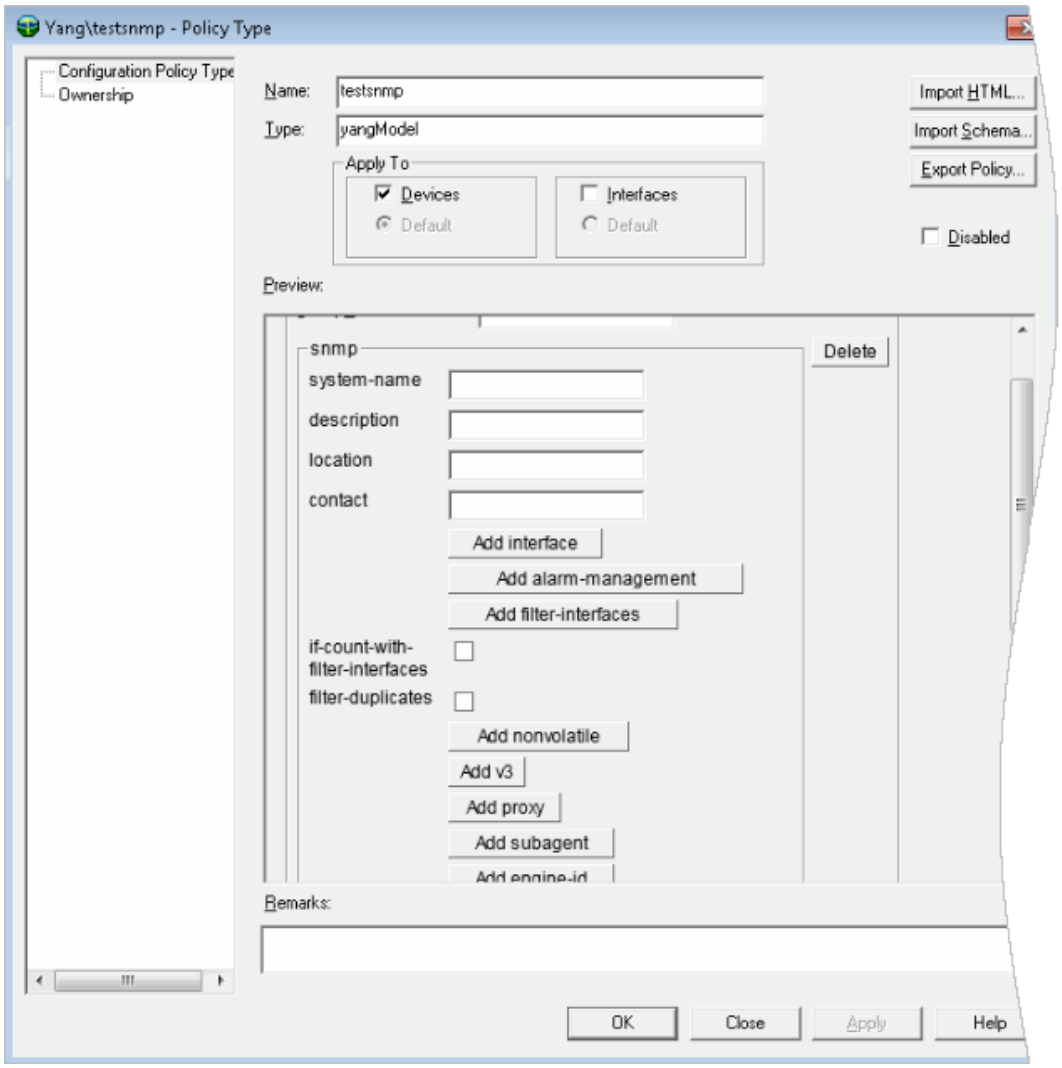
   IP Service Activator generates a build message, creates the configuration policy, and adds the policy to your IP Service Activator system.

Figure 3-2 Adding a Configuration Policy



IP Service Activator creates the HTML based on the YANG model:

Figure 3-3 HTML, Based on the YANG Model



4. Obtain the most recent model for the device.

   To obtain the most recent model for a specific device:

   a. Connect to the device and send the following command:

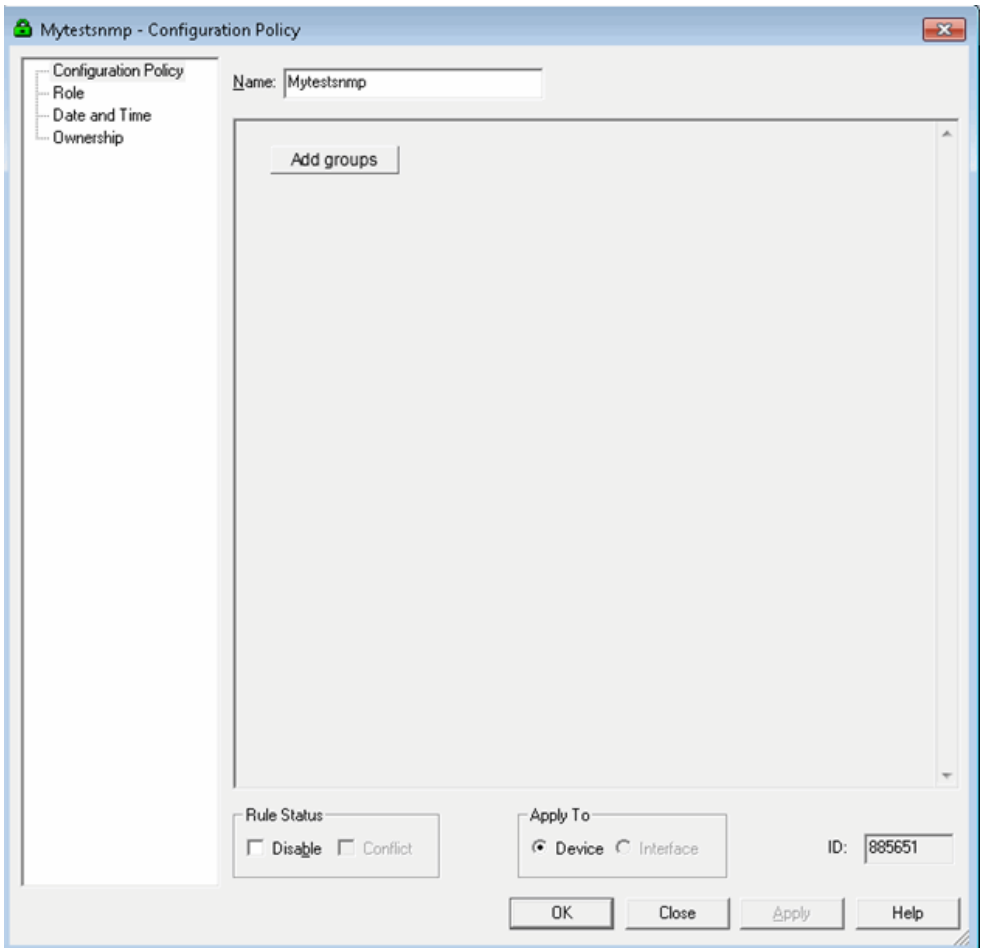
      show system schema module configuration format yang output-file-name /var/tmp/config.yang

