Oracle® Communications IP Service Activator

OSS Java Development Library Guide

Release 7.2

E47731-01

October 2013

ORACLE®

Oracle Communications IP Service Activator OSS Java Development Library Guide, Release 7.2

E47731-01

Contents

Preface v

Audience v
Documentation Accessibility v

1 Overview

About the OJDL 1-1
System Architecture 1-1
Prerequisites for Installing OJDL 1-3
Installing OJDL 1-3

2 Using the OJDL

Java Development Environment 2-1
OJDL Directory and File Structure 2-1
The doc Directory 2-2
The lib Directory 2-2
The Samples Directory 2-3
JavaDocs 2-3
Java Classes 2-3
Best Practices for Minimizing Commits 2-4

A Managing Configuration Policies Using the OJDL API

Initial Setup A-1
Creating a Configuration Policy A-1
Creating the Configuration Policy Data Type A-2
Creating the RuleGeneric Object to Contain the Configuration Policy A-2
Assigning the Configuration Policy to the Required Device and Interface Roles A-2
Modifying a Configuration Policy A-3
Querying the EOM for the Configuration Policy A-3
Modifying the Policy Definition A-3
Registering an Interface Policy A-3
Creating a Subinterface A-3
Creating the Subinterface Object A-4
Linking the New Subinterface Object to the Interface Policy Registration A-4
Modifying the Interface Configuration Policy Data A-4
Linking the New Subinterface to an Interface Role A-4

Optionally Discovering the Device A-4
Creating a Main Interface A-5
Decorating an Interface A-5
Comparing Created and Discovered Interfaces A-5
Configuration Policy Classes A-5
Example Source Code A-9

Preface

This guide provides information for developing application programming interfaces to Oracle Communications IP Service Activator.

The OSS Java Development Library (OJDL) provides a Java-based Application Programming Interface (API) to IP Service Activator. It comprises a set of Java classes together with some code samples and an example Web interface.

Audience

This guide is intended for systems integrators and developers who will be using the OJDL to develop their own Java-based interfaces, for example customized web-based applications for Customer Network Management.

It assumes that readers have the following knowledge:

- Knowledge of the OSS Integration Manager (OIM), including the External Object Model (EOM), the OIM command language, and the ability to write scripts

  Refer to *IP Service Activator OSS Integration Manager Guide* for further information
- Familiarity with the core IP Service Activator features
- Experience of the Java programming language and Java technologies
- Knowledge of the Oracle Solaris operating system and its commands, including the ability to use a text editor such as vi

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 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.

1 Overview

This chapter provides a brief introduction to the OSS Java Development Library (OJDL), the components of the package, and its relationship to the rest of the Oracle Communications IP Service Activator system.

About the OJDL

The OJDL is a generic Java API for IP Service Activator, which allows Java developers to develop or customize interfaces, including web-based or intranet-based user interfaces.

The OJDL package comprises:

