Oracle® Communications Design Studio

Modeling Activation

Release 7.3.1

E65404-01

December 2015

ORACLE®

Oracle Communications Design Studio Modeling Activation, Release 7.3.1

E65404-01

Copyright © 2006, 2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

1 Getting Started with Design Studio for ASAP

2 Working with Design Studio for IP Service Activator

3 Working with Activation Cartridge Projects

4 Modeling Activation Cartridge Project Data

5 Modeling ASAP Services

7 Working with Network Elements

8 Packaging and Deploying Activation Cartridges

9 Testing ASAP Cartridges in Design Studio

10 Documenting Cartridges

About Design Studio Help for Modeling Activation

This document describes how to model activation in Oracle Communications Design Studio.

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.

1

Getting Started with Design Studio for ASAP

Oracle Communications Design Studio for ASAP has two components (Activation and Service Request Translation) that simplify the creation, assembly, and deployment of services across multiple domains. It is a powerful graphical tool that dramatically reduces the time-to-market for new services. Design Studio offers support for the activities performed by a wide variety of users including solution designers, network element administrators, release engineers, and cartridge developers. Design Studio functionality includes:

- Create, deploy and manage cartridges.
- Extend cartridges into customer specific service configurations.
- Manage and deploy complex multi-domain services to production, test and development environments.
- Rapidly model network element (NE) instances using pre-defined network element instance and connection attributes.
- Support for creation and deployment of patches.

The services that can be modeled in Design Studio are typically those that are provided to end users of telecommunications networks, such as voice services (including wireless, voice over IP), data services (including digital subscriber line, IPTV), or any other services that require controlled and coordinated activation in the network. These services are modeled in Design Studio by defining and relating objects at different levels of abstraction.

When getting started with Design Studio for ASAP, see the following topics:

- About Design Studio for ASAP Users and Tasks
- Configuring Activation Network Cartridge Projects
- Configuring Activation Service Cartridge Projects
- Defining Design Studio Activation Preferences

About Design Studio for ASAP Users and Tasks

The following table lists the roles and the tasks each role typically performs in Oracle Communications Design Studio for ASAP.

Role	Task	Reference
Cartridge Designer	Setting up an Activation Network Cartridge	"Creating New Activation Cartridge Projects"

Role	Task	Reference
Cartridge Designer	Defining and editing elements (service actions, atomic actions, action processors)	"Creating Model Elements" "Creating Activation Run-Time Type Parameters in the Data Dictionary" "Understanding Model Element Relationships"
Cartridge Designer	Defining relationships between elements	"Understanding Model Element Relationships" "Generating Framework Models"
Solution Designer	Load (import) cartridges	"Importing Projects" "Importing Activation Cartridges from SAR Files" "Importing Cartridges from Environments"
Solution Designer	Setting up an Activation Service Cartridge	"Creating New Activation Cartridge Projects"
Solution Designer	Designing a service model	"About Vendor, Technology, and Software Load-Specific Service Models" "About Common Service Models" "About Common Service Models" "Creating Model Elements" "Creating Activation Run-Time Type Parameters in the Data Dictionary" "Understanding Model Element Relationships" "Generating Framework Models"
Solution Designer	Extending user-defined exit types	"Extending User-Defined Exit Types"
Solution Designer	Creating custom action processors	"Creating Custom Action Processors"
Solution Designer	Configuring network elements	"Working with Network Elements"
Solution Designer	Build, deploy, and undeploy to or from development environments	"Packaging and Deploying Activation Cartridges"
Developer	Developing the implementation	"Working with the Action Processor" "Working with User-Defined Exit Types" "Working with Java NE Connection Handlers"
Developer	Configuring sample network elements	"Working with Network Elements"

Role	Task	Reference
Developer	Build, deploy, and undeploy to or from development environments	"Packaging and Deploying Activation Cartridges"
Release Engineer	Build, deploy, and undeploy to or from test and production environments	"Packaging and Deploying Activation Cartridges"

Related Topics

Getting Started with Design Studio for ASAP

Configuring Activation Network Cartridge Projects

You can create and configure Activation Network cartridges. See "Understanding Design Studio for Activation Projects".

To configure an Activation Network cartridge:

1. Create an Activation Cartridge project and set up the Activation Network cartridge.

   a. Use the Activation Network cartridge project wizard to create an Activation Network Cartridge project and display it in the Studio Projects view of the Design Perspective. See "Creating New Activation Cartridge Projects" for more information.

   b. Configure the cartridge details in the Activation Network Project editor. See "Activation Project Editor" for more information.

2. Define and edit the elements.

   a. Create service actions, atomic actions, action processors, and activation run-time parameters as elements for the model. See "Creating Model Elements" and "Creating Activation Run-Time Type Parameters in the Data Dictionary"for more information.

      You can create elements with a wizard or with the Cartridge Generation feature, which creates and links the elements. See "Generating Framework Models" for more information.

   b. Do any one of the following:

      For elements that you created with a wizard, define their relationships by manually linking service actions, atomic actions, and action processors, and define the element parameters.

      For elements that you created with the Cartridge Generation feature, the linking process is automated, and elements are automatically created and linked.

      See "Understanding Model Element Relationships" for more information.

   c. Add information that will be used for auto-generation of documentation for the model.

      See "Documenting Cartridges" for more information.

3. Implement the action processor.

   a. Define a Java action processor implementation.

      Use the **Java with code generation** implementation to write the logic in the execute method of the processor class (the proxy automatically performs several steps of code generation). See "Understanding Java with Code Generation" and "Understanding the Java Processor Class" for more information.

      Configure Java libraries. See "Understanding Java Libraries in Design Studio" for more information.

   b. Use the Unit Test procedure to generate a test case, which enables you to test the processor outside of the ASAP environment.

      Configure a Unit test for the processor. See "Testing ASAP Cartridges in Design Studio".

4. Configure the user-defined exit types (UDET). See "Configuring User-Defined Exit Types" for more information.

   a. Create UDETs as elements for the model.

   b. Configure the UDETs in the editor to define the content.

5. Implement the connection handler.

   Write a Java Connection Handler. See "Generating a Telnet NE Connection Handler Implementation" and "Generating a Custom NE Connection Handler Implementation" for more information.

6. Define the network element model. See "Working with Network Elements" for more information.

   You must create and configure at least one of the following elements (all three elements are related as they involve connection to equipment):

   - NE templates.

     A template that can be copied to create one or many specific network elements.

   - Network elements.

     A network element represents one specific piece of equipment (a single instance) in the network. A connection pool contains one or more connections that can be used to connect to the network element (possibly simultaneously). Each network element has a single connection pool associated with it.

   - Dynamic NE templates.

     In some cases it is not ideal to configure static network element instances for a specific customer solution. In such cases, dynamic NE templates can be configured allowing network element attributes to be dynamically sent to ASAP 5.x on work orders. Dynamic NE templates are used when upstream systems (such as Inventory) contain the necessary information to connect to the network element instance. Passing this information to ASAP 5.x dynamically avoids having to configure it in multiple locations (for example, in ASAP 5.x as well as in an inventory system).

7. Package the cartridge.

   See "Packaging Activation Cartridges" for more information.

   a. Model the packaging.

Use the Project editor to specify which elements will be included in the cartridge (SAR file).

   b. Create the JAR with ANT.

   c. Include JARs in the SAR.

   d. Put external JARs in the NEP classpath on the ASAP server.

8. Deploy the cartridge.

   See "Deploying Cartridge Projects" for more information.

   a. Create a Studio Environment project and a Studio environment in the Studio Projects view (use corresponding wizards for both the tasks).

   b. On the **Connection Information** tab of the Studio Environment editor, specify how to connect to the activation environment. See "Studio Environment Editor Activation Connection Details Area" for more information.

      The NEP Map editor and Activation Test case utility use the information specified in the Activation Connection Details area to deploy network element configuration and to submit test work orders to a run-time ASAP environment respectively.

   c. In the Cartridge Management view, deploy cartridges to the run-time environment, and undeploy them from the run-time environment.

   d. Use the NEP Map editor to deploy and manage network elements.

Related Topics

Getting Started with Design Studio for ASAP

Configuring Activation Service Cartridge Projects

You can create and configure Activation Service cartridges. See "Understanding Design Studio for Activation Projects".

To configure an Activation Service cartridge:

1. Import the Activation Network cartridges that contain the entities you need for your Service cartridge. See "Importing Projects", "Importing Activation Cartridges from SAR Files", and "Importing Cartridges from Environments" for more information.

   See "Importing Projects" for more information.

2. Create an Activation Cartridge project and set up the Activation Service cartridge. See "Creating New Activation Cartridge Projects" for more information.

   a. Create an Activation Service Cartridge project and display it in the Studio Projects view of the Design Perspective.

   b. Configure the cartridge details in the Activation Service Project editor.

3. Design the service model.

   a. Determine what type of service model you need. Options are:

      The vendor, technology and software load-specific service model. See "About Vendor, Technology, and Software Load-Specific Service Models" for more information.

      The common service model. See "About Common Service Models" for more information.

The mixed service model. See "About Mixed Service Models" for more information.

b. Create elements for the service model.

   You can create service actions with a wizard or with the Cartridge Generation feature. You can create the necessary atomic actions with a wizard (this is called a common service model) or use atomic actions from imported Activation Network cartridges (this is called a mixed service model). See "Creating Model Elements" and "Generating Framework Models" for more information.

c. Define the relationship between model elements by linking them manually and defining their parameters.

   See "Understanding Model Element Relationships" for more information.

4. Extend the user-defined exit types.

   See "Configuring User-Defined Exit Types" for more information.

5. Create custom action processors.

   See "Working with Custom Action Processors" for more information.

6. Configure network elements.

   a. Define a network element.

      A network element represents one specific piece of equipment (a single instance) in the network. A connection pool contains one or more connections that can be used to connect to the network element (possibly simultaneously). Each network element has a single connection pool associated with it. See "About Network Elements" for more information.

   b. Define a dynamic NE template.

      If you do not intend to configure static network element instances for a specific customer solution, you can configure dynamic NE templates to allow network element attributes to be dynamically sent to ASAP 5.x on work orders. Dynamic NE templates are used when upstream systems (such as Inventory) contain the necessary information to connect to the network element instance. Passing this information to ASAP 5.x dynamically avoids having to configure it in multiple locations (for example, in ASAP 5.x as well as in an inventory system). See "Creating NE Templates" for more information.

7. Package the cartridge.

   See "Packaging Activation Cartridges" for more information.

   a. Model the package.

      Use the Project editor to specify which elements will be included in the cartridge (SAR file). If the service model you have created depends upon elements in an Activation Network cartridge it is also necessary to specify which elements of the Activation Network cartridge should be deployed. Only deploy those elements that are required in the dependent Activation Network cartridges. For example, if you are not reusing the service actions and atomic actions in an Activation Network cartridge, then deselect those elements.

   b. Create the JAR with ANT.

   c. Include JARs in the SAR.

   d. Put external JARs in the NEP classpath on the ASAP server.

8. Deploy the cartridge.

   See "Deploying Cartridge Projects" for more information.

   a. Create a Studio Environment project and a Studio environment in the Studio Projects view (use corresponding wizards for both the tasks).

   b. On the **Connection Information** tab of the Studio Environment editor, specify how to connect to the activation environment. See "Studio Environment Editor Activation Connection Details Area" for more information.

      The NEP Map editor and Activation Test case utility use the information specified in the Activation Connection Details area to deploy network element configuration and to submit test work orders to a run-time ASAP environment respectively.

   c. In the Cartridge Management view, deploy cartridges to the run-time environment, and undeploy them from the run-time environment.

   d. Use the NEP Map editor to deploy and manage network elements.

Related Topics

Getting Started with Design Studio for ASAP

Defining Design Studio Activation Preferences

To define activation preferences, from the **Window** menu, select **Preferences**, then expand **Oracle Design Studio** in the Preferences navigation tree, and then select **Activation Preferences**.

Field	Use
Allow NE Template Deployment	Select this check box to cause Design Studio to generate artifacts for NE templates and package them in the .sar file. If this check box is not selected, NE Templates will not appear as an available option in the Activation Project editor **Packaging** tab. This check box is selected by default.
Service Model Package Format	These options determine how the XML files are generated when the activation cartridge is built. Do one of the following: ▪ Select **Single file per type** if you would like all of the service actions to be packaged into the **cartridgeBuild/ServiceModel/CommonService.xml** file and all of the atomic actions to be packaged into the **cartridgeBuild/ServiceModel/AtomicService.xml** file. ▪ Select **Single file per instance** if you would like separate XML files for each of the atomic actions and service actions in the **cartridgeBuild/model** directory. The default option is **Single file per instance**.
Complex Type Delimiter	Enter the character to use to separate the parent and child elements of a "scalars" runtime-type parameter. Valid values are period (**.**), hyphen (**-**), and underscore (**_**). The default is underscore. See "Grouping Scalar Parameters using Structured Elements" for more information about scalars parameters.

Field	Use
Allow CSDL Label Overwrite	Select this check box if you want to be able to overwrite the CSDL labels (Service Action Label) for data schema elements. The **Service Action Label** field is available on the **Activation** tab of the data schema element. By default, this check box is not selected and, in this case, the **Service Action Label** field is read-only and is the same as the **Atomic Action Label**.
CSDL Primitive Type Merge Diagnostic Level	If two or more atomic actions are using the same data element name with a different type, and are contributing to a single CSDL, Design Studio attempts to resolve the primitive type merge for the data element in the CSDL. If Design Studio is unable to resolve the merge type, it diagnoses the issue as either an error or a warning. Do one of the following: ■ Select **Error**. ■ Select **Warning**. The default option is **Error**. The message is displayed in Design Studio **Problems** view.

Related Topics

Getting Started with Design Studio for ASAP

2

Working with Design Studio for IP Service Activator

You use Oracle Communications Design Studio to generate fully-formed service actions for Oracle Communications IP Service Activator from CTM Templates.

About CTM Templates

CTM templates define service actions. You can import the CTM template into Design Studio or you can load the CTM template into the IP Service Activator CTM server.

A CTM template is a valid XML file. See *IP Service Activator Concepts* for more information.

Import the CTM template into Design Studio if you want to use it as part of your IP Service Activator flow-through. Design Studio generates a service action when the template is imported, which you can then add to Activation Tasks. Each CTM Template contains one service action.

Load the CTM Template to the CTM Server if you want to invoke it from the IP Service Activator UI.

For more information about creating CTM Templates or invoking them from the IP Service Activator UI, see *IP Service Activator Concepts* and *IP Service Activator System Administrator's Guide*.

Before you can import a CTM Template into Design Studio, you must already have an activation IP Service Activator project.

Related Topics

Creating an Activation IP Service Activator Project

Importing CTM Templates into Design Studio

Working with Activation Tasks

Creating an Activation IP Service Activator Project

You must create an activation IP Service Activator project in Design Studio before you can import CTM Templates.

To create an Activation IP Service Activator project:

1. Switch to the Studio Projects view.
2. From the **Studio** menu, select **New**, then **Project**, then **Activation IPSA Project**.

   The Activation IPSA Cartridge Project wizard appears.

3. Do the following:

   a. In the **Project** name field, enter a name for your project.

   b. (Optional) Deselect **Use default location** and specify a location to save your project. By default, Design Studio saves the project to your default workspace.

   c. (Optional) In the **Package** field, enter the package name.

   > **Note:** There is no need to modify the Java settings for activation IP Service Activator projects.

4. Click **Finish**.

   The activation IP Service Activator project is displayed in the Studio Projects view. You can now import CTM templates into Design Studio.

Related Topics

About CTM Templates

Importing CTM Templates into Design Studio

Importing CTM Templates into Design Studio

Importing CTM templates into Design Studio allows you to use the generated service actions in your activation tasks for IP Service Activator.

This procedure requires an activation IP Service Activator project in Design Studio and assumes you already have a valid CTM template file.

To import a CTM template into Design Studio:

1. Switch to the Studio Projects view.
2. Right-click in the Studio Projects view.
3. From the context menu, click **Import** and select **IPSA CTM Template**.

   The IPSA CTM Template Import Wizard dialog box appears.

4. Do all the following:

   - From the **Project** list, specify into which activation IP Service Activator project you want to import the CTM template.
   - Click **Browse** next to the **CTM Template** field.

     The Select CTM Template XML dialog box appears.

   - Navigate to the CTM Template file and click **OK**.

5. Click **Finish**.

   The IPSA CTM Template Import Wizard generates the following entities in the specified activation IP Service Activator project:

   - atomic action
   - CTM template
   - service action

   These entities are fully modeled and cannot be modified in Design Studio. The Design Studio editors for these entities are read only.

6. Add the activation IP Service Activator project as a dependency to an Order and Service Management cartridge project to use the generated service actions in an activation task.

Related Topics

Working with Activation Tasks

Creating an Activation IP Service Activator Project

About CTM Templates

IP Service Activator Atomic Action Editor

Service Action Editor

CTM Template Editor

IP Service Activator Atomic Action Editor

The Atomic Action editor for atomic actions generated by Design Studio from CTM templates for IP Service Activator is read only and shows the action parameters and their definition.

> **Note:** For information about modeling activation atomic actions, or for a description of the Atomic Action editor subtabs and fields, see Atomic Action Editor.

CTM Template Editor

Use the CTM Template editor **Blueprint** tab to view the generated documentation for the CTM template entity.

3 Working with Activation Cartridge Projects

Oracle Communications Design Studio allows you to create Activation cartridge projects or import data or components and modify them.