   b. Use SFTP to collect the model.

      You may have to remove the first line and the last line of the model if the model includes `<output>` as the first line and `</output>` as the last line.

5. Apply your policy to a NETCONF-enabled device that supports the YANG configuration policy.
6. On the Configuration Policy properties Configuration Policy page, name the policy and click **Add groups**.

Figure 3-4 Configuration Policy Properties



7. Enter a name for the group name, and then select **Add snmp**.
8. Enter the system name, the description, and all other relevant information.
9. Click **Add interface**.
10. Add a valid interface or subinterface for the device.
11. Click **Apply**.
12. On the Configuration Policy properties Role page, add a role.
13. Commit the changes.
14. Review the audit trail for **npNetconf**. For example:

```
2017-01-26 20:26:30|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?>
2017-01-26 20:26:30|10.156.68.176|<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2017-01-26 20:26:30|10.156.68.176| <capabilities>
2017-01-26 20:26:30|10.156.68.176| <capability>
2017-01-26 20:26:30|10.156.68.176| urn:ietf:params:netconf:base:1.0
2017-01-26 20:26:30|10.156.68.176| </capability>
2017-01-26 20:26:30|10.156.68.176| <capability>
2017-01-26 20:26:30|10.156.68.176|
urn:ietf:params:netconf:capability:candidate:1.0
2017-01-26 20:26:30|10.156.68.176| </capability>
2017-01-26 20:26:30|10.156.68.176| </capabilities>
```

```
2017-01-26 20:26:30|10.156.68.176|</hello>
2017-01-26 20:26:30|10.156.68.176|]]>]]>
2017-01-26 20:26:31|10.156.68.176|<rpc> <lock><target><candidate/></target></lock>
2017-01-26 20:26:31|10.156.68.176|</rpc>
2017-01-26 20:26:31|10.156.68.176|]]>]]>
2017-01-26 20:26:31|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?><rpc>
2017-01-26 20:26:31|10.156.68.176| <edit-config>
2017-01-26 20:26:31|10.156.68.176| <target><candidate/></target>
2017-01-26 20:26:31|10.156.68.176| <config>
2017-01-26 20:26:31|10.156.68.176|<configuration xpath="/configuration/groups/
snmp" xmlns="http://yang.juniper.net/yang/1.1/jc" xmlns:xs="http://www.w3.org/2001/
XMLSchema" xmlns:ser="http://www.metasolv.com/serviceactivator/servicemodel"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2017-01-26 20:26:31|10.156.68.176| <groups>
2017-01-26 20:26:31|10.156.68.176| <group_name>mysnmpgroup</group_name>
2017-01-26 20:26:31|10.156.68.176| <snmp>
2017-01-26 20:26:31|10.156.68.176| <system-name>system</system-name>
2017-01-26 20:26:31|10.156.68.176| <description>my description</description>
2017-01-26 20:26:31|10.156.68.176| <location>my location</location>
2017-01-26 20:26:31|10.156.68.176| <contact>me</contact>
2017-01-26 20:26:31|10.156.68.176| <interface yangid="1">ge-0/0/2.5</interface>
2017-01-26 20:26:31|10.156.68.176| <if-count-with-filter-interfaces/>
2017-01-26 20:26:31|10.156.68.176| <filter-duplicates/>
2017-01-26 20:26:31|10.156.68.176| </snmp>
2017-01-26 20:26:31|10.156.68.176| </groups>
2017-01-26 20:26:31|10.156.68.176|</configuration>
2017-01-26 20:26:31|10.156.68.176| </config>
2017-01-26 20:26:31|10.156.68.176| </edit-config>
2017-01-26 20:26:31|10.156.68.176|</rpc>
2017-01-26 20:26:31|10.156.68.176|<rpc> <commit/>
2017-01-26 20:26:31|10.156.68.176|</rpc>
2017-01-26 20:26:31|10.156.68.176|]]>]]>
2017-01-26 20:26:32|10.156.68.176|<rpc> <unlock><target><candidate/></target></
unlock>
2017-01-26 20:26:32|10.156.68.176|</rpc>
2017-01-26 20:26:32|10.156.68.176|]]>]]>
2017-01-26 20:26:32|10.156.68.176|<rpc
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2017-01-26 20:26:32|10.156.68.176| <close-session/>
2017-01-26 20:26:32|10.156.68.176|</rpc>
2017-01-26 20:26:32|10.156.68.176|]]>]]>
2017-01-26 20:26:32|10.156.68.176|#End Configuration
```

The router is updated with the following:

```
groups {
mysnmpgroup {
snmp {
name system;
description "my description";
location "my location";
contact me;
interface ge-0/0/2.5;
if-count-with-filter-interfaces;
filter-duplicates;
}
}
}
```