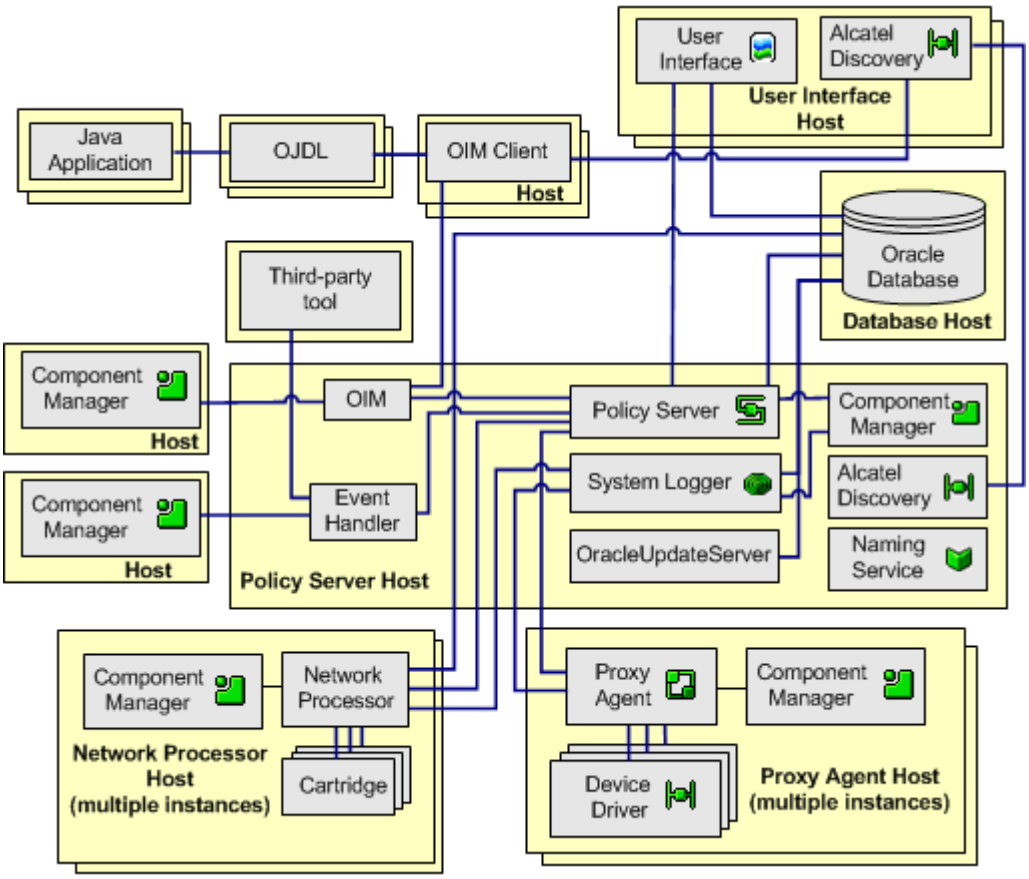
- Java Classes
- Java code samples

The OJDL can be used to develop Java-based interfaces used to integrate IP Service Activator with components of your environment. These could include, for example, your internal Operational Support System (OSS) environment or an external Customer Network Management solution.

System Architecture

Figure 1–1 shows the relationship between the OJDL and the rest of the IP Service Activator system:

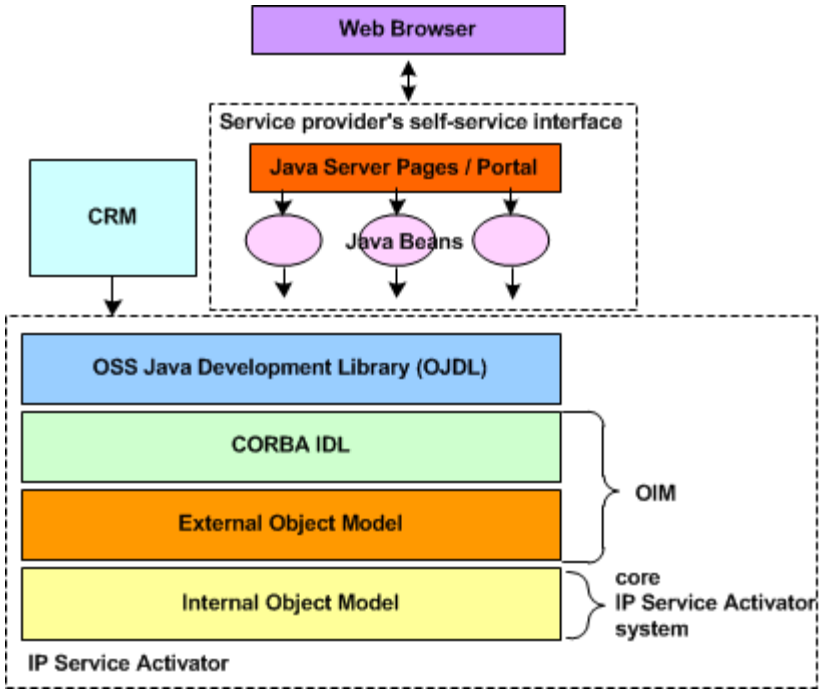
Figure 1–1 OJDL in the IP Service Activator System



The OJDL uses the OSS Integration Manager (OIM) interface and provides access to the External Object Model (EOM), a simplified version of IP Service Activator's internal object model used by the OIM API. The OJDL is OSS compliant.

In effect, the OJDL provides additional layers that are built on top of OIM, which in turn sits on top of the core IP Service Activator system, as shown in Figure 1–2.

Figure 1–2 OJDL in the IP Service Activator Architecture Layers



The EOM is a subset of IP Service Activator's internal object model. It defines all the objects that can be accessed or updated by external applications, including their attributes and the relationships between them. The EOM allows users and user programs to create and access data objects without requiring knowledge of the underlying complexity of the entire object model.

The OJDL Java Classes provide access to the objects in the EOM. The OJDL provides the same functionality as the OIM CLI, allowing you to create objects, get and set attributes, search for objects, manage transactions, and report errors.

Prerequisites for Installing OJDL

There are no restrictions on where to install the OJDL directory on the host system.

The prerequisites for using the OJDL are:

- Your IP Service Activator installation must include an instance of the OIM. For more information, see *IP Service Activator Installation Guide*.
- If you are developing Java code or running web-based applications, a suitable Java development environment must be installed, such as the Java Platform, Standard Edition (Java SE). For more information, see "Java Development Environment".

Installing OJDL

The OJDL package is not installed as part of the IP Service Activator standard installation. It is a separate package downloadable from the Oracle software delivery Web site.