When working with Activation Cartridge projects, see the following topics:

- Understanding Design Studio for Activation Projects
- Creating New Activation Cartridge Projects
- Importing Activation Cartridge Projects
- Activation Project Editor

Understanding Design Studio for Activation Projects

Design Studio supports three types of Activation cartridges:

- Activation Network cartridge projects, required for creation of an Activation Network cartridge. See "About Activation Network Cartridge Projects" for more information.
- Activation Service cartridge projects, required for creation of an Activation Service cartridge. See "About Activation Service Cartridge Projects" for more information.
- Activation SRT cartridge projects, required for creation of an Activation SRT cartridge. See "Understanding ASAP SRT Cartridges" for more information.

Additionally, an Environment perspective is required for deployment of all Activation cartridges.

When working with Design Studio for Activation cartridges, see the following topics:

- About Activation Network Cartridge Projects
- About Activation Service Cartridge Projects
- About Cartridge Project Upgrades

About Activation Network Cartridge Projects

Developers can create new Activation Network cartridges in Design Studio to support a single type of network equipment (for example, for a DMS-100 switch), or import existing cartridges into Design Studio and modify them. You configure Activation Network cartridges for a single vendor, technology, and software load. Typically, a cartridge developer will deliver Activation Network cartridges to solution designers and solution teams, who use the Activation Network cartridge components to create customer-specific service models.

Related Topics

Working with Activation Cartridge Projects

About Activation Service Cartridge Projects

Solution designers develop Activation Service cartridges to create a service model that can activate services on different types of network equipment, as the service actions and atomic actions within the Activation Service cartridge may be missing one or more of the vendor, technology and software load tokens. Modelers can use elements of Activation Network cartridges in Activation Service cartridges for implementation when creating common service models. Solution designers purchase and then import Activation Network cartridges, to build one or more Activation Service cartridges that link down into components within network cartridges and that are specific to an offered service. Service cartridges incorporate the many different types of equipment used to set up and provide telephony services.

You can use Activation Service cartridges to create customer-specific service models. These cartridges can contain customer-specific service modeling elements as well as links to elements in other cartridges. While an Activation Network cartridge always has the three associated attributes (a vendor, such as Ericsson; a technology, such as DMS; and a software load, which references the release number), an Activation Service cartridge does not necessarily contain all these attributes. Activation Service cartridges have elements that generally span multiple types of vendor equipment, can run multiple software loads, and may or may not include one or more of the Activation Network cartridge attributes.

When creating an Activation Service cartridge, there are two required attributes that you must specify: a service attribute (for example, Prepaid) and a Domain attribute (for example, Mobile).

Related Topics

Working with Activation Cartridge Projects

About Cartridge Project Upgrades

When legacy cartridge projects, developed using earlier Design Studio versions, are imported into Design Studio or when the workbench is loaded, the projects are upgraded to the new project format. The new project format is incompatible with earlier Design Studio for Activation versions. The conversion is one-way using the Design Studio Platform's Project Upgrade wizard. See *Design Studio Installation Guide* for more information.

Design Studio's project upgrade logic is responsible for identifying and orchestrating the conversion of interdependent projects. Where several projects with interdependencies need to be converted, the Design Studio Platform upgrade logic performs the conversion in a correct order. For example, an SRT project that depends on an Activation Service project, which depends on an Activation Network project, will first convert the Activation Network project, then the Activation Service project then the SRT project.

Creating New Activation Cartridge Projects

You can use the New Studio Activation Cartridge Project wizard to set up Activation Network and Activation Service cartridge projects and to specify cartridge attributes (vendor, technology, and software load). After you create the cartridge project, you can add additional details using the Project editor.

To create Activation cartridge projects:

1. Select **Studio**, then select **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Project**, then select **Activation Cartridge Project**. The New Studio Activation Cartridge Project wizard appears, displaying the Activation Cartridge Info dialog box.
3. In the **Project name** field, enter a name for the project.
4. (Optional) Select a location for the project.

   By default, Oracle Communications Design Studio saves the project to your default workspace location. To identify a location different from the system-provided default:

   a. Deselect **Use default location**.

   b. Click B**rowse.**

   c. Navigate to the directory in which to save the project.

   d. Click **OK.**
5. In Studio Settings, for **Cartridge Type**, select **Activation Network Cartridge** or **Activation Service Cartridge**, as appropriate.
6. Select the appropriate target version, and enter the package name.
7. Click **Next**.

   The Cartridge Details dialog box appears.
8. Do any one of the following:
   - For Activation Network cartridge projects, specify the vendor, technology and software load attributes for the cartridge.

     When specifying vendors, for example, you might use its stock symbol, such as ERIC (for Ericsson). Examples of technology include HLR (home location register) and DMS (digital multiplex system). You might represent software loads with the release number and dash combination, such as R11-0.
   - For Activation Service cartridge projects, specify the service and domain attributes for the cartridge.

     Examples of services:

     Service: Prepaid Domain: Mobile, Service: Postpaid Domain: Mobile, Service: Residential Domain: POTS, Service: VPN Domain: IP.
9. (Optional) Click **Next**.

   The Create Default Routing Schema dialog box appears. The dialog box appears when for the first time an Activation cartridge project is created in a workspace. A default Data Dictionary containing default routing parameters is created. For example, a default routing parameter like ID_ROUTING for ID Routing type.
10. (Optional) Click **Next**.

    You can modify the Java configuration in the Java Settings dialog box. For example, you can add libraries in the **Libraries** tab (you can decide to change the configuration later).

    If you modified the Java settings, click **Finish** to save changes and complete the cartridge project.
11. Click **Finish**.

A new Activation Network cartridge or Activation Service cartridge project appears in the Studio Projects view. The project contains an entity that represents the network cartridge and also has a Documentation group (which provides access to the Cartridge Guide generation feature for documenting the completed cartridge). See "Documenting Cartridges" for more information. Along with the Activation Network cartridge or Activation Service cartridge, an ActivationRoutingDictionary project is also created in the view.

Related Topics

Working with Activation Cartridge Projects

Importing Activation Cartridge Projects

You can import data from an external source into Design Studio workspace or load earlier versions of cartridge projects with legacy data definitions. This allows you to utilize the imported data or components and modify based on requirement.

When importing Activation cartridge projects into Design Studio, see the following topics:

- About Imported Activation Cartridge Projects
- Importing Activation Cartridges from SAR Files
- Importing Cartridges from Environments

About Imported Activation Cartridge Projects

The import functions are used to import data from an external source into your Design Studio workspace. For example, if you have purchased cartridges from Oracle, you can import them into Design Studio and reuse their components when you build Activation Service cartridges. Additionally, if an Activation Service cartridge has already been created by a coworker, you can obtain the configuration by importing it into your Design Studio workspace. Lastly, you can load data directly from an Oracle Communications ASAP environment. This approach is useful when you want to obtain a complete snapshot of all the data in an ASAP environment.

Design Studio for Activation supports loading of earlier versions of cartridge projects with legacy data definitions (that do not use the Data Dictionary). The legacy data definitions are migrated during import into a design time Data Dictionary that contains all of the data definitions within a single Data Dictionary file.

After importing an existing licensed cartridge, you will be prompted on where to store the Data Dictionary content for the cartridge. You can choose any one of the following locations:

- The Activation project associated with the existing licensed cartridge.
- A specified project (you may create a new project for the Data Dictionary or choose an existing project).

> **Note:** An existing Activation cartridge that is imported, modified, and exported using the latest version of Design Studio is importable in earlier versions of Design Studio. This allows modifications to legacy cartridges using the latest version of Design Studio.

You can use the following methods to import data into your Design Studio workspace:

- Import from a cartridge project. See "Importing Projects" for more information.
- Import from a SAR file. See "Importing Activation Cartridges from SAR Files" for more information.
- Import from an ASAP environment. See "Importing Cartridges from Environments" for more information.

You import Activation Network cartridges (from a location outside of your workspace) to set up your workspace. Cartridges can be imported in a zipped (zip or TAR files) or unzipped format.

Solution designers can import several network cartridges into the workspace to utilize their components (such as atomic actions, action processors, and NE templates) when setting up Activation Service cartridges. Solution designers can also import a suitable service cartridge as a starting point and modify it as needed.

Developers can import one or more network cartridges into their workspace (instead of building them from scratch) to use as starting point for creating new cartridges (for example, to make minor changes to cartridges for the next release) or to compare with other cartridges. See "Creating New Activation Cartridge Projects" for more information about creating new cartridges.

Note: To import cartridges into your workspace, you can zip up cartridge projects in ZIP or TAR format and export them to an archive file. This method produces a snapshot of the entire workspace as originally created (including one or multiple environments and SAR file) and includes any custom artifacts.

This method is useful when you have a project that contains a complex configuration with multiple dependencies between cartridges (for example, one or more Activation Service cartridges that depend on one or more Activation Network cartridges).

While it is possible to manually recreate the configuration by importing each of the SAR files from the original cartridges, some data loss may occur if you have custom artifacts that were not included in the original SAR file.

Note: To obtain Activation Network cartridges from Oracle, you can retrieve individual SAR files from the portal and import them into Design Studio. Each cartridge is self-contained (no dependencies exist).

Related Topics

Importing Activation Cartridge Projects

Working with Activation Cartridge Projects

Importing Activation Cartridges from SAR Files

A SAR file is one method for storing an ASAP configuration within a single artifact. Import cartridges from SAR files when you want to load into Design Studio cartridges that have been packaged in a SAR file. For example, when new Oracle Activation Network cartridges are available, they are packaged into a SAR file and posted to the portal in a zipped format (TAR file) for download. You can download the TAR file, then extract the SAR file from the TAR file.

Additionally, when you build a new Activation Service cartridge, Design Studio generates a SAR file. When you have finished creating the service model, you can email the SAR file to another Design Studio user for import or check it into a source control system.

Note: Importing activation cartridges from deployable archive (SAR) files using the Activation Archive Import Wizard creates an activation project that has only limited information compared to what was available when the cartridge was originally built. Oracle recommends that you distribute and deploy Design Studio Projects rather than SAR files.

To import a cartridge from a SAR file:

1. Select **File**, then select **Import**.

   The Import-Select dialog box appears.

2. Select **Studio Wizards**, then select **Activation Archive (SAR)** and click **Next**.

   The Activation Archive Import Wizard appears in both the cases.

3. In the Activation Archive Import Wizard, click **Browse** to search for a suitable SAR file that contains the network cartridge you need to create your service cartridge.

   After you select a SAR file, the fields in the wizard populate automatically for your cartridge.

4. Click **Next**.

   Cartridges that were created in Design Studio populate the **Cartridge Type** field automatically with the correct type (this is a non-editable field).

   Note: For cartridges not created in Design Studio, you must select the type of cartridge you are importing.

   ---

5. Click **Next**.

   The names of the two data schema entities (one entity for simple data elements and one entity for data structures) to which all the atomic action parameters of this cartridge are added appear.

6. (Optional) Select the location of the data schema entities.

   Note: The default location of the data schema entities for the cartridge is the cartridge project that is being imported.

   ---

7. Click **Finish** to start the import.

   A new Activation Service cartridge project appears in the Studio Projects view.

Note: Before working with an imported cartridge, see "Importing Activation Cartridge Projects" for more information about read-only statuses and working with sealed and unsealed cartridges.

Related Topics

Importing Activation Cartridge Projects

Working with Activation Cartridge Projects

Importing Cartridges from Environments

If you have an existing ASAP implementation, you can point Design Studio at an ASAP environment and import the configuration (service model, JAR files, and network elements) contained in that environment. This feature eliminates the need to manually run scripts that extract data from an environment into a SAR file and import that SAR file into Design Studio.

You can use this feature to obtain a snapshot of an environment configured from multiple sources, such as insert scripts, XML, or multiple Design Studio instances. In this scenario, it is possible to load the service model and network element configuration into Design Studio directly from an ASAP environment.

Additionally, importing from an existing ASAP environment is useful when you have just begun using Design Studio as a service modeling tool and you have an existing ASAP implementation.

To import a cartridge from an environment:

1. Select **Studio**, then select **Show Environment Perspective**.

   The Cartridge Management editor opens.

2. In the Environment view, select a Design Studio environment.
3. In the Cartridges area, click **Query**.

   The Test Environment Connection dialog box appears.

4. Enter the Cartridge Management web service user name and password and click **OK**.

   You are now connected to the environment on which ASAP is running. In the Cartridges area, the list of cartridges is refreshed.

5. Select a cartridge.

   In the Deployed Versions area, the deployed versions of this cartridge are available.

6. Select a deployed version and click **Import**.
7. Click **Finish**.

Related Topics

Importing Activation Cartridge Projects

Cartridge Management View

Activation Project Editor

Use the Activation Project editor to configure the cartridge specifications and parameters. In the Studio Projects view, double-click the Project entity to open the Activation Project editor.

Notes:

- Network elements and environments for ASAP are not defined in Activation cartridge projects.
- Activation cartridge projects are also Java projects (built on functionality of Java project). Java development can be done inside Activation cartridge projects; you do not need a separate Java project for development.
- Eclipse online documentation for Java projects (including configurations, properties, and settings) also applies to the Java configuration of an Activation cartridge project. See Eclipse online documentation when setting up a cartridge.

When working with the Activation Project editor, see the following topics:

- Activation Project Editor Blueprint Tab
- Activation Project Editor Properties Tab
- Project Editor Dependency Tab
- Activation Project Editor Cartridge Layout Tab
- Activation Project Editor Packaging Tab
- Activation Project Editor Locations Tab
- Activation Project Editor Testing Tab
- Project Editor Copyright Tab

Activation Project Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation of the project, including cartridge properties, service actions, atomic actions, action processors, connection handlers, and network element configuration (for network cartridges) or service configuration (for service cartridges). This tab is read only.

Related Topics

Activation Project Editor

Activation Project Editor Properties Tab

Use the **Properties** tab to configure cartridge properties and network element details. See "Project Editor Properties Tab" for more information about fields in this tab that are not specific to ASAP.

Note: In Activation, the **Default** field is not applicable.

Field	Use
Cartridge Type	Displays the cartridge type (**Activation Service Cartridge** or **Activation Network Cartridge**) selected on the New Studio Activation Cartridge Project wizard.
Vendor, Technology, and **Software Load**	Displays as specified in the New Studio Activation Cartridge Project wizard. In case of a new network element version you can change the vendor, technology, and software load. This allows you to reuse the cartridge.
Supported Hardware Models	Specify which network elements the cartridge supports.

Related Topics

Activation Project Editor

Working with Activation Cartridge Projects

Importing Activation Cartridge Projects

Activation Project Editor Cartridge Layout Tab

Use the **Cartridge Layout** tab to generate a framework model for a network or service cartridge.

For network cartridges, this tab enables you to generate a framework model, which creates a service action, atomic action and an action processor for any combination of action and entity and creates the appropriate association for the three elements in a 1:1:1 relationship. See "Generating Framework Models" for more information about framework models.

For service cartridges, this tab enables you to generate service actions. You can create a service model that is appropriate for a specific customer.

Field	Use
Add	Click to add action names and description in the cartridge project. For example, ADD, MOD, DEL, or QUERY.
Remove	Click to clear action names and description from the cartridge project.
Add	Click to add entities, which are related to actions, and their description in the cartridge project. For example, SUBSCRIBER, GSM-SUBSCRIBER, ROUTE, TRUNK, or LINE.
Remove	Click to clear entities from the cartridge project.
Generate Cartridge	Click to generate a framework model for a network or service cartridge. The framework model is based on a combination of actions and entities. See "Generating Framework Models" for more information about framework models.

Related Topics

Activation Project Editor

Working with Activation Cartridge Projects

Generating Framework Models

Activation Project Editor Packaging Tab

Use the **Packaging** tab to specify which entities to include in the cartridge SAR file. For service cartridges, usually all entities are deployed. In the Cartridge Packaging Instructions area select which entities you want to include for each of the categories.

Field	Use
Include all from Project	Select to include all entities from a specific resource. For example, if you want to include all of your Java libraries, select **Libraries** from the left-side column, then select the **Include all from Project** check box. The system will include all libraries in the package file.
Select	Click to add an entity in the package file.
Open	Click to open a specific entity.
Remove	Click to clear a specific entity from the package file.

Related Topics

Activation Project Editor

Packaging Activation Cartridges

Activation Project Editor Locations Tab

Use the **Locations** tab to display where items get stored. In Default Naming area, the **Archive File Name** field displays the same name as the project name.

Notes:

- If you change the ASAP version in the **Locations** tab to a version different than the version you selected during cartridge project creation, you must also update all dependent JAR files, such as **asaplibcommon.jar, JInterp.jar, Studio_2_6_0.jar,** and any other related JAR files. These files are required when the cartridge depends on third-party libraries. Additionally, you must delete any generated code. Lastly, you must perform a clean and full build to regenerate the code.
- Design Studio uses the default implementation package name as a prefix for generated code. Oracle recommends that you accept defaults and follow recommended naming conventions for all entities that you create.

Related Topics

Activation Project Editor

Activation Project Editor Testing Tab

Use the **Testing** tab to view ASAP test cases that you created for your projects and run them individually or simultaneously.

Field	Use
Environment	Select a Studio environment that has configuration for an ASAP environment.

Field	Use
Include all from Project	Select to include all test case entities.
Entity	Displays activation test case entities.
Status	Displays the status of a work order.
Run All	Click to execute all test cases one by one in the Studio environment.
Separate Console for each Test Case	Select to open a different console window for each test case in a Studio environment.
Select	Click to add a test case entity from the project.
Open	Click to open a specific test case entity.
Remove	Click to clear a specific test case entity from the **Testing** tab.
Run	Click to execute a specific test case.

Related Topics

Activation Project Editor

4

Modeling Activation Cartridge Project Data

The Data Schema editor provides Data Dictionary extension facility which allows you to associate data elements, in a data schema entity, to Oracle Communications ASAP run-time type parameters in the **Activation** tab of the Data Schema editor. When a data element is added in the **Parameters** tab of an atomic action entity, the properties of the data element configured in the Data Schema editor appear. The ASAP run-time type parameter values can be configured only at the root-level data elements like Scalar, Compound, XML, and XPath. Users of Oracle Communications Design Studio are allowed to override the values when required. See "Modeling Data" for more information.