Example: Importing and Parsing a Custom Configuration Policy

This example demonstrates how to use the IP Service Activator YANG import tool to send a configuration policy to a Juniper device:

```
set firewall policer p-all-1m-5k-discard if-exceeding bandwidth-limit 1m
set firewall policer p-all-1m-5k-discard if-exceeding burst-size-limit 5k
set firewall policer p-all-1m-5k-discard then discard
set firewall policer p-ftp-10p-500k-discard if-exceeding bandwidth-percent 10
set firewall policer p-ftp-10p-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-ftp-10p-500k-discard then discard
set firewall policer p-icmp-500k-500k-discard if-exceeding bandwidth-limit 500k
set firewall policer p-icmp-500k-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-icmp-500k-500k-discard then discard
set firewall family inet filter filter-ipv4-with-limits interface-specific
set firewall family inet filter filter-ipv4-with-limits term t-ftp from protocol
tcp
set firewall family inet filter filter-ipv4-with-limits term t-ftp from port ftp
set firewall family inet filter filter-ipv4-with-limits term t-ftp from port ftp-
data
set firewall family inet filter filter-ipv4-with-limits term t-ftp then policer
p-ftp-10p-500k-discard
set firewall family inet filter filter-ipv4-with-limits term t-icmp from
protocol icmp
set firewall family inet filter filter-ipv4-with-limits term t-icmp then policer
p-icmp-500k-500k-discard
set firewall family inet filter filter-ipv4-with-limits term catch-all then
accept
set interfaces fe-0/1/1 unit 1 family inet filter input filter-ipv4-with-limits
set interfaces fe-0/1/1 unit 1 family inet policer input p-all-1m-5k-discard
```

The example demonstrates how to create a property files for:

- The firewall policer
- The firewall family
- The firewall interface

For each of these property files, the XPath is defined one level above the object and uses the filter to specify the target.

The Firewall Policer

You create a property file called **firewallpolicer.properties** in IP Service Activator.

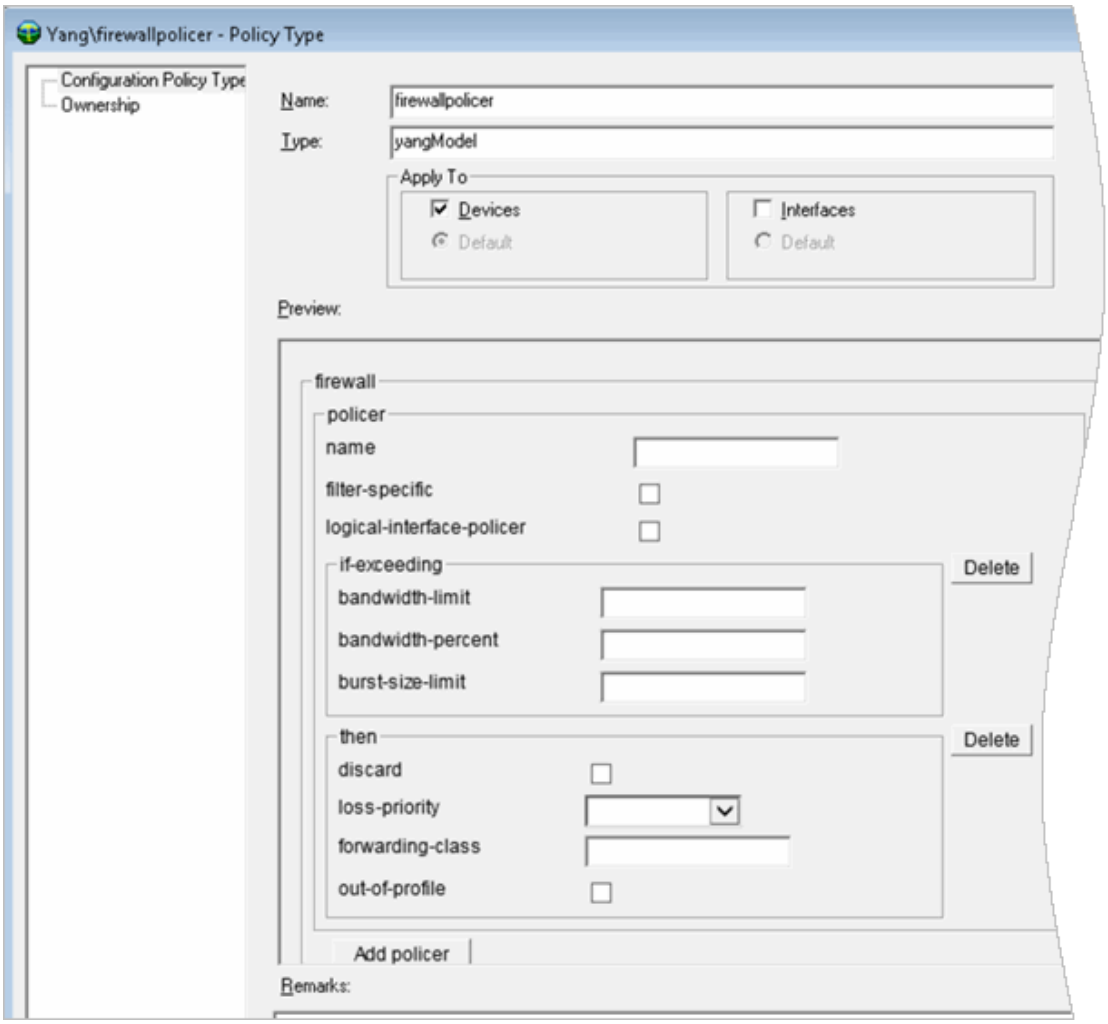
For example:

./yangImporter.sh localhost 2809 rwalter /opt/installs/configvSRX.yang firewallpolicer firewallpolicer.properties

The XPath to this properties file is defined as:

```
xpath=/configuration/firewallbigNodeFilters=firewall::container(policer)
```

Figure 3-5 Firewall Policy



The Firewall Family

You create a property file called **firewallFamily.properties** in IP Service Activator.