To install the OJDL:

1. Log in to the Oracle software delivery Web site and select **Product Downloads**. Select **Oracle Communications IP Service Activator**, then the Components folder for the desired release.
2. Download the **ojdlpackage-***rel#.build#***.zip** file available at the following path on the Oracle software delivery Web site:

   Oracle Communications IP Service Activator Media Pack -> Oracle Communications IP Service Activator Software for Solaris
3. Move the file to the desired directory on the host where you are installing OJDL. There are no restrictions on which directory path you choose.
4. Unzip the file to create the OJDL directory. The name of the OJDL directory is denoted as **ojdlpackage-***rel#.build#*.

   The OJDL directory consists of the following subdirectories:

   - doc: contains the Java documentation (JavaDocs)
   - lib: contains the OJDL jar file that contains the java classes
   - samples: contains code samples for testing purposes

2

Using the OJDL

This chapter outlines the OJDL, including the Java classes provided for developers.

Java Development Environment

In order to develop Java code you need a suitable development environment, such as the Java Platform, Standard Edition (Java SE), which includes the Java Development Kit (JDK). Java SE can be downloaded from the Oracle Technology Network Web site:

`http://www.oracle.com/technetwork/java/javase/downloads/index.html`

For information about the recommended JDK version for use with the OJDL, see *IP Service Activator Installation Guide*.

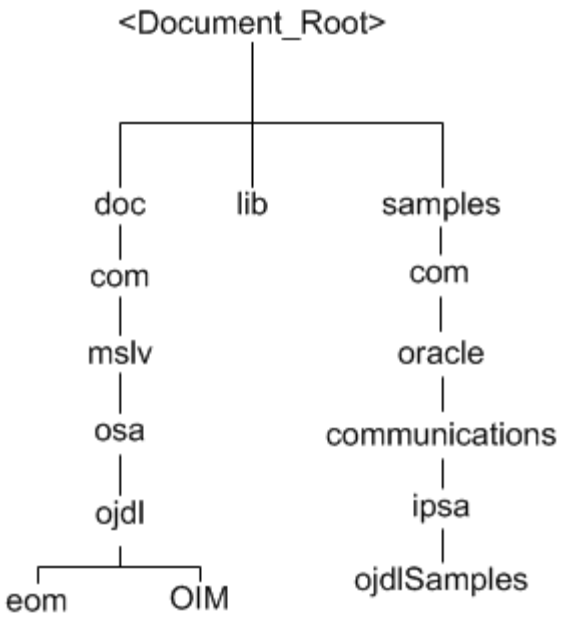
Java SE/JDK can be downloaded free of charge, and can be used for commercial or non-commercial purposes. However, you must retain the copyright notices.

This guide assumes the use of JDK, but other suitable Java tools can be used if required. You need to ensure they are configured correctly.

OJDL Directory and File Structure

When using the OJDL for developing Java code, you need the directories shown in Figure 2–1.

Figure 2–1 OJDL Directories



The doc Directory

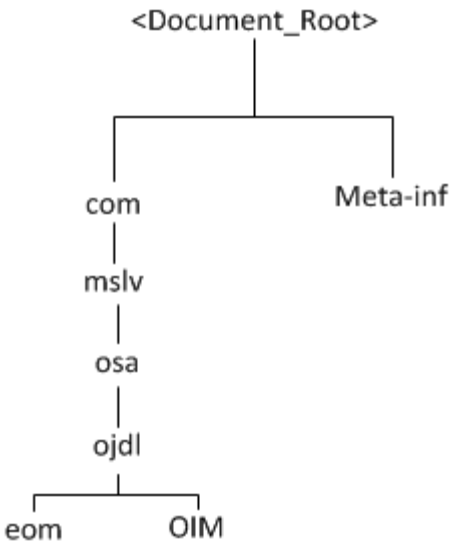
The doc directory contains HTML files containing class and package information. The **index.html** file lists all the classes and packages, and contains links to access the HTML files which provide the relevant information. The doc directory contains the following two subdirectories:

- **doc\com\mslv\osa\ojdl\eom** contains HTML files describing the EOM Java API classes.
- **doc\com\mslv\osa\ojdl\Oim** contains HTML files describing the OIM Java API subclasses (for the OIM IDL).

The lib Directory

The lib directory contains the **ojdl.jar** file, which contains a compressed version of all the OJDL Java classes. Figure 2–2 shows the internal directory structure of **ojdl.jar**.

Figure 2–2 Internal Directory Structure of ojdl.jar



Note: You must put the **ojdl.jar** file in the class path.