When modeling Activation cartridge project data, see the following topics:

- About Activation Cartridge Project Data Modeling
- About the Atomic Action Editor Context Menu
- Data Schema Editor Activation Tab

About Activation Cartridge Project Data Modeling

Service action data models are the primary data sharing point with other Design Studio features. You can drag, to other Design Studio features, a structured data element corresponding to the entire service action or the data element corresponding to individual parameters. For example, drag a structured element from the Dictionary view to the data element tree of the order template of an Oracle Communications Order and Service Management (OSM) cartridge. This allows you to reuse the service action's data model, which is the target data model of an activation task in OSM.

The Service Action editor does not support data tree control. The data elements in a service action entity are indirectly associated with the data tree and are not directly editable.

The data elements in the data model of an atomic action entity can be dragged to the data model of another entity in which you can model data in the workspace. An atomic action entity can be the destination of a drag operation. This is more specifically done by dropping a data element onto the data element tree of the Atomic Action editor.

About the Atomic Action Editor Context Menu

The Atomic Action editor context menu contains actions specific to simple and structured data elements. To access these actions, you must right-click in the Parameters area in the **Parameters** tab. The context menu options that are available depend on the selection in the area. For example, the list of context menu options that appear when you have a data element actively selected is different from the list that appears when no data element in the area is selected.

Field	Use
Select Simple Element	Add a simple data element to the atomic action entity. See "Adding Existing Simple and Structured Data Elements to Entities" for more information.
Select Structured Element	Add a structured data element to the atomic action entity. See "Adding Existing Simple and Structured Data Elements to Entities" for more information.
Delete	Deletes data elements from the entity.
Move Up and **Move Down**	Repositions a data element in the Parameters area by moving it up or down in the list.
Refactoring	Select to access a menu of options for improving names and locations of the data elements. See "Refactoring Entities and Data Elements" for more information.
Expand	Expands a structured data element to display all child elements of the structure.
Collapse	Collapses a structured data element to hide all child elements of the structure.
Refresh	Refreshes the view.

Related Topics

Atomic Action Editor

Data Schema Editor Activation Tab

Use the **Activation** tab, in the Data Schema editor, to associate data elements in the Data Dictionary to ASAP run-time type parameters. See "Data Schema Editor" for more information on Data Schema editor.

Field	Use
Compound Member Label	Displays the same name as the data element. You can use the compound member in the code generated Java Action Processors (of the Java code) to access the members of the compound parameter. To rename the compound member, select **Rename** from the Data Schema editor context menu. See "About the Data Schema Editor Context Menu" for more information.

Field	Use
Runtime Type	Select any one of the following run-time type parameters in order to associate a data element to an ASAP parameter: **Note:** The specified type will be applied each time the data element is used on the **Parameters** tab of the Atomic Action editor. ■ **SCALAR:** Conventional name-value pair parameters for simple data elements. ■ **SCALARS:** Applicable to root-level of structured data elements. The leaf (child) elements become conventional name-value pair parameters. ■ **COMPOUND:** Contains structures or arrays of information that are represented by a particular structure name or compound parameter name. ■ **XML:** Used as values for both information parameters and extended work order properties. ■ **XPATH:** Defines an XPath expression into XML data. Depending on the run-time type parameter, the labels in this tab are visible. See *ASAP Developer's Guide* for more information on parameter types.
Atomic Action Label	Specify a unique name for the run-time type parameter in the atomic action. The default name is the name of the data element.
Service Action Label	Displays the name of the parameter in a service action. The name is unique and is the same as the data element name.
Indexed	Select the check box if you want to index the run-time type parameter.
Data Restrictions	Specify the description of the run-time type parameter that has any restriction on the value of the parameter.
Dependent XML	Displays the path of the XML that defines the parameter. This field is available only for the **XPATH** run-time type parameter.

Related Topics

Atomic Action Editor

5

Modeling ASAP Services

You can model ASAP services to develop Activation Network or Activation Service cartridges and provision services. The Cartridge Generation feature allows you to automatically generate a framework model.

When modeling ASAP services, see the following topics:

- About Modeling ASAP Services
- Understanding Service Models
- Modeling Services for Activation Network Cartridges
- Modeling Services for Activation Service Cartridges
- Working with Model Elements
- Generating Framework Models
- Creating Event Templates
- Documenting Models
- Example 1: Modeling Services
- Example 2: Modeling Services
- Action Processor Editor
- Atomic Action Editor
- Service Action Editor
- Event Template Editor

About Modeling ASAP Services

The steps you take to model services for an Activation cartridge depend on the type of cartridge (Activation Network or Activation Service cartridge) and, when modeling Activation Service cartridges, the type of model (mixed or common).

After you model a service, complete the parameter descriptions for entities by opening each editor and providing data (for example, open the Project editor to configure the cartridge and its elements). Also, complete any documentation information that needs to be included in the cartridge auto-generated documentation.

An interactive traceability allows you to trace from the data validation messages to the following:

- The affected cartridge design details that use the entity.

- The data schema entity definition in the Data Dictionary or (in the case of missing data element definitions) the name of the Data Dictionary where the data schema entity is defined. See"Modeling Activation Cartridge Project Data" for more information.

Related Topics

Modeling ASAP Services

Understanding Service Models

A service model is a combination of service actions and atomic actions that may or may not be specific to vendor, technology, and software load.

When working with service models, see the following topics:

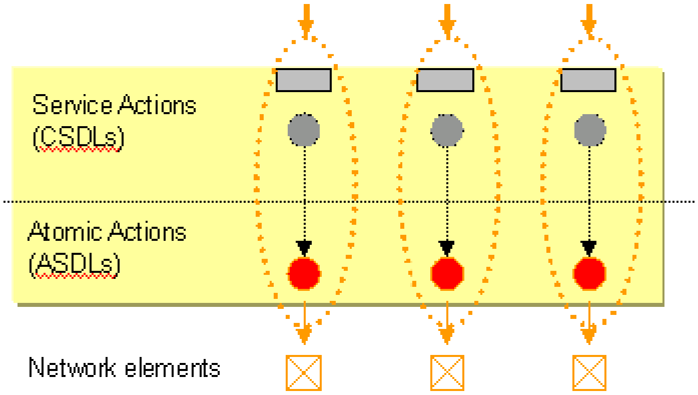
- About Vendor, Technology, and Software Load-Specific Service Models
- About Common Service Models
- About Mixed Service Models

About Vendor, Technology, and Software Load-Specific Service Models

Vendor, technology, and software load-specific service models are provided out-of-the-box with delivered cartridges. Entities within the service model containing the vendor, technology, and software load tokens and there is generally a one-to-one relationship (or limited one-to-many relationship) between service actions and atomic actions as shown in the following example and diagram:

```
C_NT-HLRPS_MSP8_ADD_CFB --> A_NT-HLRPS_MSP8_ADD_CFB
C_NT-HLRPS_MSP8_ADD_3WC --> A_NT-HLRPS_MSP8_ADD_3WC
C_NT-HLRPS_MSP8_ADD_SUB --> A_NT-HLRPS_MSP8_ADD_SUB

C_CSCO-CCM_4-1-X_GET_VOIP-INFO
     --> A_CSCO-CCM_4-1-X_GET_LINE
     --> A_CSCO-CCM_4-1-X_GET_PHONE
     --> A_CSCO-CCM_4-1-X_GET_USER
```



Service models designed in this way enable upstream systems to directly access device-specific operations. Using out-of-the-box service model design preserves simplicity in the Oracle Communications ASAP service model and requires less service modeling work within ASAP in the short term. However, it also forces upstream systems, which will be required to make selections of service actions based on the vendor, technology, and software loads being activated, to collate service actions together into meaningful work orders. Additionally, vendor equipment changes may create future maintenance.

> **Note:** When utilizing an out-of-the-box cartridge service model, consider consolidating service actions into meaningful building blocks to avoid pushing additional logic to upstream systems.

Consider the use of cartridge (vendor, technology, and software load-specific) service models when:

- Services are implemented very differently across vendors (for example, use of preconfigured profiles vs. passing of raw parameters to a network element) or next generation services whose standards are evolving (multiple vendors at different phases of support for new technologies, for example, and who have different interface specifications).
- One single type of vendor equipment is present in the network (for example, a specific vendor for HLRs, a specific vendor for voice mail) without future plans to introduce additional vendors into the network.
- Atomic actions are technology oriented (for example, nail up a relay point) rather than service oriented (for example, add a subscriber).
- Significant knowledge of the network infrastructure exists in upstream systems, such as Inventory.
- Highly complex domains (IP-VPN, ATM) with homogeneous networks (for example, Cisco) are used.
- Different activation steps (API calls) are required in order to activate the same services across different vendor equipment.

Lastly, if you have customer-specific parameter values that you want to expose to upstream systems, you can create new atomic actions with customer-specified atomic action parameters defined as defaults. This approach exposes only a subset of the cartridge atomic actions via the service actions. However, to use this variant you must create duplicate service actions and atomic actions with minor differences, which may create maintenance challenges.

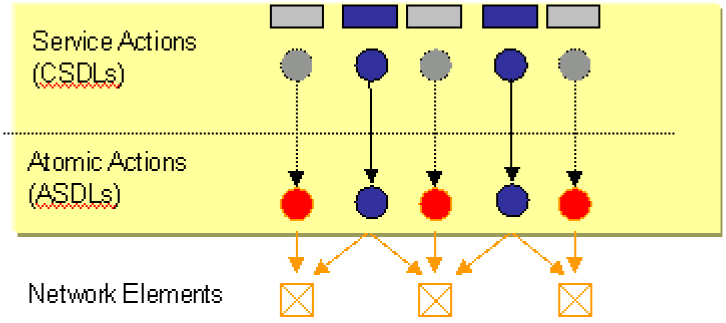
Related Topics

Modeling ASAP Services

About Common Service Models

Common service actions are most often associated with common atomic actions to create a consistently abstract service model (both service actions and atomic actions are common). In common service models one or more of the vendor, technology, and software load attributes are left out of the names of both service actions and atomic actions to indicate that they may be used to activate services on equipment from multiple vendors. ASAP has a built-in mechanism to map common atomic actions to a specific vendor, technology, and software load implementation based on the vendor, technology, and software load of the network element on which the service is being activated (see tbl_nep_asdl_prog for mapping details). The following example shows a common service that adds a subscriber regardless of whether it is a Nortel DMS 100 (POTS) or a Nortel CS2K (VoIP):

```
C_ADD_SUB --> A_ADD_SUB
```



Common service actions map to one or more common atomic actions (both shown in blue). The atomic actions map to multiple vendor, technology, and software load-specific implementations allowing for multiple technologies to be activated. The common atomic actions may simply be a renaming of the cartridge atomic actions to exclude one or more of the vendor, technology, software load attributes. In some cases, when a common service model is implemented in which many similar atomic actions across various network elements are required, the technology token must be maintained in the atomic actions to distinguish between the types technologies being activated.

Consider use of common service actions with common atomic actions when:

- There exists a subscriber and service oriented domain where services are implemented uniformly across vendor equipment. Similar activation steps are used to activate similar services uniformly across different vendor equipment.
- Changes may occur to the types of vendor equipment in the network or there is significant churn in the software loads. Limited change to the service model is desired despite changes to the network equipment.
- Inventory systems containing information about the equipment in the network are not available, or little information about the network is stored in upstream systems.

Common Service Model Example

The following wireless example demonstrates how a GSM subscriber is activated on the authentication center (AUC), flexible number routing network element (FNR), voice mail server (VMS) and home location register (HLR). This service model can be implemented to support technologies from many different vendors (for example, Nokia AUC, Ericsson AUC, and so on):

```
C_ADD_GSM-SUBSCRIBER -->
A_AUC_ADD_SUB
A_FNR_ADD_SUB
A_VMS_ADD_SUB
A_HLR_ADD_SUB
```

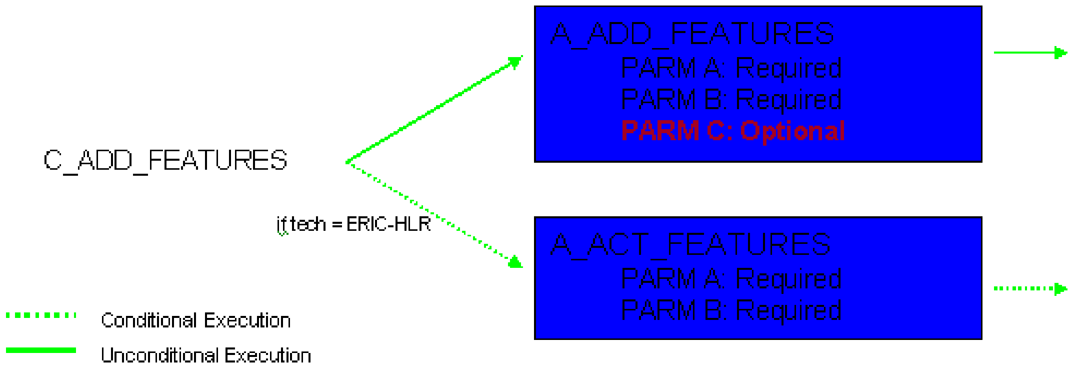
Common Service Model Challenges

Consider the following challenges when implementing a common service model:

- Different activation steps are required even when implementing simple subscriber oriented features.

- Significant parameter deltas can exist for similar services. For example, one service may require that you configure a subscriber by assigning a preconfigured profile to a subscriber on one network element (which is identified by its profile name); another may require that you configure a subscriber on a network element by passing all the details of the subscriber and services. In this case the atomic action parameters sets are dissimilar.

The following diagram shows an example of the some of the complications that can be involved in using a common service model even when the activating simple services. This example illustrates how to add a feature to a subscriber on an Ericsson HLR by first adding and then activating the feature (these are implemented as separate atomic actions); meanwhile, adding a feature on the Nortel HLR is a single atomic action. In order for a common service model to work in this case, you must configure spawning logic on the activate atomic action so that it does not execute when activating services on the Nortel HLR. The diagram also shows that `PARM C` is needed only when activating a service on the Nortel HLR. Therefore, it must be made optional so that a failure of the atomic action does not result when activating services on the Ericsson HLR (in which case `PARM C` will not be present on the work order).



Note: While it is ideal to use a common service model, multiple service activation differences on different vendor equipment often result in increased maintenance, complicated spawning logic, and numerous atomic actions per service action if using this method. In such cases, consider implementing logic upstream to select the correct vendor, technology, and software load-specific service actions.

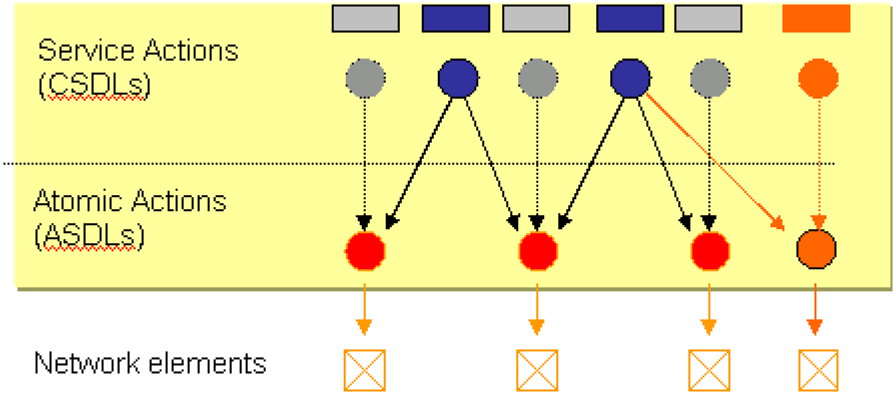
Related Topics

Modeling ASAP Services

About Mixed Service Models

Mixed service models combine common service actions and vendor, technology, and software load-specific atomic actions, and should be used with discretion when implementing solutions. Consider the following example:

```
C_HLR_ADD_SUB --> A_NT-HLRPS_MSP8_ADD_CFB
      --> A_NOK-HLR_9-0_ADD_CFB
```



Common service actions (shown in blue) map to multiple vendor, technology, and software load-specific atomic actions (shown in red). The common service actions are shown alongside vendor, technology, and software load-specific service actions from delivered cartridges (in gray) and other solution extensions (shown in orange) that will be discussed in subsequent sections.

Consider the use of common service actions with vendor, technology, and software load-specific atomic actions when the same services are implemented differently across different vendor equipment (resulting in many optional atomic action parameters using a common model) and service actions must be agnostic to avoid exposing network specific details to upstream systems. This model requires that spawning logic spawn the correct atomic action.

> **Note:** When implementing solutions, discard from the cartridges those service actions and atomic actions that are not used. Include in your production environment only those service modeling entities that you intend to use. Unused actions are exposed through the OCA client during fallout management, and may create confusion and unnecessary overhead when starting and stopping the ASAP system.

Related Topics

Modeling ASAP Services

Modeling Services for Activation Network Cartridges

To model services within an Activation Network cartridge:

1. Create service actions (with the vendor, technology, and software load tokens).
2. Create atomic actions (with the vendor, technology, and software load tokens).
3. Create action processors.
4. Create Java methods.
5. Associate elements (service actions, atomic actions, action processors, Java methods in a 1:1:1 relationship).

Note: While creating and associating these elements can be done manually, it is more common to use the Cartridge Generation feature for Activation Network cartridges. See "Generating Framework Models" for more information.

Related Topics

Modeling ASAP Services

Modeling Services for Activation Service Cartridges

You can model services for Activation Service cartridges that use either a common or mixed service model.