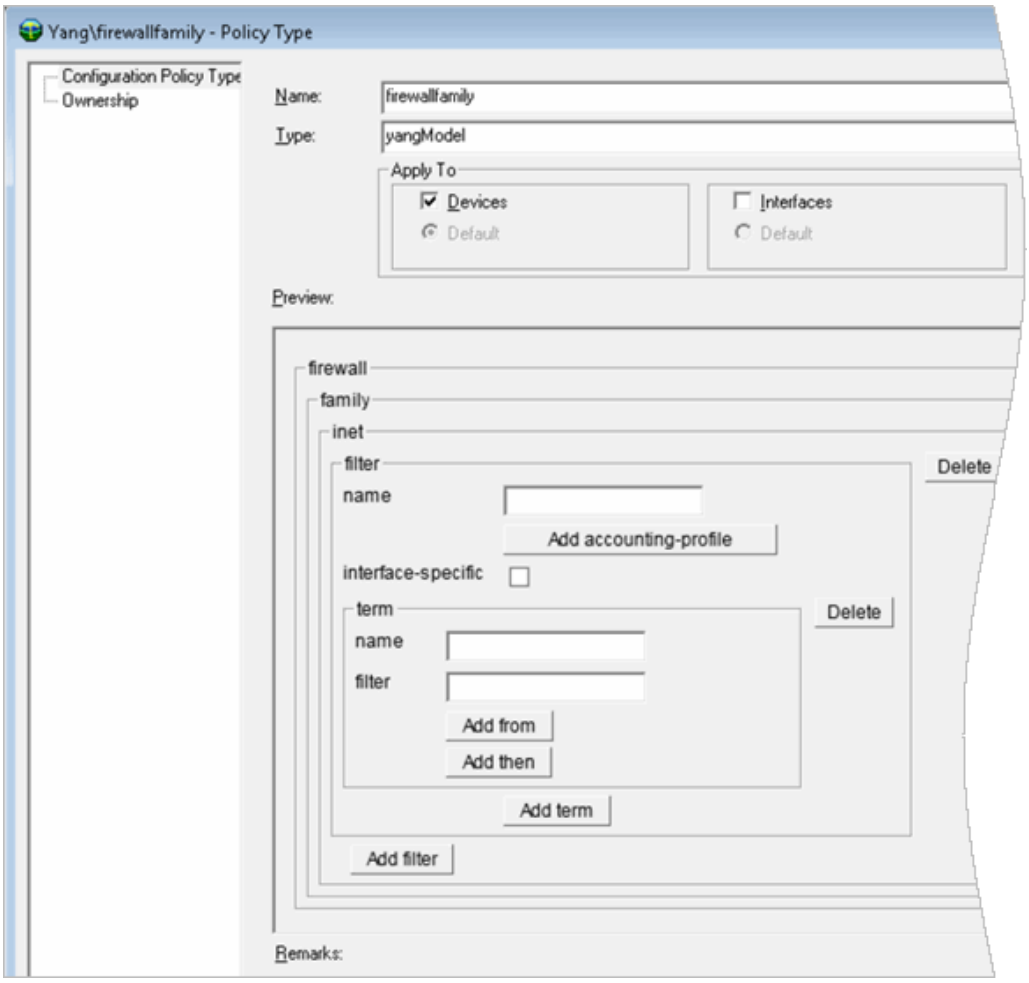
For example:

./yangImporter.sh localhost 2809 rwalter /opt/installs/configvSRX.yang firewallfamily firewallFamily.properties

The XPath to this properties file is defined as:

```
xpath=/configuration/interfaces/interface/unit/family/inet
bigNodeFilters=inet::container(policer,filter);
```

Figure 3-6 Firewall Family



The Firewall Interface

You create a property file called **firewallInterfaceFamilyInet.properties** in IP Service Activator.

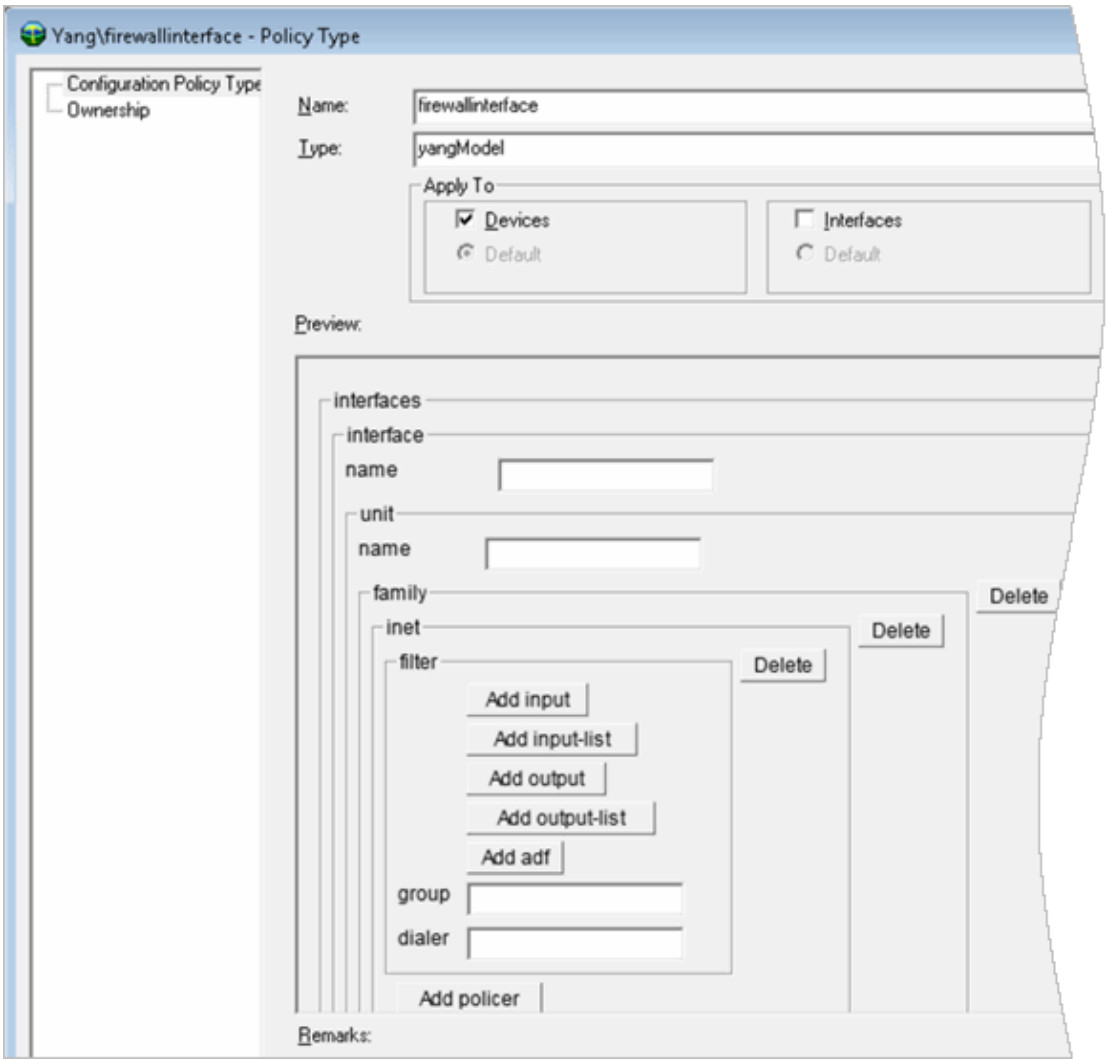
For example:

./yangImporter.sh 127.0.0.1 2809 rwalter /opt/installs/configvSRX.yang firewallInterfaceFamilyInet firewallInterfaceFamilyInet.properties

The XPath to this properties file is defined as:

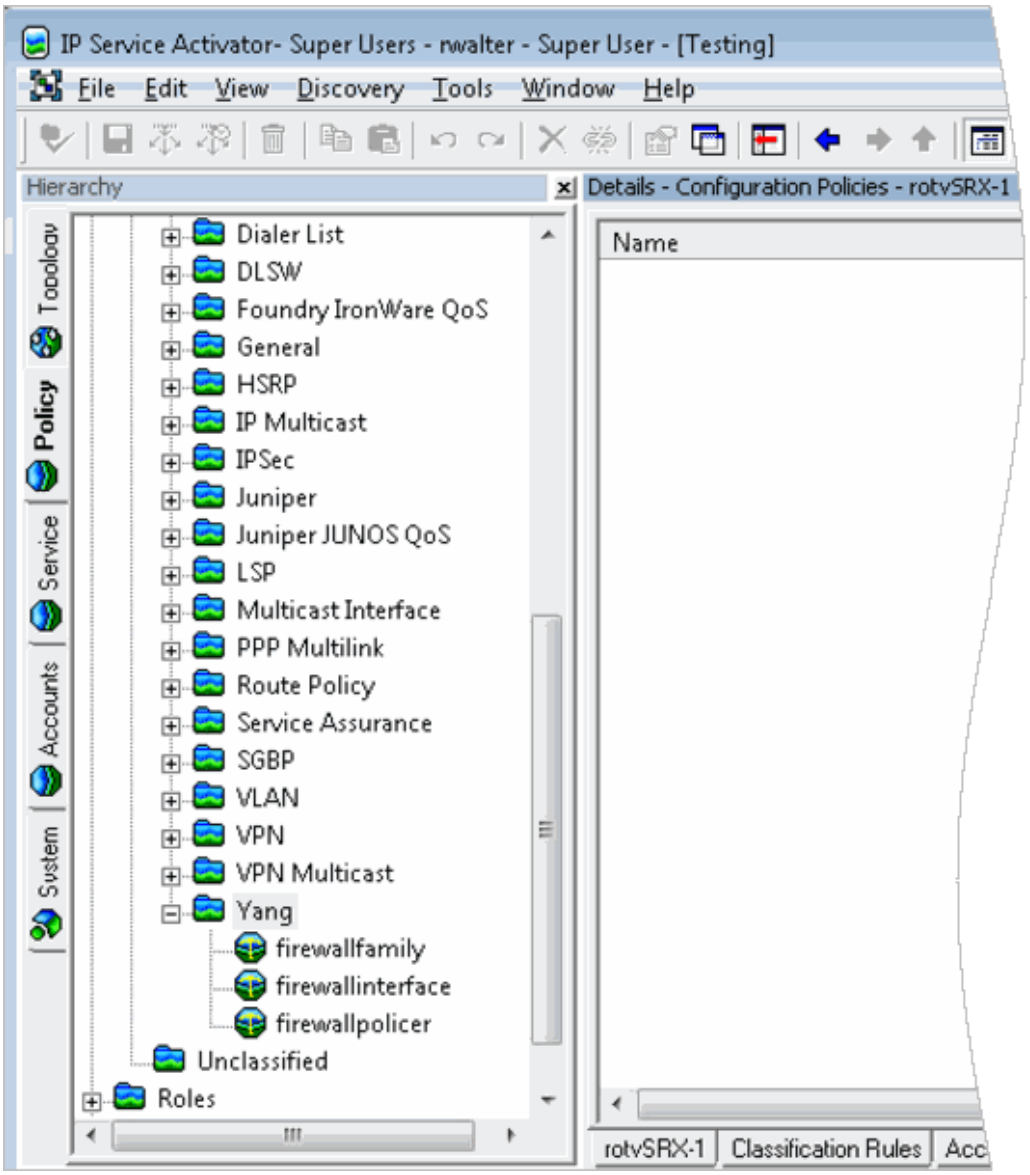
```
xpath=/configuration/firewall/family/
inetbigNodeFilters=firewall::container(filter)
```

Figure 3-7 Firewall Interface



When finished, the following configuration policies now exist in IP Service Activator:

Figure 3-8 Configuration Policies in IP Service Activator



In this example, the device is managed by the Juniper NETCONF cartridge. You apply the firewall policer first, as the firewall family and firewall interface property files are both dependent on the firewall policer property file. When you define the XPath as previously indicated and commit, IP Service Activator sends the following to the router:

```
2017-01-31 20:35:03|10.156.68.176|#Start Configuration
2017-01-31 20:35:03|10.156.68.176|#Applying Configuration
2017-01-31 20:35:04|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?>
2017-01-31 20:35:04|10.156.68.176|<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2017-01-31 20:35:04|10.156.68.176| <capabilities>
2017-01-31 20:35:04|10.156.68.176| <capability>
2017-01-31 20:35:04|10.156.68.176| urn:ietf:params:netconf:base:1.0
2017-01-31 20:35:04|10.156.68.176| </capability>
2017-01-31 20:35:04|10.156.68.176| <capability>
2017-01-31 20:35:04|10.156.68.176|
```

```
urn:ietf:params:netconf:capability:candidate:1.0
2017-01-31 20:35:04|10.156.68.176| </capability>
2017-01-31 20:35:04|10.156.68.176| </capabilities>
2017-01-31 20:35:04|10.156.68.176|</hello>
2017-01-31 20:35:04|10.156.68.176|]]>]]>
2017-01-31 20:35:04|10.156.68.176|<rpc> <lock><target><candidate/></target></lock>
2017-01-31 20:35:04|10.156.68.176|</rpc>
2017-01-31 20:35:04|10.156.68.176|]]>]]>
2017-01-31 20:35:04|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?><rpc>
2017-01-31 20:35:04|10.156.68.176| <edit-config>
2017-01-31 20:35:04|10.156.68.176| <target><candidate/></target>
2017-01-31 20:35:04|10.156.68.176| <config>
2017-01-31 20:35:04|10.156.68.176|<configuration xpath="/configuration/firewall/
policer" xmlns="http://yang.juniper.net/yang/1.1/jc" xmlns:xs="http://www.w3.org/2001/
XMLSchema" xmlns:ser="http://www.metasolv.com/serviceactivator/servicemodel"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2017-01-31 20:35:04|10.156.68.176| <firewall>
2017-01-31 20:35:04|10.156.68.176| <policer>
2017-01-31 20:35:04|10.156.68.176| <name>p-all-1m-5k-discard</name>
2017-01-31 20:35:04|10.156.68.176| <if-exceeding>
2017-01-31 20:35:04|10.156.68.176| <bandwidth-limit>1m</bandwidth-limit>
2017-01-31 20:35:04|10.156.68.176| <burst-size-limit>5k</burst-size-limit>
2017-01-31 20:35:04|10.156.68.176| </if-exceeding>
2017-01-31 20:35:04|10.156.68.176| <then>
2017-01-31 20:35:04|10.156.68.176| <discard/>
2017-01-31 20:35:04|10.156.68.176| </then>
2017-01-31 20:35:04|10.156.68.176| </policer>
2017-01-31 20:35:04|10.156.68.176| <policer yangid="1">
2017-01-31 20:35:04|10.156.68.176| <name>p-ftp-10p-500k-discard</name>
2017-01-31 20:35:04|10.156.68.176| <if-exceeding>
2017-01-31 20:35:04|10.156.68.176| <bandwidth-percent>10</bandwidth-percent>
2017-01-31 20:35:04|10.156.68.176| <burst-size-limit>500k</burst-size-limit>
2017-01-31 20:35:04|10.156.68.176| </if-exceeding>
2017-01-31 20:35:04|10.156.68.176| <then>
2017-01-31 20:35:04|10.156.68.176| <discard/>
2017-01-31 20:35:04|10.156.68.176| </then>
2017-01-31 20:35:04|10.156.68.176| </policer>
2017-01-31 20:35:04|10.156.68.176| <policer yangid="2">
2017-01-31 20:35:04|10.156.68.176| <name>p-icmp-500k-500k-discard</name>
2017-01-31 20:35:04|10.156.68.176| <if-exceeding>
2017-01-31 20:35:04|10.156.68.176| <bandwidth-limit>500k</bandwidth-limit>
2017-01-31 20:35:04|10.156.68.176| <burst-size-limit>500k</burst-size-limit>
2017-01-31 20:35:04|10.156.68.176| </if-exceeding>
2017-01-31 20:35:04|10.156.68.176| <then>
2017-01-31 20:35:04|10.156.68.176| <discard/>
2017-01-31 20:35:04|10.156.68.176| </then>
2017-01-31 20:35:04|10.156.68.176| </policer>
2017-01-31 20:35:04|10.156.68.176| </firewall>
2017-01-31 20:35:04|10.156.68.176|</configuration>
2017-01-31 20:35:04|10.156.68.176| </config>
2017-01-31 20:35:04|10.156.68.176| </edit-config>
2017-01-31 20:35:04|10.156.68.176|</rpc>
2017-01-31 20:35:05|10.156.68.176|<rpc> <commit/>
2017-01-31 20:35:05|10.156.68.176|</rpc>
```

The configuration on the router is updated to the following:

```
show firewall | display set
set firewall policer p-all-1m-5k-discard if-exceeding bandwidth-limit 1m
set firewall policer p-all-1m-5k-discard if-exceeding burst-size-limit 5k
```

```
set firewall policer p-all-1m-5k-discard then discard
set firewall policer p-ftp-10p-500k-discard if-exceeding bandwidth-percent 10
set firewall policer p-ftp-10p-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-ftp-10p-500k-discard then discard
set firewall policer p-icmp-500k-500k-discard if-exceeding bandwidth-limit 500k
set firewall policer p-icmp-500k-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-icmp-500k-500k-discard then discard
```