The Samples Directory

The samples directory contains Java code samples, which provide an illustration of the use of the OJDL classes. The Java code samples are available in the following directory:

samples\com\oracle\communications\ipsa\ojdlSamples

For a brief explanation of the code samples, see the **README.txt** file in the ojdlSamples folder.

JavaDocs

The JavaDocs are stored in the doc directory. For more information, see "The doc Directory".

Java Classes

The OJDL Java classes provide access to the EOM objects. The classes can also be used to create Java beans which can then be used to create reusable user interface components for particular tasks. These may be written as Java applications, applets, or as scripts within a Java Server Page.

The OJDL Java classes are stored in the lib directory. For more information, see "The lib Directory".

The Java classes are documented as follows:

- Information on the classes is accessed through the **doc\index.htm** file.
- Details of methods and variables used are contained in the **doc\index-all.htm** file

The main classes are summarized in Table 2–1. Refer to the JavaDocs for details.

Table 2–1 Main Java Classes

Class	Description
EomAttribute	A base class for representing an attribute within the EOM.
EomAttributesSe	Holds a set of EomAttribute objects.
EomConnectionManager	Defines a connection manager interface for connecting to the OIM.
EomDebug	Provides a way to enable traces.
EomDefaultConnection	The default implementation of EomConnectManager using the JDK ORB.
EomDifferenceResolver	Finds the logical difference of EomObjects contained in two iterators.
EomDiscovery	Enables the discovery of devices.
EomException	The base class for any exception thrown by the OJDL. May be thrown by methods interacting with the OIM.
EomExtendedSearchIterator	Extends the EomSearchIterator by searching on an iterator of EomObjects.
EomIntersectionResolver	Finds the logical intersection of EomObjects contained in two iterators.
EomIterator	An extension of the **java.util.Iterator** interface. Provides a wrapper around the search functionality of the OIM **find** command.
EomIteratorParameters	Provides a way to pass parameters to an EomSearchIterator to refine the search.
EomNonIntegerException	Thrown when a non-integer value is being assigned to an integer variable.
EomObject	A base class for representing objects within the EOM. Each object has an Id, name, and set of attributes.
EomObjectException	Thrown during an EomObject creation.
EomObjectFactory	A factory to build EomObjects.
EomOimException	Thrown when a command cannot be executed by OIM.
EomResolver	A base class for combining the results of two iterator sets.
EomSearchIterator	Looks for all objects of a specific type with a given attribute.
EomSession	Represents a connection to the OIM.
EomSessionException	Thrown by a method in EomSession when connected to the OIM.
EomTransaction	Models the general IP Service Activator transaction concept.
EomTransactionStateChange	A base class that allows IP Service Activator to synchronously return configuration success or failure messages through the OJDL for transactions which perform adds, modifies, or deletes.

Best Practices for Minimizing Commits

It is good to minimize the number of IP Service Activator transaction commits to complete an operation, because each commit introduces a delay when the object model is updated to reflect the new changes.

The following example shows how commits may be minimized when an application generates a large number of devices for testing. These devices are all of the same type and use the device capabilities of an existing device.

To link the desired capabilities, you must first unlink the default capabilities that are linked when the device is created. In the least efficient case, the client code would take three commits; device create, commit, setpath to device and unlink existing caps, commit, link new caps, commit.

In the more efficient form, the client code could accomplish this through one commit by constructing a reference to the default capabilities of the device by appending the following to the path of the device object:

```
/DeviceCapabilities:"DeviceCapabilities"
```

The following excerpt shows how to construct an EomIdentifier that references the capabilities linked to the device when the transaction is committed:

```
EomIdentifier idForNotYetLinkedDefaultCaps = new
EomIdentifier("DeviceCapabilities", "DeviceCapabilities", newDevice.getPath() +
"/" + "DeviceCapabilities" + ":\"DeviceCapabilities\"");
```

Then the default caps are unlinked and the appropriate device caps are linked using the following excerpt:

```
itsTransaction.unlinkObjects(newDevice,
itsSession.createEomObject(idForNotYetLinkedDefaultCaps));
itsTransaction.linkObjects(newDevice, deviceCaps);
tr = eomSession.openTransaction();
tr.commit();
```

Very large transactions can take more time to process once they are committed. This must be balanced against the overall number of commits you issue.

A

Managing Configuration Policies Using the OJDL API

This appendix outlines how to use the OJDL API to manage Configuration Policies in Oracle Communications IP Service Activator.

Initial Setup

The following JAR files are required in the application classpath in order to create configuration policies using the OJDL API. They are installed with the Network Processor.

- **servicemodelextensions.jar** contains XML Bean Classes for the Configuration Policy Service Model Extensions.
- **xbean.jar** contains Apache XML Beans API.
- **jsr173_api.jar** contains streaming API for XML, provided as part of the Apache XML Beans API.
- **ojdl.jar** contains IP Service Activator OSS Java Development Library (OJDL) API.