To model services for Activation Service cartridges:

1. Create service actions (common).
2. To model services that use a common service model:
   a. Create common atomic actions.
   b. Associate service actions, atomic actions and action processors from Activation Network cartridges in a 1:many:many relationship (action processors are already linked to Java methods).
3. To model services that use a mixed service model:

   Associate service actions to atomic actions from Activation Network cartridges in a 1:many:1 relationship (atomic actions from Activation Network cartridges are already linked to action processors and Java methods).

   Note: Using the Cartridge Generation feature, you can generate only service actions, you must select which atomic actions (one or possibly more) will be associated with the service actions in the cartridge.

   Note: You may need to create (for all types of Activation Service cartridges), customized action processors and Java methods to implement solution- specific behavior. See "Creating Custom Action Processors" for more information.

Related Topics

Modeling ASAP Services

Working with Model Elements

For modeling Activation cartridges, you need three model elements as follows:

- Atomic actions
- Service actions
- Action processors

When working with model elements, see the following topics:

- Understanding Model Element Relationships

- Creating Model Elements
- Creating Activation Run-Time Type Parameters in the Data Dictionary
- Configuring Element Properties

Understanding Model Element Relationships

After you have created the three types of elements required for modeling (service actions, action processors and atomic actions), you can define their relationship by linking them. For Activation Network cartridges, the three types of elements are generally in a 1:1:1 relationship. For Activation Service cartridges, the three types of elements are most commonly linked in a 1:1:many or 1:many:many relationship.

See "Modeling ASAP Services" for more information about service models.

Note: In most cases, you will not use the service models created for Activation Network cartridges in customer specific solutions. For Activation Network cartridges, these elements mainly provide a complete service model that can be used to test the cartridge action processors upon cartridge delivery.

Related Topics

Example 1: Modeling Services

Example 2: Modeling Services

Creating Model Elements

Creating Activation Run-Time Type Parameters in the Data Dictionary

Importing Activation Cartridge Projects

Modeling ASAP Services

About Activation Network Cartridge Relationships

For fast and convenient modeling of Activation Network cartridges, use the Cartridge Generation feature to generate the three element types for any combination of action and entity and to map them in the appropriate 1:1:1 relationship. See "Generating Framework Models" for information about the Cartridge Generation feature.

About Activation Service Cartridge Relationships

For Activation Service cartridges, you can use the Cartridge Generation feature only to generate service actions (the Cartridge Generation feature cannot determine which atomic actions (one or possibly more) must be spawned for each service action). To obtain the necessary atomic actions and action processors, you either create these with element wizards, or utilize appropriate ones from imported Activation Network cartridges. You then map each service action to several atomic actions, each of which need to be mapped to one or more action processors (1:1:many or 1:many:many relationship).

Note: An atomic action from an imported cartridge is already mapped to one action processor. If you reuse the atomic action from the cartridge, do not change this mapping and do not map additional action processors to the atomic action. An atomic action that you create is not yet linked to other elements (service actions and action processors) and therefore always needs to be mapped. To link elements (action processors to atomic actions, and atomic actions to service actions), drag an element from the Studio Projects view to the mapping tab in the editor of the element to which you want to link. For example, select an action processor from a cartridge in your Studio Projects view, then drag the action to the **Action Processor Mapping** tab in the editor of the appropriate atomic action. Similarly, you can drag atomic actions to the **Atomic Action** tab of a Service Action editor.

Creating Model Elements

You can use atomic action, service action, and action processor elements when modeling services for an Activation Cartridge project.

Note: Follow the steps in this procedure to create elements for Activation Service cartridges. After you create the elements, you must link the elements manually. Elements for Activation Network cartridges are usually created and linked with the Cartridge Generation feature.

To create a model element:

1. From the **Studio** menu, select **Show Design Perspective**.
2. From the **Studio** menu, select **New**, select **Activation**, then select the action element.

   The Studio Model Entity wizard appears.

   Note: The vendor, technology, software load fields of the wizards are non-editable for elements of Activation Network cartridges, but are writable for those of Activation Service cartridges. Activation Service cartridges may contain different types of service models, some of which do not identify a specific vendor, technology, or software-load attribute to indicate that they may be used to activate services on equipment from multiple vendors.

3. Select the applicable project.
4. Enter an action or select a previously defined action from the list (for example, ADD, MOD, DEL, or QUERY).
5. Enter a name for the entity (for example, SUBSCRIBER, GSM-SUBSCRIBER, ROUTE, TRUNK, or LINE).

   An updated name and a location name appears in non-editable fields based on the information in the **Vendor**, **Technology**, **Software Load**, **Action**, and **Entity** fields.
6. (Optional) Select a location for the entity.

By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

a. Deselect the **Use recommended name and location** check box.

b. Click the **Folder** field **Browse** button.

c. Navigate to the directory in which to save the entity.

d. Click **OK**.

7. Click **Finish**.

The new action entity appears in the Project folder in the Studio Projects view.

> **Note:** You must correct any problem markers on any entities before you deploy the cartridge. Refer to the Problems view for a short description of existing problems. For best results, set the Problem view filter to **On selected element and its children** to view problems in their full context.
>
> If problem markers seem invalid (for example, if they continue to reappear after you fix the problem in the configuration), you can usually remove these problems by selecting **Project, Clean** from the main menu. Select one or all listed projects and click **OK**. Oracle Communications Design Studio discards all build problems and build states of the selected projects and rebuilds the projects from scratch.

Related Topics

About Activation Network Cartridge Projects

About Activation Service Cartridge Projects

Generating Framework Models

Modeling ASAP Services

Creating Activation Run-Time Type Parameters in the Data Dictionary

After defining the element relationships, you need to create activation run-time type parameters that can be used in the linked elements in their respective editors (you can also do this before linking the elements).

> **Note:** If you want to use the automatically generated code (see "Defining Action Processor Properties") and if you want the code to include content that support parameters, you must first create the parameters and associate them to atomic actions (see "Defining Atomic Action Properties") before generating the code.

For more information about activation parameters, see *ASAP Cartridge Development Guide*.

Design Studio supports the following:

- Creating a Scalar Parameter
- Creating a Compound Parameter
- Creating an XML Parameter

- Creating an XPATH Parameter
- Grouping Scalar Parameters using Structured Elements

Creating a Scalar Parameter

Scalar parameters are conventional name-value pair parameters.

```
Service Action      C-ADD_FEATURE
     PARM  NE_ID   NEWYORK
     PARM  LEN     2111112
     PARM  LATA    516
     PARM  LCC     555
```

To create a mandatory, optional, or indexed scalar parameter:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the tab for the Dictionary view.
3. Right click in the Dictionary view and select **Add Simple Schema Element**.

   The Create Data Schema Element wizard appears.
4. Enter the following:
   a. In the **Entity** field, enter the name of the project to which you want to add a scalar parameter.

   b. In the **Name** field, enter an element name.

   c. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see Oracle Communications Design Studio Help.

   d. In the Multiplicity field, select one of the following:
      - **Required**: This attribute makes the parameter mandatory.
      - **Optional**: This attribute makes the parameter optional.
      - **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.
5. Click **Finish**.

   The new parameter appears in the Dictionary view. You may need to expand the schema for your cartridge to see it.
6. Double-click the new parameter to open the Data Schema editor with that parameter selected.
7. Click the **Activation** subtab.
8. From the **Runtime type** list, select **SCALAR**.
9. (Optional) Select **Indexed** to index the parameter.

Related Topics

Creating Model Elements

Configuring Element Properties

Modeling ASAP Services

Atomic Action Editor

Creating a Compound Parameter

To create a compound parameter:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the tab for the Dictionary view.
3. Right click in the Dictionary view and select **Add Structured Schema Element**.

   The Create Data Schema Structure wizard appears.
4. Enter the following:
   a. In the **Entity** field, enter the name of the project to which you want to add a scalar parameter.
   b. In the **Name** field, enter an element name.
   c. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see the Oracle Communications Design Studio Help.
   d. In the **Multiplicity** field, select one of the following:
      – **Required**: This attribute makes the parameter mandatory.
      – **Optional**: This attribute makes the parameter optional.
      – **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.
5. Click **Finish**.

   The new parameter appears in the Dictionary view. You may need to expand the schema for your cartridge to see it.
6. Double-click the new parameter to open the Data Schema editor with that parameter selected.
7. Click the **Activation** subtab.
8. From the **Runtime type** list, select **COMPOUND**.

   > **Note:** All child elements inherit the **Activation** tab attributes from the base compound element.

9. (Optional) Select **Indexed** to index the parameter.
10. Right click the new parameter in the Dictionary view and select **Add Simple Child Schema Element**.

    The Create Data Schema Element wizard appears.

    > **Note:** Compound parameters do not support structured child schema elements.

11. Enter the following:

a. In the **Name** field, enter an element name.

b. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see the Oracle Communications Design Studio Help.

c. In the **Multiplicity** field, select one of the following:

   – **Required**: This attribute makes the parameter mandatory.

   – **Optional**: This attribute makes the parameter optional.

   – **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.

12. Click **Finish**.

13. Repeat steps 10 to 12 for any additional parameters to be included in the compound parameter.

Related Topics

Creating Model Elements

Configuring Element Properties

Modeling ASAP Services

Atomic Action Editor

Creating an XML Parameter

To create an XML parameter:

1. From the **Studio** menu, select **Show Design Perspective**.

2. Click the tab for the Dictionary view.

3. Right click in the Dictionary view and select **Add Simple Schema Element**.

   The Create Data Schema Element wizard appears.

4. Enter the following:

   a. In the **Entity** field, enter the name of the project to which you want to add a scalar parameter.

   b. In the **Name** field, enter an element name.

   c. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see the Oracle Communications Design Studio Help.

   d. In the **Multiplicity** field, select one of the following:

      – **Required**: This attribute makes the parameter mandatory.

      – **Optional**: This attribute makes the parameter optional.

      – **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.

5. Click **Finish**.

   The new parameter appears in the Dictionary view. You may need to expand the schema for your cartridge to see it.

6. Double-click the new parameter to open the Data Schema editor with that parameter selected.
7. Click the **Activation** subtab.
8. From the **Runtime type** list, select **XML**.

Related Topics

Creating Model Elements

Configuring Element Properties

Modeling ASAP Services

Atomic Action Editor

Creating an XPATH Parameter

To create an XPATH parameter:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the tab for the Dictionary view.
3. Right click in the Dictionary view and select **Add Simple Schema Element**.

   The Create Data Schema Element wizard appears.

4. Enter the following:

   a. In the **Entity** field, enter the name of the project to which you want to add a scalar parameter.

   b. In the **Name** field, enter an element name.

   c. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see the Oracle Communications Design Studio Help.

   d. In the **Multiplicity** field, select one of the following:

      – **Required**: This attribute makes the parameter mandatory.

      – **Optional**: This attribute makes the parameter optional.

      – **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.

5. Click **Finish**.

   The new parameter appears in the Dictionary view. You may need to expand the schema for your cartridge to see it.

6. Double-click the new parameter to open the Data Schema editor with that parameter selected.
7. Click the **Activation** subtab.
8. From the **Runtime type** list, select **XPATH**.

9. (Optional) Select **Indexed** to index the parameter.
10. In the Dependent XML field create or select a dependent XML. This attribute displays the path of the XML file that defines the parameter. This field is available only for the XPATH run-time type parameter.

Related Topics

Creating Model Elements

Configuring Element Properties

Modeling ASAP Services

Atomic Action Editor

Grouping Scalar Parameters using Structured Elements

You can group ASAP scalar parameters using the structured schema element feature. The structure element is a container that holds ASAP parameters. For example the following scalar groups can be defined using two levels of structure elements:

```
Structure element1
     Structure element2
              Scalar1
              Scalar2
Structure element3
     Structure element4
              Scalar3
              Scalar4
```

In a real world scenario, these structure could be as follows:

```
Person
     Name
         First_name
         Last_name
Place
     Address
         Number
         Street
```

The structure elements used in Design Studio are converted into individual ASAP scalar parameters by absorbing the structured element names into the scalar parameter name. The example used above describing a person and place would by default look as follows as ASAP parameters:

```
Person_Name_First_name
Person_Name_Last_name
Place_Address_Number
Place_Address_Street
```

The default character used to separate the elements in the ASAP parameter names is the underscore (_). It is possible to change this character. See "Defining Design Studio Activation Preferences" for more information.

To group scalar parameters:

1. From the **Studio** menu, select **Show Design Perspective**.
2. Click the tab for the Dictionary view.
3. Right click in the Dictionary view and select **Add Structured Schema Element**.

The Create Data Schema Structure wizard appears.

4. Enter the following:

   a. In the **Entity** field, enter the name of the project to which you want to add a scalar parameter.

   b. In the **Name** field, enter an element name.

   c. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see the Oracle Communications Design Studio Help.

   d. In the **Multiplicity** field, select one of the following:

      – **Required**: This attribute makes the parameter mandatory.

      – **Optional**: This attribute makes the parameter optional.

      – **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.

5. Click **Finish**.

   The new parameter appears in the Dictionary view. You may need to expand the schema for your cartridge to see it.

6. Double-click the new parameter to open the Data Schema editor with that parameter selected.

7. Click the **Activation** subtab.

8. From the **Runtime type** list, select **SCALARS**.

9. Right click the new parameter in the Dictionary view and select one of the following:

   - **Add Simple Child Schema Element**: Select this attribute if you want to immediately define xml or scalar parameters within the first structured element. If you select this option, go to step 10.

   - **Add Structured Child Schema Element**: Select this attribute if you want additional structured child schema elements below the first structured element. If you select this option, repeat steps 4 to 9.

10. Enter the following:

    a. In the **Name** field, enter an element name.

    b. In the **Display Name** field, enter a display name. The Data Schema editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. For more information, see the Oracle Communications Design Studio Help.

    c. In the Multiplicity field, select one of the following:

       – **Required**: This attribute makes the parameter mandatory.

       – **Optional**: This attribute makes the parameter optional.

       – **Range**: Any ranged parameter with a **Minimum** value greater than 0 is considered a mandatory ASAP parameter. Any ranged parameter with a **Minimum** value of 0 is considered an optional ASAP parameter.

11. Click **Finish**.
12. Repeat steps 10 to 11 for any additional parameters to be included in the scalar or XML parameter group.

Related Topics

Creating Model Elements

Configuring Element Properties

Modeling ASAP Services

Atomic Action Editor

Configuring Element Properties

After defining the element relationships, you need to define the properties and parameters for the linked elements in their respective editors (you can also do this before linking the elements).

When configuring element properties you can define:

- Defining Service Action Properties
- Defining Atomic Action Properties
- Defining Action Processor Properties

In the Studio Projects view, double-click the entity and access the editors.

> **Note:** You can maximize the editor view to see all editor content by double-clicking the view's title bar.

> **Note:** If you create compound parameters, it is recommended to define the members for every compound parameter. This is beneficial once the code is generated during implementation. Ensure your parameters are valid Java parameters. See "Understanding Java with Code Generation" for more information.

Related Topics

Modeling ASAP Services

Defining Service Action Properties

You can configure service action properties using the Service Action editor.

To define properties and parameters for service actions:

1. In the Studio Projects view, double-click a service action entity.

   The Service Action editor appears.
2. In the **Description** field, enter a description for the editor.
3. Click the **Atomic Action(s)** tab.

   The **Atomic Action(s)** tab allows you to map an atomic action to a service action. Depending on the cartridge model, you can map more than one atomic action to one service action.

4. Click **Add**.

   The Atomic Action Selection dialog box appears.

5. Select an atomic action to which you want to map the service action.

   In the Service Action Details area, a new row with default values appears.

6. Click the row and edit the values in the Atomic Action Conditions area.
7. Click the **Properties** tab and define the properties for the service action.
8. In the **Level** field, enter the sequence level for the service action in the work order.
9. Select or enter values for **Service Action Completion Event** and **Service Action Failure Event**.

   When the service action completes or fails, the respective event is returned.

10. (Optional) To configure rollback for a service action, do the following:

    a. Select the **Rollback** check box.

    b. In the **Atomic Actions** tab, click in the Rollback Point column for the atomic action that you want to set as Point of No Return (PNR).

    c. From the list, select a rollback value.

    > **Notes:**
    >
    > - A PNR can be specified only if the service action has rollback enabled with the **Rollback** check box selected in the **Properties** tab.
    > - If rollback is enabled but not configured in the **Atomic Action(s)** tab, the default behavior is a complete rollback of all atomic actions in a failed work order.

11. Click the **Parameters** tab to see the mapping of a service action and an atomic action. The details of the atomic action are based on the parameters defined on the Atomic Action editor.
12. Click the **Command Overview** tab to see the details of the atomic action and service actions.
13. Select **File**, then select **Save**.

Related Topics

Creating Model Elements

Creating Activation Run-Time Type Parameters in the Data Dictionary

Configuring Element Properties

Modeling ASAP Services

Service Action Editor

Defining Atomic Action Properties

You can configure atomic action properties using the Atomic Action editor.

To define properties and parameters for atomic actions:

1. In the Studio Projects view, double-click an atomic action entity.

The Atomic Action editor appears.

2. In the **Description** field, enter a description for the editor.
3. In the **Details** tab, select a routing support.
4. In the **Parameters** tab, the mandatory parameters for the routing support are automatically added, depending on the routing support you selected, to the Parameters area. In the **Activation** tab, the **Service Action Label** and **Atomic Action Label** are automatically defined for these routing parameters.

   For example, in the **Details** tab if you select **Dynamic Routing**, in the Parameter Details area, you can enter the data restrictions and other details for the parameter.
5. Right-click in the parameters area and select one of the following:
   - If you want to add a scalar, XML, or XPATH run-time parameter, select **Select Simple Data Element** and add a parameter from the activation data dictionary.
   - If you want to add a grouping of scalar parameters or a compound parameter, select **Select Structured Data Element** and add a parameter from the activation data dictionary.