Next, you create the firewall Family property file. When you define the XPath as previously indicated and commit, IP Service Activator sends the following:

```
2017-01-31 20:41:56|10.156.68.176|#Start Configuration
2017-01-31 20:41:56|10.156.68.176|#Applying Configuration
2017-01-31 20:41:57|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?>
2017-01-31 20:41:57|10.156.68.176|<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2017-01-31 20:41:57|10.156.68.176| <capabilities>
2017-01-31 20:41:57|10.156.68.176| <capability>
2017-01-31 20:41:57|10.156.68.176| urn:ietf:params:netconf:base:1.0
2017-01-31 20:41:57|10.156.68.176| </capability>
2017-01-31 20:41:57|10.156.68.176| <capability>
2017-01-31 20:41:57|10.156.68.176|
urn:ietf:params:netconf:capability:candidate:1.0
2017-01-31 20:41:57|10.156.68.176| </capability>
2017-01-31 20:41:57|10.156.68.176| </capabilities>
2017-01-31 20:41:57|10.156.68.176|</hello>
2017-01-31 20:41:57|10.156.68.176|]]>]]>
2017-01-31 20:41:57|10.156.68.176|<rpc> <lock><target><candidate/></target></
lock>
2017-01-31 20:41:57|10.156.68.176|</rpc>
2017-01-31 20:41:57|10.156.68.176|]]>]]>
2017-01-31 20:41:57|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?><rpc>
2017-01-31 20:41:57|10.156.68.176| <edit-config>
2017-01-31 20:41:57|10.156.68.176| <target><candidate/></target>
2017-01-31 20:41:57|10.156.68.176| <config>
2017-01-31 20:41:57|10.156.68.176|<configuration xpath="/configuration/firewall/
family/inet/filter" xmlns="http://yang.juniper.net/yang/1.1/jc" xmlns:xs="http://
www.w3.org/2001/XMLSchema" xmlns:ser="http://www.metasolv.com/serviceactivator/
servicemodel" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2017-01-31 20:41:57|10.156.68.176| <firewall>
2017-01-31 20:41:57|10.156.68.176| <family>
2017-01-31 20:41:57|10.156.68.176| <inet>
2017-01-31 20:41:57|10.156.68.176| <filter>
2017-01-31 20:41:57|10.156.68.176| <name>filter-ipv4-with-limits</name>
2017-01-31 20:41:57|10.156.68.176| <interface-specific/>
2017-01-31 20:41:57|10.156.68.176| <term yangid="1">
2017-01-31 20:41:57|10.156.68.176| <name>t-ftp</name>
2017-01-31 20:41:57|10.156.68.176| <from>
2017-01-31 20:41:57|10.156.68.176| <protocol yangid="2">tcp</protocol>
2017-01-31 20:41:57|10.156.68.176| <port yangid="3">ftp</port>
2017-01-31 20:41:57|10.156.68.176| <port yangid="4">ftp-data</port>
2017-01-31 20:41:57|10.156.68.176| </from>
2017-01-31 20:41:57|10.156.68.176| <then>
2017-01-31 20:41:57|10.156.68.176| <policer>p-ftp-10p-500k-discard</policer>
2017-01-31 20:41:57|10.156.68.176| </then>
2017-01-31 20:41:57|10.156.68.176| </term>
2017-01-31 20:41:57|10.156.68.176| <term yangid="6">
2017-01-31 20:41:57|10.156.68.176| <name>t-icmp</name>
2017-01-31 20:41:57|10.156.68.176| <from>
2017-01-31 20:41:57|10.156.68.176| <protocol yangid="7">icmp</protocol>
2017-01-31 20:41:57|10.156.68.176| </from>
```

```
2017-01-31 20:41:57|10.156.68.176| <then>
2017-01-31 20:41:57|10.156.68.176| <policer>p-icmp-500k-500k-discard</policer>
2017-01-31 20:41:57|10.156.68.176| </then>
2017-01-31 20:41:57|10.156.68.176| </term>
2017-01-31 20:41:57|10.156.68.176| <term yangid="8">
2017-01-31 20:41:57|10.156.68.176| <name>catch-all</name>
2017-01-31 20:41:57|10.156.68.176| <then>
2017-01-31 20:41:57|10.156.68.176| <accept/>
2017-01-31 20:41:57|10.156.68.176| </then>
2017-01-31 20:41:57|10.156.68.176| </term>
2017-01-31 20:41:57|10.156.68.176| </filter>
2017-01-31 20:41:57|10.156.68.176| </inet>
2017-01-31 20:41:57|10.156.68.176| </family>
2017-01-31 20:41:57|10.156.68.176| </firewall>
2017-01-31 20:41:57|10.156.68.176|</configuration>
```

The configuration on the router is updated to:

```
show firewall | display set
set firewall family inet filter filter-ipv4-with-limits interface-specific
set firewall family inet filter filter-ipv4-with-limits term t-ftp from protocol tcp
set firewall family inet filter filter-ipv4-with-limits term t-ftp from port ftp
set firewall family inet filter filter-ipv4-with-limits term t-ftp from port ftp-data
set firewall family inet filter filter-ipv4-with-limits term t-ftp then policer p-
ftp-10p-500k-discard
set firewall family inet filter filter-ipv4-with-limits term t-icmp from protocol icmp
set firewall family inet filter filter-ipv4-with-limits term t-icmp then policer p-
icmp-500k-500k-discard
set firewall family inet filter filter-ipv4-with-limits term catch-all then accept
set firewall policer p-all-1m-5k-discard if-exceeding bandwidth-limit 1m
set firewall policer p-all-1m-5k-discard if-exceeding burst-size-limit 5k
set firewall policer p-all-1m-5k-discard then discard
set firewall policer p-ftp-10p-500k-discard if-exceeding bandwidth-percent 10
set firewall policer p-ftp-10p-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-ftp-10p-500k-discard then discard
set firewall policer p-icmp-500k-500k-discard if-exceeding bandwidth-limit 500k
set firewall policer p-icmp-500k-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-icmp-500k-500k-discard then discard
```