These files are located in the following IP Service Activator installation directory:

- Solaris: **/opt/OracleCommunications/ServiceActivator/lib/java-lib**

If your development environment is on a separate PC you will have to copy the JAR files from an IP Service Activator machine.

Creating a Configuration Policy

Configuration policies are optional XML extensions to the IP Service Activator object model that are supported by the Network Processor cartridges.

A configuration policy can be created using the OJDL API by creating a RuleGeneric object. A RuleGeneric object must have two parent objects: a Policy Type object, and the object to which you want it to apply. The latter object can be an interface, or other applicable objects. For details, refer to the discussion of the RuleGeneric object and the external object model in *IP Service Activator OSS Integration Manager Guide*.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the StaticNATsConfigurationPolicyExample file in **samples\com\oracle\communications\ipsa\ojdlSamples**. The Configuration Policy XML definition is set in the RuleGeneric ContentValue attribute.

The structure of the XML for each configuration policy is defined by an XML Schema specification in **servicemodelextension-api-***versionNum.buildNum***.zip**, which is

installed with the IP Service Activator client in the *Service_Activator_Home***SamplePolicy** folder. An API is provided to programmatically construct the configuration policy XML data structures using Java XML Beans, using the Apache XML Beans technology available at the Apache web site:

http://xmlbeans.apache.org/

The following example includes code which interacts with configuration policy objects using the OJDL APIs.

Creating the Configuration Policy Data Type

Each configuration policy top level XML element is represented by an XML Beans Document class. For example, the StaticNats configuration policy is created as a StaticNatsDocument object. Refer to "Configuration Policy Classes" for the complete configuration policy class mapping.

The content of the StaticNats object is set using the XML Beans API.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the StaticNATsConfigurationPolicyExample file in the samples directory.

Creating the RuleGeneric Object to Contain the Configuration Policy

Configuration Policy objects are represented in the IP Service Activator External Object Model (EOM) as RuleGeneric objects. The following two attributes must be set:

- ContentType: the configuration policy type
- ContentValue: the configuration policy xml string

The ContentValue configuration policy XML is generated by invoking the **toString()** function.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the StaticNATsConfigurationPolicyExample file in the samples directory.

When passing XML strings into the EOM object attributes, some special characters need to be escaped by pre-pending an additional \ character. For example, \" and \' must be fully escaped to \\\" and \\\' respectively. This conversion is performed by the **escapeForOIM()** function provided in the example.

Assigning the Configuration Policy to the Required Device and Interface Roles

The RuleGeneric object can be created as a child of many objects in the object hierarchy (as documented in *IP Service Activator OSS Integration Manager Guide*). However, the policy object **Concrete** is applied on any of the Interface objects in the inheritance hierarchy that match the RuleGeneric Roles. The RuleGeneric device and interface roles must match the device and interface roles on the interface where the configuration policy is applied.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the StaticNATsConfigurationPolicyExample file in the samples directory.

Modifying a Configuration Policy

Modification of a configuration policy involves querying the object model for the current configuration policy definition, modifying the configuration policy, and updating the whole definition back into the object model.

Querying the EOM for the Configuration Policy

The configuration policy XML can be obtained from the RuleGeneric ContentValue parameter. The XML is parsed back into the XML Beans object definition of the service model extension.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the StaticNATsConfigurationPolicyExample file in the samples directory.

As with creating the configuration policy, the XML content of RuleGeneric is updated to handle the extra escape characters around the \" and \' characters. This conversion is performed by the **unescapeFromOIM()** function.

Modifying the Policy Definition

The configuration policy definition is modified using the XML Bean API for the service model extension documents.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the StaticNATsConfigurationPolicyExample file in the samples directory.

Registering an Interface Policy

Creating a new interface in IP Service Activator through the OIM and OJDL APIs involves a specialized use of configuration policies with the interface configuration management framework. As with interface management through IP Service Activator, there are three types of interface management interactions:

- Main interface creation
- Subinterface creation
- Interface decoration

Each possible interaction must be registered as an Interface Policy Registration. The Interface Policy Registration objects can either be pre-configured in IP Service Activator, or created using the IP Service Activator APIs.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the nterfaceManagementPolicyExample file in the samples directory.

Once an Interface Policy Registration is used to create or decorate an interface it cannot be modified or deleted until all dependent parent interfaces have been deleted or unlinked from the policy registration.

Creating a Subinterface

This section describes how to create a new interface in IP Service Activator so that the new interface configuration will also be correctly provisioned on the device.

As a prerequisite the appropriate subinterface creation Interface Policy Registration must be created.

Creating the Subinterface Object

Create a new subinterface object under the target interface. For consistency, it is recommended that you create the child subinterface with the correct ifType, although IP Service Activator will update this value on the next device discovery.