   See "Creating Activation Run-Time Type Parameters in the Data Dictionary" for more information about activation run-time parameters.
6. In the **Mappings** tab, you can map an atomic action to an action processor.
7. (Optional) Click the **Add** button and map an atomic action to any of the existing action processors.

   The Action Processor Selection Dialog dialog box appears. You can add an action processor from the list of existing action processors in workspace.
8. (Optional) On the Action Processor Selection Dialog dialog box, click the **New** button to create a new action processor.

   The Action Processor Wizard appears. You can fill the appropriate fields of this wizard to create a new action processor entity.
9. (Optional) You can do the following:
   - Click the **Clone** button to create a copy of the selected action processor.
   - Click the **Open** button to open the selected action processor.
   - Click the **Remove** button to remove the selected action processor.
10. In the **Details** tab, in the Detailed Information area, provide one or more atomic action properties.
11. (Optional) To configure an atomic action for rollback, do the following:
    a. Click **Select**.

       The Rollback Atomic Service Selection dialog box appears.

    b. Select an atomic action for rollback and click **OK**.
12. Select **File**, then select **Save**.

Related Topics

Creating Model Elements

Creating Activation Run-Time Type Parameters in the Data Dictionary

Configuring Element Properties

Modeling ASAP Services

Atomic Action Editor

Defining Action Processor Properties

You can configure action processor properties using the Action Processor editor.

To define properties and parameters for action processors:

1. In the Studio Projects view, double-click an action processor entity.

   The Action Processor editor appears.
2. In the **Description** field, enter a description for the editor.
3. From the list, select the type.

   Java Action Processor (with Code Generation) is the default value.
4. Depending on the type you selected, define the remaining parameters.

   For example, if you select the Java Action Processor, define the class and method.
5. Select **File**, then select **Save**.

Related Topics

Creating Model Elements

Creating Activation Run-Time Type Parameters in the Data Dictionary

Configuring Element Properties

Modeling ASAP Services

Action Processor Editor

Generating Framework Models

To simplify and expedite the creation of cartridges, a Cartridge Generation feature is available in Design Studio. The cartridge generation feature works differently depending on the type of cartridge you are creating. For Activation Network cartridges, service actions, atomic actions, and action processors are created and linked in a 1:1:1 relationship for all combinations of the actions and entities you specify. For an Activation Service cartridges, only service actions are created, as the feature cannot determine which atomic actions (one or possibly more) you want to associate with the service actions in the cartridge you are creating. For Activation Service cartridges, a decision must also be made about the type of service model you will create for the customer (common, mixed or vendor/technology/software load-specific) which will affect the naming convention used for the atomic actions.

> **Note:** For Activation Service cartridges, you must create and associate atomic actions with the service actions that you created with the Cartridge Generation feature. For the atomic actions that you create (regardless of the type of service model used), you can reuse the action processors from cartridges. When using common atomic actions (that you have created) you must manually link atomic actions with action processors. If you are reusing atomic actions from cartridges, the links between the atomic actions and action processors will already exist.

To use the cartridge generation feature for generating a framework model of Activation Network cartridges, you specify the actions that will be performed by the cartridge (ADD, MOD, DEL, QUERY), and the entities on which these will be performed (PORT, SUBSCRIBER, SUBSCRIPTION, LINE, and so on). Also, you enter descriptions for the actions and entities, which are combined for each action and entity combination (for example, add a single port on the device). After you have entered this information into the Project editor **Cartridge Layout** tab, you generate a framework model by running the Cartridge Generation tool.

> **Note:** The Cartridge Generation tool does not overwrite a framework that already exists. Rather, it adds to framework new and modified actions and entities. Additionally, Design Studio does not delete old actions or entities. You can however, delete them manually.

When generating framework models, see the following topics:

- Generating Framework Models for Activation Network Cartridges
- Generating Framework Models for Activation Services Cartridges

Related Topics

About Common Service Models

About Vendor, Technology, and Software Load-Specific Service Models

About Mixed Service Models

Generating Framework Models for Activation Network Cartridges

You can use the Cartridge Generation tool to generate models for Activation Network cartridges.

To generate framework models for Activation Network cartridges:

1. In the Studio Projects view, double-click an Activation Project entity to open the Project editor.
2. Click the **Cartridge Layout** tab.
3. In the Add area, click **Add**.

   Design Studio prompts you to enter an action and description.
4. In the Entity area, click **Add**.

   Design Studio prompts you to enter an entity and description.
5. Repeat steps 1 and 2 and add any additional action and entity combinations.

6. Click **Generate Cartridge**.

   A dialog box appears and prompts you to confirm the generation.

7. Click **Yes**.

   Design Studio creates service actions, atomic actions, and action processors for all action and entity combinations. The generated entities appear in the Studio Projects view. You can view the hierarchy and relationships in the Relation Graph view.

8. To complete the modeling, adjust the properties and define parameters for the atomic action.

Related Topics

Generating Framework Models

Modeling ASAP Services

Generating Framework Models for Activation Services Cartridges

You can use the Cartridge Generation tool to generate models for Activation Service cartridges.

To generate framework models for Activation Service cartridges:

1. In the Studio Projects view, double-click the desired Activation Service entity, and open the Project editor.

2. Click the **Cartridge Layout** tab.

3. In the Add area, click **Add**.

   Design Studio prompts you to enter an action and description.

4. In the Entity area, click **Add**.

   Design Studio prompts you to enter an entity and description.

5. Repeat steps 1 and 2 and add any additional action and entity combinations.

6. Click **Generate Cartridge**.

   A dialog box appears and prompts you to confirm the generation.

7. Click **Yes**.

   Design Studio creates service actions for all action and entity combinations. The generated entities appear in the Studio Projects view. You can view the hierarchy and relationships in the Relation Graph view.

8. Create or locate the appropriate atomic actions and link them to the generated service actions.

   If you use common atomic actions, link them to the action processors in the Activation Network cartridges.

Related Topics

Generating Framework Models

Modeling ASAP Services

Creating Event Templates

You can use the Event Template Wizard to create an event template entity.

To create an event template:

1. From the **Studio** menu, select **Show Design Perspective**.
2. From the **Studio** menu, select **New**, select **Activation**, then select **Event Template**.
3. Select a project for this element and enter a name for the entity.
4. (Optional) Select a location for the entity.

   By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

   a. Deselect the **Use recommended name and location** check box.

   b. Click the **Folder** field **Browse** button.

   c. Navigate to the directory in which to save the entity.

   d. Click **OK**.
5. Click **Finish**.

   In the Studio Projects view, in the project folder, the new event template entity appears.

See *ASAP System Administrator's Guide* and *ASAP Developer's Guide* for more information on event templates.

Related Topics

Event Template Editor

Documenting Models

Design Studio automatically generates documentation for Activation cartridge modeling in HTML format and updates the documentation with each build. To access the documentation within each of the editors, navigate to the **Blueprint** tab. The blueprint displays documentation for the entity and provides links to the other entities.

There are some entities for which Design Studio does not generate documentation. You must include text for the following:

- For for each of the three modeling elements (service action, atomic action, and action processor), you must define the **Description** field.
- For the action processor, you must include documentation for commands used (in the Command Overview), the output of the action processor (in the **Output** tab), and comments (in the **Development Notes** tab).
- For atomic actions, you must define the **Data Restriction** field, which enables you to specify valid range and values for parameters within the atomic action.
- For service actions, you must define any properties.

Related Topics

Modeling ASAP Services

Example 1: Modeling Services

In this example, you model a service to create a postpaid mobile subscriber with the following constraints:

- Only Ericsson Home Location Registers (HLRs) are used, but there are multiple software loads present in the network for this network element type (release 11-0 and release 12-0).
- Each time a subscriber is added, the subscriber must also be added to the Flexible Number Register (FNR) for number portability purposes.
- Only Ericsson FNRs are used, but there are multiple software loads present in the network for this network element type (release 8-0 and release 9-1).

Given these constraint, you determine that an activation network service model is inappropriate and decide to implement a common service model, as there are multiple software loads and technologies involved.

> **Note:** When different vendors exist in the network, it may not be possible to use a common software load. See "About Common Service Models" for information about advantages and disadvantages of using a common service model. Additionally, see "Example 2: Modeling Services".

To model a service to create a postpaid mobile subscriber:

1. Import the required cartridges:
   - Ericsson HLR R11-0
   - Ericsson HLR R12-0
   - Ericsson FNR R8-0
   - Ericsson FNR R9-1
2. Create a common service action that is vendor, technology, software load agnostic.

   This service action will be required to activate services on different vendor equipment (Ericsson HLR and Ericsson FNR) running different software loads (multiple software loads for each technology):

   ```
   C_CREATE_POSTPAID-SUBS
   ```
3. Create common atomic actions for the Ericsson HLR and Ericsson FNR to create the subscriber.

   These should be software load agnostic to take advantage of the fact that there is only one vendor with the need to support multiple software loads:

   - `A_ERIC-HLR_CREATE_SUBSCRIBER`
   - `A_ERIC-FNR_CREATE_SUBSCRIBER`
4. Drag the action processors (from the imported cartridges) to the common atomic actions.

   By reusing cartridge action processors, the parameters for the atomic action will be automatically generated:

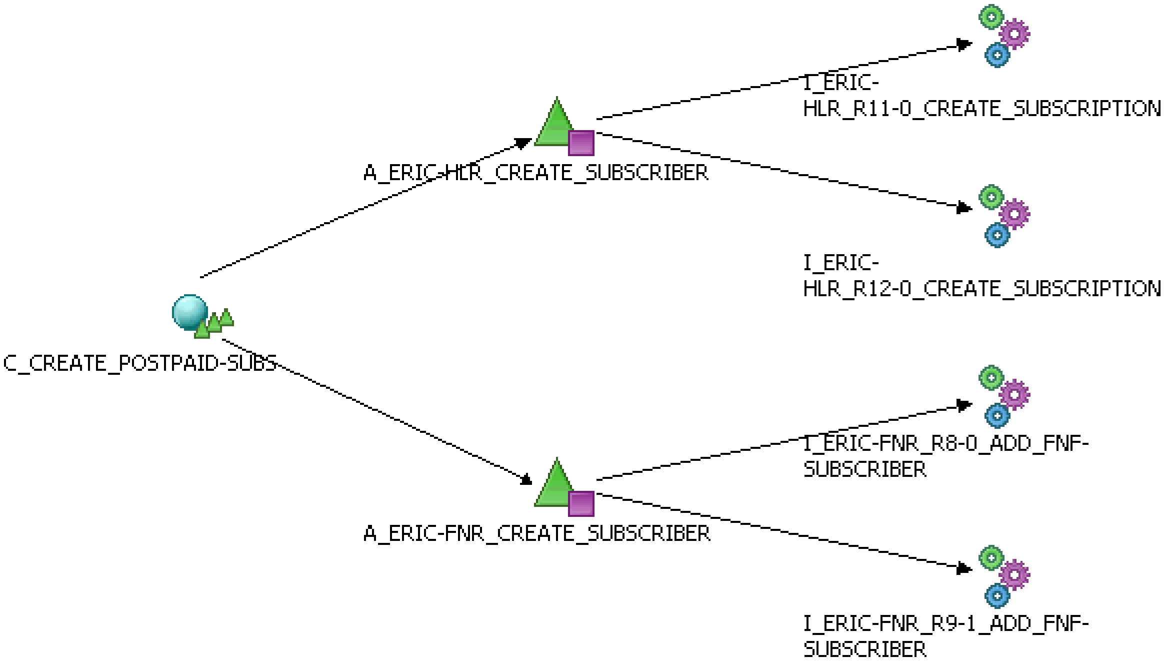
   ```
   A_ERIC-HLR_CREATE_SUSBSCRIBER -> I_ERIC-HLR_R11-0_CREATE_SUBSCRIPTION
   A_ERIC-HLR_CREATE_SUSBSCRIBER -> I_ERIC-HLR_R12-0_CREATE_SUBSCRIPTION
   A_ERIC-FNR_CREATE_SUBSCRIBER -> I_ERIC-FNR_R8-0_ADD_FNF-SUBSCRIBER
   ```

```
A_ERIC-FNR_CREATE_SUBSCRIBER -> I_ERIC-FNR_R9-1_ADD_FNF-SUBSCRIBER
```

5. Drag each of the common atomic actions you have created onto the common service action you have created to link all elements together:

```
C_CREATE_POSTPAID-SUBS -> A_ERIC-HLR_CREATE_SUSBSCRIBER
C_CREATE_POSTPAID-SUBS -> A_ERIC-FNR_CREATE_SUSBSCRIBER
```

The linked elements should now appear in the Relation Graph view:


I_ERIC-HLR_R11-0_CREATE_SUBSCRIPTION
A_ERIC-HLR_CREATE_SUBSCRIBER
I_ERIC-HLR_R12-0_CREATE_SUBSCRIPTION
C_CREATE_POSTPAID-SUBS
I_ERIC-FNR_R8-0_ADD_FNF-SUBSCRIBER
A_ERIC-FNR_CREATE_SUBSCRIBER
I_ERIC-FNR_R9-1_ADD_FNF-SUBSCRIBER

Note: If the customer has additional software loads to support in the future, these can be added by dragging the action processors to the atomic actions (the parameter set is rationalized automatically but should be verified manually to ensure accuracy of parameter attributes).

Related Topics

Understanding Model Element Relationships

Relation Graph General View

Modeling ASAP Services

Example 2: Modeling Services

In this example, you model a service to create a PSTN subscriber with the following constraints:

- There are two vendors, Lucent 5ESS and Nortel DMS 100.
- There is currently only one software load in the network for each (16 and SN06, respectively).

Given these constraints, you determine that it may be too challenging to use an exclusively common service model if the atomic action from each cartridge is different (for example, if the action contains different parameters sets). One possible way to model this is to use a common service action but reuse the atomic actions from the cartridges.

To model a service to create a PSTN subscriber:

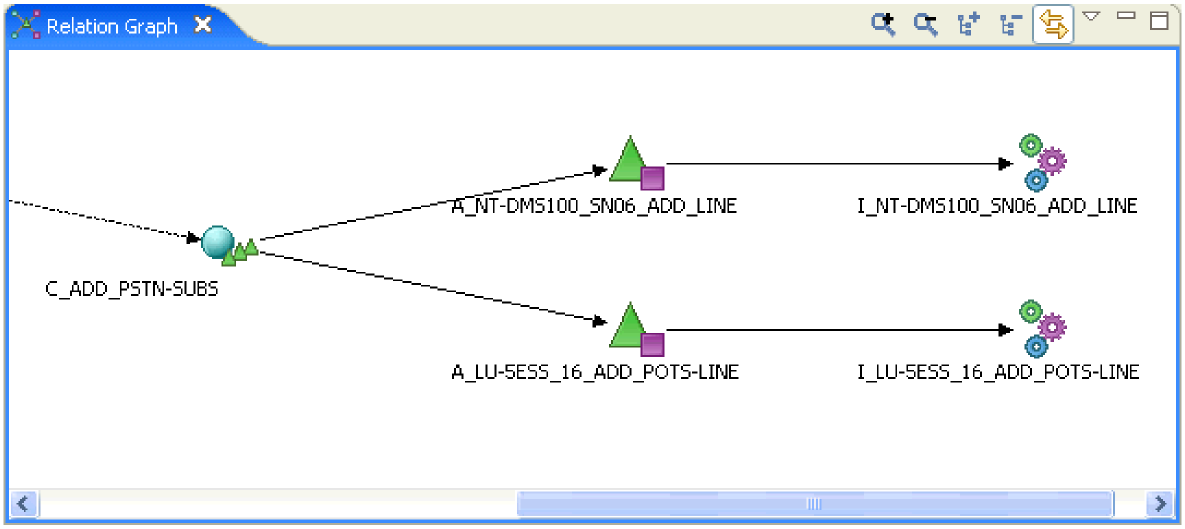
1. Import the required cartridges:
   - Lucent 5ESS 16
   - Nortel DMS 100 SN06
2. Create a common service action (vendor, technology, software load agnostic).

   This service action will be required to activate services on different vendor equipment (Lucent and Nortel):

   ```
   C_ADD_PSTN-SUBS
   ```
3. Drag the atomic actions from the cartridges to the service action you have just created:

   ```
   C_ADD_PSTN_SUBS -> A_NT-DMS100_SN06_ADD_LINE
   C_ADD_PSTN_SUBS -> A_LU-5ESS_16_ADD_POTS-LINE
   ```

   The linked elements now appear in the Relation Graph view:


Relation Graph
A_NT-DMS100_SN06_ADD_LINE
I_NT-DMS100_SN06_ADD_LINE
C_ADD_PSTN-SUBS
A_LU-5ESS_16_ADD_POTS-LINE
I_LU-5ESS_16_ADD_POTS-LINE

4. Create the appropriate spawning logic using the vendor, technology, and software load to ensure the correct atomic action runs for each work order.

Note: Alternatively, you could implement this service model using logic in the SRT component to execute the correct vendor specific service model, as this configuration has performance inefficiencies due to the need for conditional spawning logic. Moreover, the configuration has complexities that are introduced into the service model when multiple software loads require support (for example, a dedicated atomic action is required for each).

Related Topics

Understanding Model Element Relationships

Relation Graph General View

Modeling ASAP Services

Action Processor Editor

Use the Action Processor editor to define an action processor entity. In the Studio Projects view, double-click an action processor entity to open the Action Processor editor. You can create the editor using the Action Processor Wizard.

When working with the Action Processor editor, see the following topics:

- Action Processor Editor Editor Tab
- Action Processor Editor Blueprint Tab

Action Processor Editor Editor Tab

Use the **Editor** tab for defining the parameters and generating the code for an action processor. In the Action Processor Properties area, you can define the values for an action processor entity.