Next, you create the firewall Interface property file. When you define the XPath as previously indicated and commit, IP Service Activator sends the following:

```
2017-01-31 20:46:32|10.156.68.176|#Applying Configuration
2017-01-31 20:46:33|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?>
2017-01-31 20:46:33|10.156.68.176|<hello
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2017-01-31 20:46:33|10.156.68.176| <capabilities>
2017-01-31 20:46:33|10.156.68.176| <capability>
2017-01-31 20:46:33|10.156.68.176| urn:ietf:params:netconf:base:1.0
2017-01-31 20:46:33|10.156.68.176| </capability>
2017-01-31 20:46:33|10.156.68.176| <capability>
2017-01-31 20:46:33|10.156.68.176| urn:ietf:params:netconf:capability:candidate:1.0
2017-01-31 20:46:33|10.156.68.176| </capability>
2017-01-31 20:46:33|10.156.68.176| </capabilities>
2017-01-31 20:46:33|10.156.68.176|</hello>
2017-01-31 20:46:33|10.156.68.176|]]>]]>
2017-01-31 20:46:33|10.156.68.176|<rpc> <lock><target><candidate/></target></lock>
2017-01-31 20:46:33|10.156.68.176|</rpc>
2017-01-31 20:46:33|10.156.68.176|]]>]]>
2017-01-31 20:46:33|10.156.68.176|<?xml version="1.0" encoding="UTF-8"?><rpc>
2017-01-31 20:46:33|10.156.68.176| <edit-config>
```

```
2017-01-31 20:46:33|10.156.68.176| <target><candidate/></target>
2017-01-31 20:46:33|10.156.68.176| <config>
2017-01-31 20:46:33|10.156.68.176|<configuration xpath="/configuration/
interfaces/interface/unit/family/inet/filter,policer" xmlns="http://
yang.juniper.net/yang/1.1/jc" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:ser="http://www.metasolv.com/serviceactivator/servicemodel"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
2017-01-31 20:46:33|10.156.68.176| <interfaces>
2017-01-31 20:46:33|10.156.68.176| <interface>
2017-01-31 20:46:33|10.156.68.176| <name>ge-0/0/3</name>
2017-01-31 20:46:33|10.156.68.176| <unit yangid="1">
2017-01-31 20:46:33|10.156.68.176| <name>60</name>
2017-01-31 20:46:33|10.156.68.176| <family>
2017-01-31 20:46:33|10.156.68.176| <inet>
2017-01-31 20:46:33|10.156.68.176| <filter>
2017-01-31 20:46:33|10.156.68.176| <input>
2017-01-31 20:46:33|10.156.68.176| <filter-name>filter-ipv4-with-limits</filter-
name>
2017-01-31 20:46:33|10.156.68.176| </input>
2017-01-31 20:46:33|10.156.68.176| </filter>
2017-01-31 20:46:33|10.156.68.176| <policer>
2017-01-31 20:46:33|10.156.68.176| <input>p-all-1m-5k-discard</input>
2017-01-31 20:46:33|10.156.68.176| </policer>
2017-01-31 20:46:33|10.156.68.176| </inet>
2017-01-31 20:46:33|10.156.68.176| </family>
2017-01-31 20:46:33|10.156.68.176| </unit>
2017-01-31 20:46:33|10.156.68.176| </interface>
2017-01-31 20:46:33|10.156.68.176| </interfaces>
2017-01-31 20:46:33|10.156.68.176|</configuration>
2017-01-31 20:46:33|10.156.68.176| </config>
2017-01-31 20:46:33|10.156.68.176| </edit-config>
2017-01-31 20:46:33|10.156.68.176|</rpc>
2017-01-31 20:46:34|10.156.68.176|<rpc> <commit/>
2017-01-31 20:46:34|10.156.68.176|</rpc>
2017-01-31 20:46:34|10.156.68.176|]]>]]>
2017-01-31 20:46:35|10.156.68.176|<rpc> <unlock><target><candidate/></target></
unlock>
2017-01-31 20:46:35|10.156.68.176|</rpc>
2017-01-31 20:46:35|10.156.68.176|]]>]]>
2017-01-31 20:46:35|10.156.68.176|<rpc
xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
2017-01-31 20:46:35|10.156.68.176| <close-session/>
2017-01-31 20:46:35|10.156.68.176|</rpc>
2017-01-31 20:46:35|10.156.68.176|]]>]]>
2017-01-31 20:46:35|10.156.68.176|#End Configuration
```

The configuration on the router is updated to:

```
set interfaces ge-0/0/3 unit 60 family inet filter input filter-ipv4-with-limits
set interfaces ge-0/0/3 unit 60 family inet policer input p-all-1m-5k-discard
.
.
.
set firewall family inet filter filter-ipv4-with-limits interface-specific
set firewall family inet filter filter-ipv4-with-limits term t-ftp from protocol
tcp
set firewall family inet filter filter-ipv4-with-limits term t-ftp from port ftp
set firewall family inet filter filter-ipv4-with-limits term t-ftp from port ftp-
data
set firewall family inet filter filter-ipv4-with-limits term t-ftp then policer
p-ftp-10p-500k-discard
```

```
set firewall family inet filter filter-ipv4-with-limits term t-icmp from protocol icmp
set firewall family inet filter filter-ipv4-with-limits term t-icmp then policer p-
icmp-500k-500k-discard
set firewall family inet filter filter-ipv4-with-limits term catch-all then accept
set firewall policer p-all-1m-5k-discard if-exceeding bandwidth-limit 1m
set firewall policer p-all-1m-5k-discard if-exceeding burst-size-limit 5k
set firewall policer p-all-1m-5k-discard then discard
set firewall policer p-ftp-10p-500k-discard if-exceeding bandwidth-percent 10
set firewall policer p-ftp-10p-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-ftp-10p-500k-discard then discard
set firewall policer p-icmp-500k-500k-discard if-exceeding bandwidth-limit 500k
set firewall policer p-icmp-500k-500k-discard if-exceeding burst-size-limit 500k
set firewall policer p-icmp-500k-500k-discard then discard
```

A

Configuration Policy Generation Properties

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

This file contains a number of properties that customize the generated configuration policy 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 (i.e. **true** or **false**).

Regex patterns are validated. For a regex it maybe necessary to use an escape sequence preceding special characters in order for them to be translated to the source code.

Table A-1 shows the naming and packaging properties.

Table A-1 Naming and Packaging Properties

Property	Description	Example
sdk_global_configPolicyName	Configuration policy name. This variable is used throughout the generated source code. This property is mandatory.	`bannerSample`
sdk_global_package	This is the configuration policy path in dotted notation used for packaging. Its value is translated to a directory structure for the source files path generation. The value is used in build scripts. The generated files are placed in *SDK_home***configPolicies***sdk_global_configPolicyName***src***sdk_global_package* This property is mandatory	`com.metasolv.serviceactivator.bannerSample` becomes `com\metasolv\serviceactivator\bannerSample`

Table A-1 (Cont.) Naming and Packaging Properties

Property	Description	Example
sdk_global_configPolicyVersion	Configuration policy version that is being developed. The version is used by the Network Processor at runtime to detect that an upgrade of an existing service model may be required in the event of an upgrade of the configuration policy. Refer to "Configuration Policy Version". This property is mandatory.	`1.0`
sdk_schema_namespace	Target namespace of the configuration policy schema. This property is mandatory.	--
sdk_schema_namespaceAbbr	Abbreviation of the target namespace of the configuration policy schema. This is used as a namespace prefix in the schema. This property is mandatory.	`Bn`
sdk_schema_topLevelTag	Name of the element in the schema of type sdk_schema_topLevelType. The generated schema will contain one such element. This property is mandatory.	`banners`
sdk_schema_topLevelType	Name of the complex type in the configuration policy schema which will describe the configuration policy. This property is mandatory.	`Banners`
sdk_xmlbeans_package	The java package in which the XmlBeans will be generated. This is primarily used to build the HTML GUI. This property is mandatory.	`com.metasolv.serviceactivator.banner`