The following example shows the creation of a new subinterface:

```
// Create the new subinterface interface object
String subinterfaceName = "Serial1/3.100";
attributes = new EomAttributesSet();
attributes.setAttribute("Type", "32");
EomObject subinterface = tr.createObject(parentInterface, "Subinterface",
  subinterfaceName, attributes);
```

Linking the New Subinterface Object to the Interface Policy Registration

The created subinterface object is linked to the previously defined Interface Policy Registration. The act of linking the policy registration automatically creates a new RuleGeneric configuration policy object with the correct data type settings based on the Interface Policy Registration definition.

The following example shows the linking of the subinterface object with the interface policy registration:

```
// Link the new subinterface object to the interface policy registration
tr.linkObjects(subinterface, registrationPolicy);
tr.commit();
```

The new RuleGeneric object name consists of the interface names with **-Data** appended to it.

Modifying the Interface Configuration Policy Data

The interface management configuration policy does not contain any default settings. These must be manually created using the appropriate XML data structure for the configuration policy data type defined in the interface registration policy. The XML content can be created manually or using the XML Beans API provided.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the InterfaceManagementPolicyExample file in the samples directory.

Linking the New Subinterface to an Interface Role

Up to this point the new subinterface has only been created in the IP Service Activator object model. Before committing the subinterface creation to the device, the new subinterface object must be linked to an appropriate interface role.

There are code examples available in the additional documentation included with the OJDL libraries. For more information, see the InterfaceManagementPolicyExample file in the samples directory.

Optionally Discovering the Device

Optionally, the device can be re-discovered to align any interface changes (such as to the ifType or VC objects) with the object model. For interface types that have child VC

object created by the configuration (such as the framerelay DLCI) device re-discovery is recommended.

The following example shows the optional device discovery:

```
// Optionally rediscover the device the get any VC level objects (in this example
// the DLCI)
eomSession.sendCommandtoOIM("discover " + parentDeviceId);
```

Creating a Main Interface

The steps for main interface creation are largely the same as for subinterface creation. For a main interface, the new interface is created as a child of the device and is linked to an appropriate Interface type Interface Policy Registration object.

When creating a main interface, the Interface Policy Registration must define the default capabilities that the interface (and its sub-interfaces and VCs) will be assigned. If the default settings are used, the created interface will not have any capabilities assigned and a capabilities reset and re-discovery must be performed instead.

Decorating an Interface

For interface decoration, follow the same steps as with subinterface creation, with the exception that the interface does not need to be created first. For interface decoration, the existing interface must be linked to a **Decorate** type Interface Policy Registration object.

Comparing Created and Discovered Interfaces

It is possible to determine if an interface was created using the IP Service Activator interface configuration management framework or was initially discovered from the device by inspecting the IsConfigurable parameter on the Interface or SubInterface object.

If IsConfigurable is set to **True** then the interface was created within IP Service Activator. If it is set to **False** then the interface was added through discovery.

Configuration Policy Classes

Table A–1 lists the configuration policy classes.

Table A–1 Configuration Policy Classes

Extension	Configuration Policy	Java XMLBeans Class
AlcatelSamAccessInterfaceModule	alcatelSRL3Interface	com.metasolv.serviceactivator.alcatelSamAccessInterface.AlcatelSRL3InterfaceDocument
AlcatelSamAggregatedSchedulerModule	alcatelSamAggregatedScheduler	com.metasolv.serviceactivator.alcatelSamAggregatedScheduler.AlcatelSamAggregatedSchedulerDocument
AlcatelSamQosOverridePolicyModule	alcatelSamQosOverridePolicy	com.metasolv.serviceactivator.alcatelSamQosOverridePolicy.AlcatelSamQosOverridePolicyDocument
alcatelSamQosPoolModule	alcatelSamQosPool	com.metasolv.serviceactivator.alcatelSamQosPool.AlcatelSamQosPoolDocument
alcatelSamSchedulerOverridePolicyModule	alcatelSamSchedulerOverridePolicy	com.metasolv.serviceactivator.alcatelSamSchedulerOverridePolicy.AlcatelSamSchedulerOverridePolicyDocument