Field	Use
Description	Specify a description for the Action Processor editor.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load for the action processor entity in the cartridge project.
Action	Specify or select an action for the action processor. **Note:** If an action is not in use by any action processor in any cartridge in the Design Studio workspace, enter the action in the **Action** list.
Type	Select any one of the following types from the list: ■ Java Action Processor (with Code Generation) This type is the default value and recommended. Based on the selected action, this type generates a basic class and the skeleton of a method required for an action processor. ■ Java Action Processor This type requires that you (or developers) map an existing class and method to the action processor. Unlike for Java Action Processor (with Code Generation), you must manually write the code for the class and method for the action processor. ■ State Table This requires that you (or developers) map an existing state table and program to the action processor. Similar to the Java Action Processor type, you must manually write code for the state table and program.

Field	Use
Class and **Method**	For Java Action Processor (with Code Generation), the fields are blank and grayed out until a code is generated using **New** button. When the code is generated the class and method names mapped to the action processor are populated in these fields. For Java Action Processor, the fields display the default values for the class and method as **myProcessor** and **execute** respectively. The code for the default values does not exist. You must modify these values with existing class and method names and map them using **Select**.
State Table and **Program**	Displays the default values for the state table and program as **S_MY_STATE_TABLE** and **doAction** respectively. **Note:** The code for the default values does not exist. You must modify the default values with existing state table and program names and map them using **Select**.
Open	Click to see the code. This button is enabled only when the code is available.
New	Click to create the Java action processor implementation. **Note: New** is grayed out for Java Action Processor and State Table.
Select	Click to select Java implementation or state table implementation. Use this button to map the class and method for the Java Action Processor type and state table and program for the State Table type. **Note: Select** is grayed out for Java Action Processor (with Code Generation).
Command Overview, Output, and **Development Notes**	Specify documentation in the Command Overview area for the MML commands, output from the action processor (for example, return the following parameters as INFO or CSDL) in the Output area, and comments provided by the cartridge developer in the Development Notes area respectively.

Related Topics

Action Processor Editor

Defining Action Processor Properties

Modeling Entities

Action Processor Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the action processor entity. This tab is read only.

Related Topics

Action Processor Editor

Modeling Entities

Atomic Action Editor

Use the Atomic Action editor to define activation run-time type parameters, define a routing type, and map the atomic action to one or more action processors.

For atomic actions generated by Design Studio from CTM Templates for IP Service Activator, see IP Service Activator Atomic Action Editor.

You can create the Atomic Action editor using the Atomic Action Wizard. In the Studio Projects view, double-click an atomic action entity to open the editor.

When working with the Atomic Action editor, see the following topics:

- Atomic Action Editor Parameters Tab
- Atomic Action Editor Details Tab
- Atomic Action Editor Mappings Tab
- Atomic Action Editor Blueprint Tab

Atomic Action Editor Parameters Tab

Use the **Parameters** tab to add data elements from the Data Dictionary and configure values to the ASAP run-time type parameters.

> **Note:** In Activation, the **Enumeration**, **Tags**, and **Usage** tabs are not applicable.

When working with the **Parameters** tab, see the following topics:

- Atomic Action Editor Parameters Area
- Details Tab
- Activation Tab
- Notes Tab

Atomic Action Editor Parameters Area

Use the Atomic Action editor Parameters area to add required data elements from the Data Dictionary to the atomic action entity. You can view a hierarchical representation of the data elements.

You can modify data dictionary elements that you have imported into the Atomic Action editor without changing the default parameter values in other atomic actions that use the same data dictionary element. Do not modify data dictionary elements from within the data dictionary unless you want to make the change globally across all atomic actions that use the parameter.

Related Topics

Atomic Action Editor

About the Atomic Action Editor Context Menu

Modeling Activation Cartridge Project Data

Details Tab

Use the **Details** tab to edit the data element display name and define specific constraint values. When you select a data element in the Parameters area, the details for the selected data element appear in the Parameter Details area.

Field	Use
Select	(Optional) Select a data element that exists in the Data Dictionary, in case you want the selected parameter, in the Parameters area, to behave like a data element in the data schema. Click the **Type** label to open the data element.
Primitive Type	Displays the data type for simple data elements as in the data schema.
Name	Displays the name of the data element as in the data schema.
Display Name	Edit the data element display name. The Atomic Action editor supports multiple languages for this field. The field adjacent to **Display Name** displays your language. You can define a **Display Name** field value for any language you select from the list. If your preferences are set up to work in one language only, the system displays only the **[default]** option. See "Defining Language Preferences" for more information.
Path	Displays to which atomic action the parameter belongs.This field is read only.
Namespace	Not applicable in Activation.
Multiplicity	Select any one of the following: ■ **Required:** If the required parameters are not available, ASAP will not validate the work orders. The minimum and maximum values are both 1. ■ **Optional:** If the optional parameters are not available, ASAP will proceed validating the work orders. The minimum value is 0 and maximum value is 1. ■ **Range:** You can change **Minimum**, **Maximum**, and **Unbounded** only when you select **Range**. Depending on the minimum value, multiplicity will be either required or optional for an ASAP server. If minimum is 0, multiplicity is an optional parameter for ASAP. If minimum is greater than 0, multiplicity is a required ASAP parameter. Multiplicity can be seen in the build artifact file of the atomic actions in the generated SAR file.
Abstract and **Internal**	Not applicable in Activation.
Deprecated	Select to discourage use to the data element.
Length	Specify the minimum and maximum length for String data type. This field is read only for data elements that inherit from a parent element. You must define the minimum and maximum lengths with a non-negative integer between 0 and 999999999. Select **Unbounded** to define the maximum length as 999999999.
Default	Specify a default value. If there is no value in the work order for the parameter the default value is used.

Related Topics

Atomic Action Editor Parameters Area

Activation Tab

Activation Tab

Use the **Activation** tab to modify the run-time type parameter of the data element added in the **Parameters** tab of the Atomic Action editor. This modification is specific to the atomic action entity.

Field	Use
Compound Member Label	Displays the same name as the parameter in the Parameters area. You can use the compound member in the code generated Java Action processors (of java code) to access the compound members of a compound parameter. To rename the compound member, you must select **Refactoring** from the Atomic Action editor context menu. See "About the Atomic Action Editor Context Menu" for more information.
Runtime Type	Displays the parameter as in the Data Dictionary for this data element. You can select any one of the following run-time type parameters, specific to this atomic action entity, and define in this tab: ■ **SCALAR:** Conventional name-value pair parameters for simple data elements. This is the default when the run-time type parameter is left blank in the Data Schema editor. ■ **SCALARS:** Applicable to the root-level of structured data elements. The leaf (child) elements become conventional name-value pair parameters. This is the default when the run-time type is left blank in the Data Schema editor. For example, you add a structure element, Structure1, and a leaf child element, Child1, with the run-time type left blank in the Data Schema editor. You then add Structure1 to the Parameters area on the Atomic Action editor. Child1 is added along with Structure1. For Structure1, the default ASAP run-time type **SCALARS** appears in the **Activation** tab and the leaf child element inherits from the root and has **SCALAR**. Structured scalars elements are converted into individual ASAP scalar parameters by absorbing the structured element names into the scalar parameter name. The example used above for Structure1 and Child1 would appear by default as the ASAP scalar parameter structure1_child1. The default character used to separate the elements in the ASAP parameter names is the underscore (_). It is possible to change this character. See "Defining Design Studio Activation Preferences" for more information. ■ **COMPOUND:** Contains structures or arrays of information that are represented by a particular structure name or compound parameter name. ■ **XML:** Used as values for both information parameters and extended work order properties. ■ **XPATH:** Defines an XPath expression into XML data. Depending on the run-time type parameter, the labels in this tab are visible. See *ASAP Developer's Guide* for more information on parameter types.
Atomic Action Label	Displays the name of the parameter in the atomic action as in the Data Dictionary for this data element. You can specify a unique name for the atomic action, but this name will be specific to this atomic action entity.

Field	Use
Service Action Label	Displays the name of the parameter in a service action mapped to this atomic action. The name is unique and is same as the data element name.
Indexed	Select the check box if you want to index the run-time type parameter.
Data Restrictions	Specify any restriction on the value of the parameter.
Dependent XML	Displays the path of the XML that defines the parameter. This field is available only for the **XPATH** run-time type parameter.

Related Topics

Atomic Action Editor Parameters Area

Details Tab

Modeling Activation Cartridge Project Data

Atomic Action Editor Details Tab

Use the **Details** tab for selecting the routing type of an atomic action to the network element.

Field	Use
Routing Support	Select a routing type. Based on the selected routing type, the mandatory parameters for the routing type are automatically added to the Parameters area in the **Parameters** tab. **Note:** If you select **None**, no parameter is added in the Parameters area. It is recommended to select a routing type. See *ASAP System Administrator's Guide*.
Provide Parameter Count	Select to indicate that the Network Element Processor (NEP) should send the current index value for the atomic action.
Index Count	Specify the name of the parameter for obtaining the index value in Java provisioning classes.
Timeout (Second)	Specify the number of seconds before the ASAP server considers an atomic action in-progress as failed. The default value is **0**, which means ASAP server will not consider the atomic action in-progress as failed.
Select	Click to select an atomic action that is called to rollback the changes of the current atomic action in case of a failure scenario. Click X link, if you want to clear the atomic action from the field. For example, atomic action A is mapped to service action B. The rollback is configured on the service action. On the Atomic Action editor, in the **Details** tab, for the atomic action entity A, you select an atomic action Y. In case of a failure scenario, the service action B is rolled back and atomic action Y is called to rollback the action of atomic action A.
Retry	Select the check box to enable the **Retry Count** and the **Retry Interval** fields.
Retry Count	Specify the number of times the atomic action can be tried at the network element.
Retry Interval	Specify the time interval, in seconds, between each retry attempt by ASAP.

Related Topics

Atomic Action Editor

Defining Atomic Action Properties

Modeling ASAP Services

Atomic Action Editor Mappings Tab

Use the **Mappings** tab for mapping action processors to the atomic action.

Field	Use
Open	Click to open the action processor selected in the grid.
Clone	Click to create a copy of the action processor selected in the grid.
Remove	Click to clear the action processor from the grid.
Add	Click to map an action processor to the atomic action. The mapped action processor appears in the grid.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load for the selected action processor. You can select an atomic action to map to different action processors based on the vendor, technology, and software load. For example, later when the network element software version changes, you can create another action processor and map that one also to the same atomic action. As entries in the mappings cannot have duplicate vendor, technology, and software, you can update some of them accordingly. This allows you to use the same cartridge even when the software version changes.
Action Processor	Displays the name of the action processor selected in the grid.
Script	Displays the script that contains the method, as seen on the Action Processor editor, that is executed for the selected action processor.
Command Overview	Describes the expected behavior of the action processor when the script in the **Script** field is called.

Related Topics

Atomic Action Editor

Defining Atomic Action Properties

Modeling ASAP Services

Atomic Action Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the atomic action entity. This tab is read only.

Related Topics

Atomic Action Editor

Modeling ASAP Services

Service Action Editor

Use the Service Action editor to map the service action entity to atomic actions. In the Studio Projects view, double-click a service action entity to open the Service Action editor. You can create the editor using the Service Action Wizard.

> **Caution:** The Service Action editor for IP Service Activator service actions is read only. Service actions for IP Service Activator cannot be modeled in Design Studio, only viewed.

When working with the Service Action editor, see the following topics:

- Service Action Editor Editor Tab
- Service Action Editor Blueprint Tab
- Service Action Editor Realization Tab

Service Action Editor Editor Tab

Use the **Editor** tab for mapping the ASAP service action to atomic actions, defining properties for the service action, and display the parameter mapping of the service action and atomic actions.

The **Editor** tab for IPSA service actions is read only and cannot be used to modify or model the service action.

Field	Use
Description	Specify a description for the Service Action editor.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load for the service action entity in the cartridge project. There are three states for these fields as follows: ■ **Unknown:** When a service action entity is created but not mapped to an atomic action. ■ **Multiple:** When a service action is mapped to more than one atomic action. ■ *Vendor, Technology* and *Software Load*: When a service action is mapped to an atomic action, the exact values of the vendor, technology and software load are populated in these fields.
Action	Displays the action of the atomic action added in the Service Action Details area in the **Atomic Action(s)** tab. There are three states for this field as follows: ■ **Unknown:** When a service action entity is created but not mapped to an atomic action. ■ **Multiple:** When a service action is mapped to more than one atomic action. ■ *Action*: When a service action is mapped to an atomic action, the action of the atomic action is populated in this field.

When working with the **Editor** tab, see the following topics:

- Parameters Tab

- Atomic Action(s) Tab
- Command Overview Tab
- Properties Tab

Related Topics

Service Action Editor

Defining Service Action Properties

Modeling ASAP Services

Parameters Tab

Use the **Parameters** tab to view the mapping details of the service action and one or more atomic actions.

Field	Use
Summary List	Select to display in the Service Action Details area, the parameter details of corresponding one or more atomic actions as configured on the Atomic Action editor. On clicking in a specific row, the corresponding atomic action's parameter details are available in the Parameter Map area.
Complete Parameter Map	Select to display in the Service Action Details area, the parameter details of the service action and corresponding one or more atomic actions as configured on the Atomic Action editor.

Related Topics

Atomic Action(s) Tab

Command Overview Tab

Properties Tab

Atomic Action(s) Tab

Use the **Atomic Action(s)** tab to map the service action to one or more atomic actions. In the Service Action Details area, you can define the conditions for mapping the service action and the atomic actions.

Service Action Details

Field	Use
Seq	Displays the sequence number for each atomic action added in the Service Action Details area.
Atomic Action	Displays the atomic actions mapped to the service action.
Condition	Displays the condition defined for spawning an atomic action.
Label, Value, and **Expression**	Displays the parameter name, value, and regular expression specified in the Atomic Action Conditions area based on the selected condition.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load of the atomic action mapped to an action processor.

Field	Use
Action	Displays the action of the atomic action added in the Service Action Details area in the **Atomic Action(s)** tab. If the service action is mapped to atomic actions with different actions then **Multiple** appears.
Rollback Point	(Optional) Select any one of the following: - **State:** The atomic action is the PNR for partial rollback. If rollback occurs and execution continues beyond this point, rollback occurs to the atomic action but not further. - **Stop:** Beyond the specific atomic action, no rollback can occur. The default behavior of rollback, when nothing is selected in the Rollback Point column, is a complete rollback of all the atomic actions in a failed work order. **Note:** If you want an atomic action to rollback, you must select the **Rollback** check box in the **Properties** tab.
Open	Click to open the atomic action in the Atomic Action editor.
Remove	Click to remove the mapping of the service action and the atomic action from the editor.
Add	Click to map an atomic action to the service action.

Atomic Action Conditions

Field	Use
Always	Select for ASAP to always generate this atomic action command for the specific service action.
Equals	Select for ASAP to generate this atomic action command if the service action parameter stated in the **Parameter Label** field, in the Atomic Action Conditions area, is defined on the service action and has the parameter value as specified in the **Parameter Value** field.
Defined	Select for ASAP to generate this atomic action command if the service action parameter specified in the **Parameter Label** field, in the Atomic Action Conditions area, is defined on the service action.
Not Defined	Select for ASAP to generate this atomic action command if the service action parameter stated in the **Parameter Label** field, in the Atomic Action Conditions area, is not defined on the service action.
Parameter Label	Select a label name from the list.
Parameter Value	Specify a value in the field. This field is available when you click the **Equals** button.

Field	Use
Include Expression	Define a logical expression using a number of criteria for a service action parameter. The range of options available allow an atomic action to be generated if the service action parameter value is within a range of values, or if the service action parameter is greater than, or less than, or equal to, a specified value. More than one condition can be combined in the expression using AND or OR operator. For example, a service action, ADD_PHONE, is mapped to an atomic action, A, that has three integer parameters, namely PARAM1_VALUE, PARAM2_VALUE and PARAM3_VALUE. You can specify an expression like: `(PARAM1_VALUE LIKE "0")OR((PARAM1_VALUE LIKE PARAM2_VALUE)AND(PARAM2_VALUE !LIKE PARAM3_VALUE))` The atomic action, A, from the service action, ADD_PHONE, is spawned only if the expression matches.

Related Topics

Parameters Tab

Command Overview Tab

Properties Tab

Command Overview Tab

Use the **Command Overview** tab to view the service action and atomic action details.

Related Topics

Parameters Tab

Atomic Action(s) Tab

Properties Tab

Properties Tab

Use the **Properties** tab for defining values for the service action.

Field	Use
Level	Select a sequence level for the service action within the work order. See *ASAP Developer's Guide*.
Service Action Completion Event	(Optional) Select an event from the list that will be triggered when the service action is complete. See *ASAP Developer's Guide* and *ASAP System Administrator's Guide*.
Rollback	Select the **Rollback** check box to configure rollback on the service action. Ensure to select a rollback value in the **Atomic Action(s)** tab.
Service Action Failure Event	(Optional) Select an event from the list that will be called when the service action fails. See *ASAP Developer's Guide* and *ASAP System Administrator's Guide*.

Related Topics

Parameters Tab

Atomic Action(s) Tab

Command Overview Tab

Service Action Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the service action entity. This tab is read only.

Related Topics

Service Action Editor

Service Action Editor Editor Tab

Modeling ASAP Services

Service Action Editor Realization Tab

Use the Service Action editor **Realization** tab to associate ASAP service actions with a conceptual model technical action.

Field	Use
Realizes	Click to open the selected technical action in the Action editor.
Select	Associates the ASAP service action with a conceptual model technical action. To select from the available conceptual model technical actions, there must be a dependency defined between your ASAP cartridge project and the Model project in which your conceptual model is saved. See "Managing Project Dependencies" for more information.