Table A–1 (Cont.) Configuration Policy Classes

Extension	Configuration Policy	Java XMLBeans Class
AtmPvcVcClassModule	atmPvcVcClass	com.metasolv.serviceactivator.atmpvcvcclass.AtmPvcVcClassDocument
CatOSPolicingRuleModule	catOSPolicingRule	com.metasolv.serviceactivator.catospolicingrule.CatOSPolicingRuleDocument
CiscoEthernetPortCharacteristicsModule	ciscoEthernetPortCharacteristics	com.metasolv.serviceactivator.ciscoEthernetPortCharacteristics.CiscoEthernetPortCharacteristicsDocument
CiscoQosPfcTxPortQueuesModule	ciscoQosPfcTxPortQueues	com.metasolv.serviceactivator.ciscoqospfctxportqueues.CiscoQosPfcTxPortQueuesDocument
DlswModule	dlswDevice	com.metasolv.serviceactivator.dlsw.DlswDeviceDocument
DlswModule	dlswEthernetInterface	com.metasolv.serviceactivator.dlsw.DlswEthernetInterfaceDocument
DlswModule	dlswTokenRingInterface	com.metasolv.serviceactivator.dlsw.DlswTokenRingInterfaceDocument
InterfaceConfigMgmtModule	atmSubInterfaceData	com.metasolv.serviceactivator.subinterface.AtmSubInterfaceDataDocument
InterfaceConfigMgmtModule	backUpInterfacePolicy	com.metasolv.serviceactivator.subinterface.BackUpInterfacePolicyDocument
InterfaceConfigMgmtModule	basicRateInterfaceData	com.metasolv.serviceactivator.subinterface.BasicRateInterfaceDataDocument
InterfaceConfigMgmtModule	ciscoUniversalInterface	com.metasolv.serviceactivator.subinterface.CiscoUniversalInterfaceDocument
InterfaceConfigMgmtModule	dialerInterface	com.metasolv.serviceactivator.subinterface.DialerInterfaceDocument
InterfaceConfigMgmtModule	e1ChannelizedSerialInterface	com.metasolv.serviceactivator.subinterface.E1ChannelizedSerialInterfaceDocument
InterfaceConfigMgmtModule	e1Controller	com.metasolv.serviceactivator.controller.E1ControllerDocument
InterfaceConfigMgmtModule	e3ChannelizedSerialInterface	com.metasolv.serviceactivator.subinterface.E3ChannelizedSerialInterfaceDocument
InterfaceConfigMgmtModule	e3Controller	com.metasolv.serviceactivator.controller.E3ControllerDocument
InterfaceConfigMgmtModule	frSubInterfaceData	com.metasolv.serviceactivator.subinterface.FrSubInterfaceDataDocument
InterfaceConfigMgmtModule	hsrp	com.metasolv.serviceactivator.hsrp.HsrpDocument
InterfaceConfigMgmtModule	loopbackInterfaceData	com.metasolv.serviceactivator.subinterface.LoopbackInterfaceDataDocument
InterfaceConfigMgmtModule	multilinkInterface	com.metasolv.serviceactivator.subinterface.MultilinkInterfaceDocument
InterfaceConfigMgmtModule	plPosInterfaceData	com.metasolv.serviceactivator.subinterface.PlPosInterfaceDataDocument
InterfaceConfigMgmtModule	pppMultilink	com.metasolv.serviceactivator.subinterface.PppMultilinkDocument

Table A–1 (Cont.) Configuration Policy Classes

Extension	Configuration Policy	Java XMLBeans Class
InterfaceConfigMgmtModule	stm1ChannelizedSerialInterface	com.metasolv.serviceactivator.subinterface.Stm1ChannelizedSerialInterfaceDocument
InterfaceConfigMgmtModule	stm1Controller	com.metasolv.serviceactivator.controller.Stm1ControllerDocument
InterfaceConfigMgmtModule	t1ChannelizedSerialInterface	com.metasolv.serviceactivator.subinterface.T1ChannelizedSerialInterfaceDocument
InterfaceConfigMgmtModule	t1Controller	com.metasolv.serviceactivator.controller.T1ControllerDocument
InterfaceConfigMgmtModule	t3ChannelizedSerialInterface	com.metasolv.serviceactivator.subinterface.T3ChannelizedSerialInterfaceDocument
InterfaceConfigMgmtModule	t3Controller	com.metasolv.serviceactivator.controller.T3ControllerDocument
InterfaceConfigMgmtModule	virtualTemplateInterface	com.metasolv.serviceactivator.subinterface.VirtualTemplateInterfaceDocument
InterfaceConfigMgmtModule	vlanSubInterface	com.metasolv.serviceactivator.subinterface.VlanSubInterfaceDataDocument
InterfaceConfigMgmtModule	vrfExportRouteFilter	com.metasolv.serviceactivator.vrfexportroutefilter.VrfExportRouteFilterDocument
IpsecModule	IPsecModule	com.metasolv.serviceactivator.ipsecmodule.IpsecmoduleDocument
LspModule	lspTunnel	com.metasolv.serviceactivator.lsp.LspTunnelDocument
L2QosModule	rateLimit	com.metasolv.serviceactivator.l2Qos.RateLimitDocument
JuniperQosCosAttachmentModule	juniperQosCosAttachment	com.metasolv.serviceactivator.juniperqoscosattachment.JuniperQosCosAttachmentDocument
MiscPluginsModule	atmVcClass	com.metasolv.serviceactivator.vcclass.AtmVcClassDocument
MiscPluginsModule	banners	com.metasolv.serviceactivator.banner.BannersDocument
MiscPluginsModule	bgpCE	com.metasolv.serviceactivator.bgpce.BgpCEDocument
MiscPluginsModule	dailerList	com.metasolv.serviceactivator.dialerList.DialerListDocument
MiscPluginsModule	dslInterfaceData	com.metasolv.serviceactivator.subinterface.DslInterfaceDataDocument
MiscPluginsModule	extendedAcl	com.metasolv.serviceactivator.extendedAcl.ExtendedAclDocument
MiscPluginsModule	ipPools	com.metasolv.serviceactivator.ippool.IpPoolsDocument
MiscPluginsModule	keyChains	com.metasolv.serviceactivator.keyChain.KeyChainsDocument
MiscPluginsModule	saveConfig	com.metasolv.serviceactivator.saveConfig.SaveConfigDocument
MiscPluginsModule	staticNats	com.metasolv.serviceactivator.staticnat.StaticNatsDocument