Related Topics

Working with Conceptual Models

Service Action Editor

Event Template Editor

Use the Event Template editor to customize the parameters that are returned when an ASAP event is triggered. If there is no matching template for an event type, no parameters are returned. You can configure static event templates, using the editor, for just four types of events like Order Startup event, Order Complete event, Order Timeout event, and Order Fail event.

Notes:

- The dynamic event templates have precedence over static ones. Therefore, if there is any matching dynamic event template, no static event template will be checked. See *ASAP System Administrator's Guide* for more information.
- Depending on the variable INCLUDE_SERVICE_ACTION_DETAIL, the service action parameters are returned. See the *ASAP Server Configuration Guide* for more information.

In the Studio Projects view, double-click an event template entity to open the Event Template editor. You can create the editor using the Event Template Wizard.

When working with the Event Template editor, see the following topics:

- Event Template Editor Editor Tab
- Event Template Editor Blueprint Tab

Event Template Editor Editor Tab

Use the **Editor** tab to define values for an event template. When ASAP is processing an event, ASAP will check if there exists an event template based on the service action, order parameter name and order parameter value, or on all the three: service action, order parameter name, and order parameter value. These details are configured in the **Details** tab. Based on the event name in the **Details** tab, the configuration in the **Return Data Set** tab is selected. The event, in the **Details** tab, is populated with the parameters configured in the **Return Data Set** tab.

Field	Use
Add	Click to add the default event type, **Order Complete Event**, to the Event Templates area.
Remove	Click to clear an event template from the editor.

When working with the **Editor** tab, see the following topics:

- Details Tab
- Return Data Set Tab

Details Tab

Use the **Details** tab to define a selection criteria for selecting an event template in case of an ASAP event.

Field	Use
Use recommended name	Click to enable the **Event Template Name Suffix** field. In the **Event Template Name Suffix** field, you can specify a suffix to the event template name. You can see the suffix being added in the grayed out **Event Template Name** field. The event template name in this case will be E*_Vendor-Technology_SoftwareLoad_EventTemplateNameSuffix*.
Event Type	Select an event type from the list. **Note:** You can apply event templates to just the four event types like Order Startup event, Order Complete event, Order Timeout event, and Order Fail event.
Event Template Name	If the **Use recommended name** check box is deselected, you can specify a name to the event template.
Order Parameter Name	(Optional) Specify a name to the order parameter. This name is used together with the order parameter value**.**
Order Parameter Value	(Optional) Specify a value to the order parameter. This value is used together with the order parameter name.
Select	(Optional) Click to select a service action that is applicable for only matching Order Fail event type events. The matching is against the service action name in the Order Fail event type events. **Note:** Only Order Fail event type events include a service action name that identifies the service action that was executed when the failure occurred.

Field	Use
Open	Click to open the service action entity in the Service Action editor.
Clear	Click to clear the service action displayed in the **Service Action** field.
Event Template Desc	Specify a description to the event template.

Related Topics

Return Data Set Tab

Return Data Set Tab

Use the **Return Data Set** tab to configure parameter type related values that will be returned for an event type. Only the parameters that are defined in this tab for an event template are returned in the related work order events. If no parameter is defined in this tab for an event template, no parameter will be returned for a defined event type.

Field	Use
Parameter Type	Select a parameter type from the list that will be returned when an ASAP event is triggered.
Add	Click to add a row with a default parameter type and parameter name to the **Return Data Set** tab in the Event Template Details area. You can add rows in the **Return Data Set** tab for each of the parameter types.
Remove	Click to clear a parameter type's related details.
Parameter Name	Specify a parameter name, for the selected parameter type, that will be returned when an ASAP event is triggered.
Select	Click to select a service action. If a service action is selected, the service action parameter in the **Parameter Name** will be returned only for the specific service action. If a service action is not selected and the parameter in the **Parameter Name** is a service action parameter, the service action parameter is returned in the event for each service action that has the parameter in the service model.
Return Data Set	Specify a description for the return data set.

Related Topics

Details Tab

Event Template Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the event template entity. This tab is read only.

Related Topics

Event Template Editor

6 Modeling Entities

After modeling the service for an Activation Network cartridge, you develop implementations for the following entities:

- Action Processor
- Network Element (NE) Connection Handler
- User Defined Exit Types

Additionally, you can create custom action processors and configure retry properties and network element instances.

When modeling implementations for an Activation Network cartridge, see the following topics:

- Working with the Action Processor
- Working with Java NE Connection Handlers
- Working with User-Defined Exit Types
- Working with Custom Action Processors
- Working with Retry Properties
- Working with Network Element Instance Throughput Control
- NE Connection Handler Editor
- User Defined Exit Type Editor

Working with the Action Processor

For every action processor you need to define a processor as the implementation that performs the work.

While you can use a classic Java implementation or a state table implementation, the recommended approach is to implement using Java with code generation. This method provides code that would normally have to be written by developers.

Related Topics

Understanding Java with Code Generation

Understanding the Java Processor Class

Understanding Java Libraries in Design Studio

Understanding Unit Testing

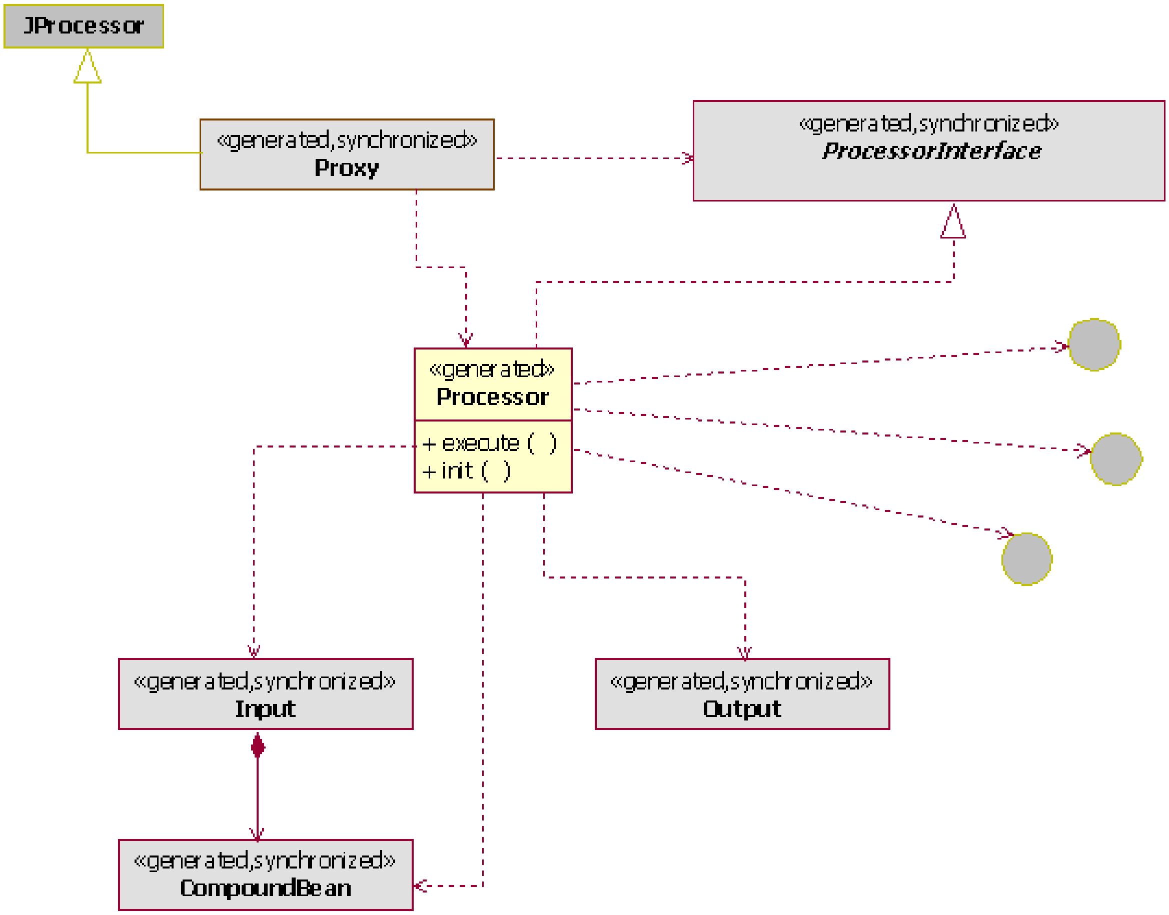
Configuring User-Defined Exit Types

Working with Java NE Connection Handlers

Modeling Entities

Understanding Java with Code Generation

The Java with code generation implementation for an action processor creates a Java processor that composes messages to be sent to a device, evaluates the response for errors, extracts output information from the response, and populates the information into output parameters.



In the Java processor with code generation, the central class is the Processor, which is editable by the developer. The Processor is generated only once and includes sample code based on the associated parameters at creation time. You should delay the creation of the processor until the action processor is associated with an atomic action that has fully defined parameters. If parameters are not yet defined or the action processor is not yet associated with the atomic action, then the generated sample code will be incomplete and, because it is generated only once, would require additional coding.

> **Note:** Synchronized classes or interfaces are rebuilt every time you save changes of atomic action parameters (for example, classes and interfaces are synchronized with the model and reflect the model). Therefore, you should never make code changes to any synchronized class or interface. Oracle Communications Design Studio overwrites these changes when you run the next build (with changes in the model). Developers should write code only for the processor class.

There are 2 methods in the Processor:

- `execute`
- `init`

The main method is `execute`. When called, it is provided with the following:

- A number of classes to perform operations.
- An input class that contains all input parameters.
- An output class to populate the output parameters.
- Access to a logger.
- An implementation of the exit type to match responses against user-defined exit types and to set the exit type for the processor.
- Access to the Connection Handler to send requests and get responses from a connected device.

Related Topics

About Processor Classes and Interfaces

Example: Typical Processor Call Sequence

Working with the Action Processor

About Processor Classes and Interfaces

The following classes and interfaces are used by the Processor:

InputBean

The InputBean is generated and synchronized, tied to the parameters of the atomic action (has set and get methods for all parameters of the atomic action), and provides setters and getters for manipulating parameters.

If the parameter is a scaler (simple type), it is received as a string and can be used immediately

CompoundBean

If the parameter is a compound parameter with named members, the InputBean returns another bean that represents the compound. The returned bean has convenience methods to get the members within the compound. A compound bean for every defined type of compound parameter is created. You also get a set of instances of these beans based on the work order (you get a list of these). If the compound parameter does not have named members, it provides a vector of the members.

Note: Specify the compound members whenever possible. Indicating the members will simplify the coding required and eliminate possible code to mode synchronization issues.

Multi-instance Compound parameters start at index one (e.g. CMPD[1]).

Bracket type (Index Parameter Identification Tokens) and delimiter (Indexed Parameter Delimiter) settings are configured on the Project editor **Cartridge Locations** tab in the Code Generation area. Design Studio applies these settings to all generated code within the cartridge. The following examples assume the defaults (square brackets with a period delimiter).

The following example shows a scalar parameter.

```
Service Action Parameter Name: SCALAR
Atomic Action Parameter Name: SCALAR
Order Format:
     SCALAR
Usage:
     String myscalar = parms.getMyScalar();
```

The following example shows a compound parameter with no members specified.

```
Service Action Parameter Name: CMPD
Atomic Action Parameter Name: CMPD
Order Format:
Entries will have the compound name as a prefix. There may be multiple entries
with that prefix. For example, a compound named "CMPD" may have the following
entries on an order.
     CMPD
     CMPD.X
     CMPD.Y
     CMPD.Z
Usage:
     String mycmpd = parms.getMyCmpd
       String x = parms.getMyCmpd ("X");
       String y = parms.getMyCmpd ("Y");
       String z = parms.getMyCmpd ("Z");
```

The following example shows a compound parameter with members.

```
Service Action Parameter Name: CMPDMBR
Atomic Action Parameter Name: CMPDMBR
Order Format:
     CMPDMBR.A
     CMPDMBR.B
     CMPDMBR.C
Usage:
     MyCmdMbrBean mycmpdmbr = parms.getMyCmpdMbr();
       mycmpdmbr.getA();
       mycmpdmbr.getB();
       mycmpdmbr.getC();
```

The following example shows a multi-instance compound parameter with no members specified.

```
Service Action Parameter Name: CMPDMULTI
Atomic Action Parameter Name: CMPDMULTI
Order Format:
```

```
Entries will have the compound name as a prefix. There may be multiple entries
with that prefix. For example, a compound named "CMPDMULTI" may have the following
entries on an order.
     CMPDMULTI[1]
     CMPDMULTI[1].X
     CMPDMULTI[1].Y
     CMPDMULTI[1].Z
     CMPDMULTI[2].X
     CMPDMULTI[2].Y
     CMPDMULTI[2].Z
Usage:
     String mycmpdmulti = parms.getMyCmpdMulti ();
       String x1 = parms.getMyCmpdMulti (1, "X");
       String y1 = parms.getMyCmpdMulti (1, "Y");
       String z1 = parms.getMyCmpdMulti (1, "Z");
       String x2 = parms.getMyCmpdMulti (2, "X");
       String y2 parms.getMyCmpdMulti (2, "Y");
       String z2 = parms.getMyCmpdMulti (2,"Z");
```

The following example shows a multi-instance compound parameter with members.

```
Service Action Parameter Name: CMPDMULTIMBR
Atomic Action Parameter Name: CMPDMULTIMBR
Order Format:
     CMPDMULTIMBR[1].A
     CMPDMULTIMBR[1].B
     CMPDMULTIMBR[1].C
     CMPDMULTIMBR[2].A
     CMPDMULTIMBR[2].B
     CMPDMULTIMBR[2].C
Usage:
     MyCmpdMultiMbrBean[] mycmpdmultimbr = parms.getMyCmpdMultiMbr();
     for (int i = 0; i <mycmpdmultimbr.length; i++)
     {
       MyCmpdMultiMbrBean bean = mycmpdmultimbr[i];
       bean.getA();
       bean.getB();
       bean.getC();
     }
```

The following example shows an indexed compound parameter with no members.

```
Service Action Parameter Name: CMPDIDX[++]
Atomic Action Parameter Name: CMPDIDX
Order Format:
Entries will have the compound name as a prefix. There may be multiple entries
with that prefix. For example, a compound named "CMPDIDX" may have the following
entries on an order.
     CMPDIDX[0]
     CMPDIDX[0].X
     CMPDIDX[0].Y
     CMPDIDX[0].Z
     CMPDIDX[1].X
     CMPDIDX[1].Y
     CMPDIDX[1].Z
Usage:
     String mycmpdidx = parms.getMyCmpdIdx();
       String x = parms.getMyCmpdIdx ("X");
       String y = parms.getMyCmpdIdx ("Y");
       String z = parms.getMyCmpdIdx ("Z");
```

Note: The implementation will be called multiple times, providing one instance of the compound during each call.

The following example shows a compound parameter with members.

```
Service Action Parameter Name: CMPDIDXMBR[++]
Atomic Action Parameter Name: CMPDIDXMBR
Order Format:
     CMPDIDXMBR[0].A
     CMPDIDXMBR[0].B
     CMPDIDXMBR[0].C
     CMPDIDXMBR[1].A
     CMPDIDXMBR[1].B
     CMPDIDXMBR[1].C
Usage:
     MyCmpdIdxMbrBean mycmpdidxmbr = parms.getMyCmpdIdxMbr();
       mycmpdidxmbr.getA();
       mycmpdidxmbr.getB();
       mycmpdidxmbr.getC();
```

Notes:

- The implementation will be called multiple times providing one instance of the compound during each call.
- For multi-instance compounds, member parameters cannot be set as required because the system cannot determine whether a member is present or if there are additional entries.

Output

The Output class enables you to populate output parameters. There are convenience methods for populating parameters to varying scope within a work order. Examples of parameters are as follows:

- Action parameters are available to the service action.
- Input parameters.
- Global parameters are available to everything.
- Rollback parameters enable you to populate for the rollback action if it is defined within the atomic action.

Note: The output parameters are not explicitly defined in the model, so there are no convenience methods. To set a parameter you need to know its string name and include it.

ILogger

`ILogger` is an interface for debug logs. When the processor is running on the Oracle Communications ASAP system, it logs to the Diagnosis log. If you are running the processor in JUnit, you can use other implementations of logger to log to the console instead.

IExitType

IExitType enables you to set the exit type explicitly or by matching a response string against the user-defined exit types.

IConnectionHandler

IConnectionHandler is an instance of the Connection Handler that is associated with the vendor, technology, and software load of the action processor. For the Telnet Connection Handler, the basic methods on the interface can be used to send requests (because it is string-based). For technologies (for example, SOAP or XML) that provide multiple convenience methods, the Processor may want to test the type of Connection Handler and pass it to a more specific Connection Handler to obtain access to the convenience methods. If you want to expose more explicit methods when writing a Connection Handler, you can define an interface that extends the IConnectionHandler and ensure that those methods are available through that interface. The Processor should always use an interface when interacting with the ConnectionHandler, to achieve the implementation in more than one way and allow for unit testing. For more information about unit testing, see "Understanding Unit Testing".

Processor

Processor needs to implement the processor interface, which is generated in the code generation and kept synchronized with the model.

Proxy

Proxy is situated between the NEP and Processor class and manages the interaction between them. Proxy sets up all classes used by the Processor and initiates and calls the Processor. Most importantly, the proxy simplifies the work required by the Processor by:

- Creating all instances of the InputBean and initializes CompoundBeans so they are available and populated through the processor.
- Performing much of the standard logging, including the entry and exit of the processor and the contents of the parameters passed in for debugging.
- Extending the JProcessor. This isolates the portion of the Java processor code that needs to relate directly to the version of the activation, and allows the processor, its interface, and all its related classes and interfaces to run outside of the ASAP system and, therefore, to be unit tested

Note: When creating a Java processor from the action processor editor, the resulting class name is "Proxy" because the proxy gets initiated by the NEP (Proxy is registered to be called in the activation). When you open that implementation it opens to the Processor class, where you write your code for editing.