Table A–1 (Cont.) Configuration Policy Classes

Extension	Configuration Policy	Java XMLBeans Class
MiscPluginsModule	staticRoutes	com.metasolv.serviceactivator.staticroute.StaticRoutesDocument
MiscPluginsModule	userAuth	com.metasolv.serviceactivator.userAuth.UserAuthDocument
MiscPluginsModule	userData	com.metasolv.serviceactivator.userData.UserDataDocument
MulticastModule	multicastAutoRp	com.metasolv.serviceactivator.multicast.MulticastAutoRpDocument
MulticastModule	multicastBootstrapRouter	com.metasolv.serviceactivator.multicast.MulticastBootstrapRouterDocument
MulticastModule	multicastDevice	com.metasolv.serviceactivator.multicast.MulticastDeviceDocument
MulticastModule	multicastInterface	com.metasolv.serviceactivator.multicast.MulticastInterfaceDocument
MulticastModule	multicastVrf	com.metasolv.serviceactivator.multicast.MulticastVrfDocument
PrefixListModule	prefixListEntries	com.metasolv.serviceactivator.prefixlist.PrefixListEntriesDocument
QosCosAttachmentModule	qosCosAttachment	com.metasolv.serviceactivator.qoscosattachment.QosCosAttachmentDocument
RoutePolicyModule	bgpRoutePolicy	com.metasolv.serviceactivator.routePolicy.BgpRoutePolicyDocument
RoutePolicyModule	vrfRoutePolicy	com.metasolv.serviceactivator.routePolicy.VrfRoutePolicyDocument
SubInterfaceModule	plSerialInterfaceData	com.metasolv.serviceactivator.subinterface.PlSerialInterfaceDataDocument
ServiceAssuranceModule	collectorParameters	com.metasolv.serviceactivator.collectorParameters.CollectorParametersDocument
ServiceAssuranceModule	netflowParameters	com.metasolv.serviceactivator.netflowParameters.NetflowParametersDocument
ServiceAssuranceModule	rtrResponder	com.metasolv.serviceactivator.rtrr.RtrResponderDocument
SgbpModule	sgbp	com.metasolv.serviceactivator.sgbp.SgbpDocument
SnmpModule	snmpCommunities	com.metasolv.serviceactivator.snmp.SnmpCommunitiesDocument
SnmpModule	snmpHosts	com.metasolv.serviceactivator.snmp.SnmpHostsDocument
VlanModule	vlanDefinitions	com.metasolv.serviceactivator.vlanModule.VlanDefinitionsDocument
VlanInterfaceModule	mgmtVlanInterface	com.metasolv.serviceactivator.vlanInterface.MgmtVlanInterfaceDocument
VlanInterfaceModule	vlanInterface	com.metasolv.serviceactivator.vlanInterface.VlanInterfaceDocument

Table A–1 (Cont.) Configuration Policy Classes

Extension	Configuration Policy	Java XMLBeans Class
VrfCustomNamingModule	vrfCustomNaming	com.metasolv.serviceactivator.vrfCustomNaming.VrfCustomNamingDocument
VrfIPsecModule	customerIPsec	com.metasolv.serviceactivator.vrfipsec.CustomerIPsecDocument
VrfIPsecModule	publicIPsec	com.metasolv.serviceactivator.vrfipsec.PublicIPsecDocument

Example Source Code

There are code examples available in the additional documentation included with the OJDL libraries.

For configuration policy example source code, see the StaticNATsConfigurationPolicyExample file in the samples directory.

For interface management example source code, see the InterfaceManagementPolicyExample file in the samples directory.