Related Topics

Understanding Java with Code Generation

Working with the Action Processor

Modeling Entities

Example: Typical Processor Call Sequence

The proxy:

1. The proxy creates the input, the output, and the exit type classes.
2. The proxy populates the exit type classes and initializes them.
3. The proxy creates the processor that will be called and initializes it.
4. If the logger needs to be used by the processor, the proxy provides this during the `init` method call.
5. The proxy invokes the processor by calling the `execute` method with the input, output, connection, and exit type.
6. The processor obtains parameters from the InputBean to compose a message or command to be sent to a device.
7. The processor calls the `send request` to send that message to the device.
8. The processor sets the exit type based on the response.
9. The processor sets output parameters based on the response.

   The processor may parse the response to obtain additional values for populating the output parameters.
10. The proxy cleans up the processor.
11. The proxy looks at the exit type that was set and populates it for return to the NEP, and cleans up the exit type.
12. The proxy extracts all output parameters for return to the NEP and to populate the work order.

    The proxy then cleans up this class and (16) the remaining classes.

Note: The developer is only responsible for the items related to the processor (steps 6 through 9); the proxy handles all other items.

Related Topics

Understanding Java with Code Generation

Working with the Action Processor

Modeling Entities

Understanding the Java Processor Class

You need to write the logic in the `execute` method of the processor class (there is a processor class for each atomic action) to achieve the desired action in a switch. Use the Java editor in the Package Explorer view of the Java perspective to write all code.

When implementing the action processor, Design Studio provides you with support such as automatic generation of code and sample data. In Design Studio, this is currently set up specifically for the Telnet protocol (SOAP, CORBA and other protocols require more coding; for example, you must write your own logic for send methods, requests, to extend the connection class, and so on.).

Code for the processor is autogenerated by the proxy (getter and setter methods for each parameter) which provides you with an API to manipulate the data. For example, for an incoming object, methods such as `getBilling` are autogenerated (the type of methods depend on the parameters specified in the service model and how they are mapped). You can use these autogenerated methods in the processor class to get the value for the parameters.

See "Understanding Java with Code Generation" for more information about autogenerating processor code.

To obtain the required method to get a value for a parameter, type the name of the parameter followed by a dot. This displays all available methods for the parameter.

Notes:

- Do not modify the autogenerated code. All changes are overwritten during any subsequent builds.
- Code generation does not overwrite existing Java code. If you change the target implementation, the old code remains. Clean up old code when creating a new Java implementation for an action processor with an existing Java implementation.

Related Topics

Writing Java Processor Execute Method Logic

Working with the Action Processor

Modeling Entities

Writing Java Processor Execute Method Logic

The basic development steps to write the logic for the execute method of a Java Processor class are as follows:

To write Java Processor execute method logic:

1. Extract parameters from InputBean (retrieve information).
2. Use these parameters to build a command.
3. Send a message or command to the switch using the send request in Telnet.
4. Handle the response by setting the user-defined exit type.

   See "Creating New User-Defined Exit Types" for more information about setting the user-defined exit type.
5. Using the OutputBean, you have the option to return some parameters upstream to log, infoparm, and so on.

Occasionally, for Telnet, you may need to build some helper classes, perform data derivation, and create parsers.

Related Topics

Understanding the Java Processor Class

Working with the Action Processor

Modeling Entities

Java Processor Class Example

The following is a coding example for an atomic action.

From the Java editor of the Package Explorer view:

1. Create an atomic action and an action processor.

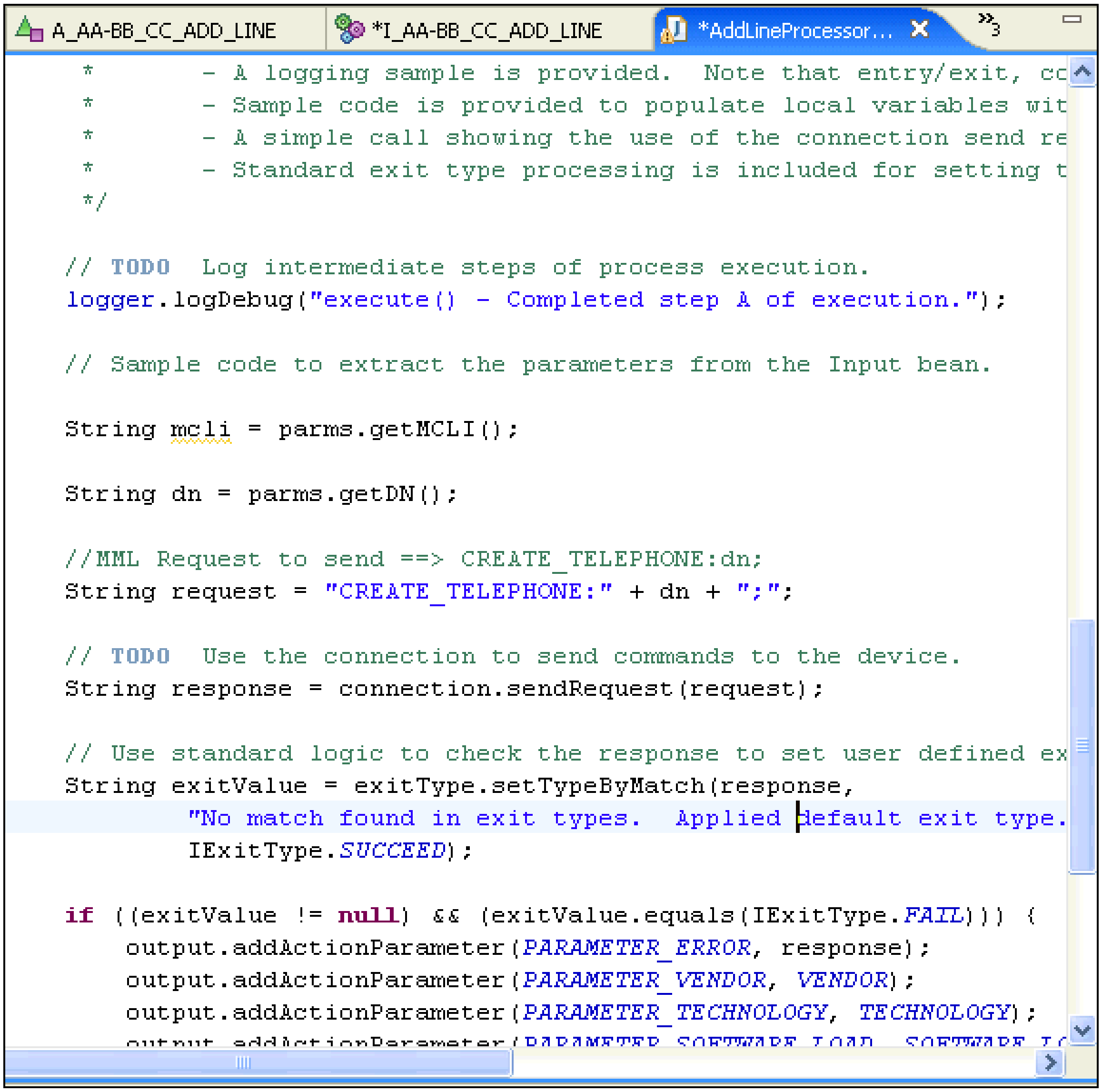
   See "Creating Model Elements" for more information.

2. Select an ADD action and LINE entity for each.
3. In the Atomic Action editor, map the action processor to the atomic action.

   See "Understanding Model Element Relationships" for more information.
4. In the **Parameters** tab, right-click in the Parameters area and select **Simple Data Element** or **Select Structured Data Element**. See "Adding Existing Simple and Structured Data Elements to Entities" for more information.
5. In the Parameter Details area, define values to the ASAP run-time type parameters.
6. In the **Mappings** tab, click the mapped action processor to open the Action Processor editor.
7. Select Java Action Processor (with Code Generation).
8. Click **New**.

   The Studio Activation Java Implementation Wizard appears.
9. Click **Finish**.

   A new atomic action class appears in the Java editor with basic code, such as the entry point, `get` functions to get the parameters defined in the atomic action, and, in the end portion, `sendRequests` and how the response is handled.



```
     *        - A logging sample is provided.  Note that entry/exit, cc
     *        - Sample code is provided to populate local variables wit
     *        - A simple call showing the use of the connection send re
     *        - Standard exit type processing is included for setting t
     */

    // TODO  Log intermediate steps of process execution.
    logger.logDebug("execute() - Completed step A of execution.");

    // Sample code to extract the parameters from the Input bean.

    String mcli = parms.getMCLI();

    String dn = parms.getDN();

    //MML Request to send ==> CREATE_TELEPHONE:dn;
    String request = "CREATE_TELEPHONE:" + dn + ";";

    // TODO  Use the connection to send commands to the device.
    String response = connection.sendRequest(request);

    // Use standard logic to check the response to set user defined ex
    String exitValue = exitType.setTypeByMatch(response,
            "No match found in exit types.  Applied default exit type.
            IExitType.SUCCEED);

    if ((exitValue != null) && (exitValue.equals(IExitType.FAIL))) {
        output.addActionParameter(PARAMETER_ERROR, response);
        output.addActionParameter(PARAMETER_VENDOR, VENDOR);
        output.addActionParameter(PARAMETER_TECHNOLOGY, TECHNOLOGY);
        output.addActionParameter(PARAMETER_SOFTWARE_LOAD, SOFTWARE_LC
```

10. Build the MML (the message itself) using the Eclipse message format.

    Use methods and classes. If no parameters are available in the atomic action, use a string to build the MML.

11. Send the message to the switch using the send request.

12. Set the user-defined exit type.

Related Topics

Understanding the Java Processor Class

Working with the Action Processor

Modeling Entities

Creating Model Elements

Understanding Model Element Relationships

Understanding Java Libraries in Design Studio

Understanding Unit Testing

Working with User-Defined Exit Types

Loading XML String Parameter into XML Related Technologies

The autogenerated code produces content in string format for all parameter types including the XML run-time parameter type. However, some network elements require different XML technologies such as document object model (DOM), simple API for XML (SAX), XML beans, and so on. You can create Java code to perform this conversion.

For example, the following sample code loads an XML string parameter into a XML DOM parameter:

```
public Document buildDocumentFromString(String xmlString)
throws Exception
{
try
{
StringReader stringReader = new StringReader(xmlString);

SAXBuilder builder = new SAXBuilder();

Diagnosis.diag(1, this, "Building document for :\n" + xmlString);
Document doc = builder.build(stringReader);
return doc;
}
catch (Exception e) {
String exceptionMessage = "Exception caught : " + e.toString();
Diagnosis.diag(1, this, exceptionMessage);
throw new Exception("XML Error" + exceptionMessage);
}
}
```

Related Topics

Understanding the Java Processor Class

Working with the Action Processor

Modeling Entities

Creating Model Elements

Creating Activation Run-Time Type Parameters in the Data Dictionary

Understanding Model Element Relationships

Understanding Java Libraries in Design Studio

Understanding Unit Testing

Working with User-Defined Exit Types

Understanding Java Libraries in Design Studio

There are several types of Java libraries available in Design Studio:

Activation libraries

Activation libraries are utilized by many cartridges and include the following:

- **studio_2_6_0.jar**

- **asaplibcommon.jar**
- **JInterp.jar**

Activation libraries are automatically added to the project when you create an action processor. They are added to the project classpath to enable the Java development toolkit access.

Notes:

- The **studio_2_6_0.jar** file is not installed by the ASAP installation. The **studio_2_6_0.jar** must be added to the ASAP installation prior to deployment of a Design Studio-created cartridge. Configure the **studio_2_6_0.jar**, **asaplibcommon.jar**, and **JInterp.jar** files on the server. See the discussion of installing a cartridge using Design Studio in *ASAP Installation Guide*.
- When you are packaging a cartridge, exclude the **studio_2_6_0.jar**, **asaplibcommon.jar**, and **JInterp.jar** files. These JAR files are installed on the Activation server and are shared by all cartridges If you include these JAR files, Design Studio generates an error.

Other Libraries

Add other libraries to the **lib** folder under the project. Update the Java project properties to set the Java buildpath to make use of those libraries. See Eclipse help for adding folder or packages to the Java buildpath.

In the Project editor **Packaging** tab, select **Libraries** to display any jar files contained in the **lib** folder.

Related Topics

Understanding Java with Code Generation

Understanding the Java Processor Class

Understanding Java Libraries in Design Studio

Understanding Unit Testing

Working with User-Defined Exit Types

Packaging Activation Cartridges

Working with the Action Processor

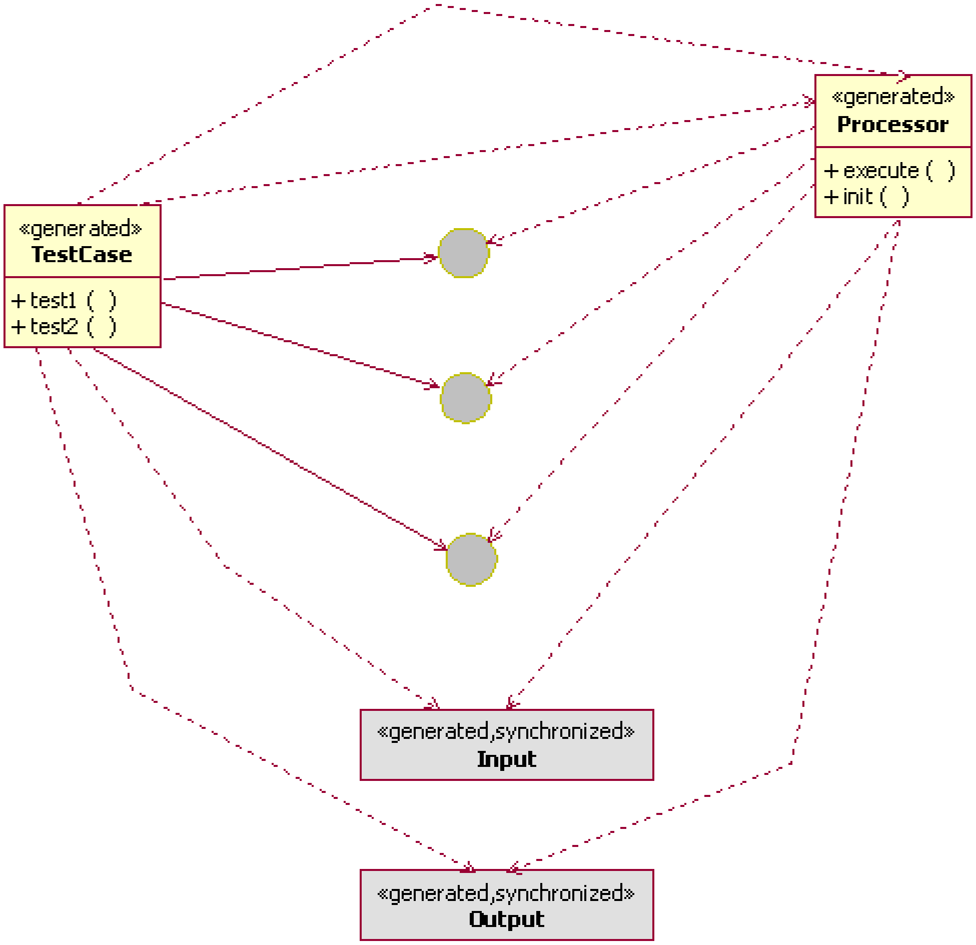
Understanding Unit Testing

Unit testing in Design Studio does not need to be implemented to complete a cartridge, although it is highly recommended for these reasons:

- Unit testing contributes to building quality code.
- Unit testing provides repeatable tests for regression.

You can test the processor outside of the ASAP system because the interfaces and generative classes of the Java processor are all independent of the ASAP system and its classes (the generated InputBeans and output are not tied to ASAP). To run the processor, a `TestCase` is generated once (with a sample test based on information at the time of creation), after which the developer owns it and can extend it.

The unit test framework initiates all tests in test subfolder. Unit testing is implemented as a JUnit test. JUnit tests can optionally be run with the JDT Debugger.



The `TestCase` simulates the proxy for each individual test, and:

- Creates an implementation of the interfaces, either the real implementation or a stubbed test implementation.
- Generates input and output beans.
- Invokes the processor.

The `TestCase` is a JUnit`TestCase`. Each `TestCase` can contain many tests, and each test is defined by a no-parameter method beginning with "test".

The generated `TestCase` has a framework that provides a test. The test runs based on input files, which find the data and test criteria for a particular test. This framework enables developers to create simple files to define new tests. This works for any standard type of test where you pass in data and check the request to ensure it was sent as expected, and that the returned exit type is the one you expected. Also, this allows for a simple, standard response to be used inside the test.

Sample test classes are provided for simulating `IExit` and `ILogger`. A base output class provides the methods required for output classes.

Related Topics

Running Unit Test Cases

Running Unit Tests with the JDT Debugger

Understanding Unit Test Property Files

Configuring a Unit Test

Working with the Action Processor

Modeling Entities

Running Unit Test Cases

Run the `TestCase` class as a JUnit test, or as a Java application. Running as a JUnit test provides a richer user experience by providing results in the JUnit view. Running as a Java application allows the `TestCase` to be run as part of an automated test framework. Java application test case results appear in the Eclipse IDE in the Console view.

To run unit test cases:

1. Right click the TestCase class and select **Run As**.
2. Select **JUnit Test** or **Java Application**.

   Design Studio displays the results in the JUnit view or Console view, depending on your selection in step 2. Logging information is sent to the Console View.

Related Topics

Understanding Unit Testing

Working with the Action Processor

Modeling Entities

Running Unit Tests with the JDT Debugger

To run unit test cases with the JDT debugger:

1. Set breakpoints in your Processor class as desired.
2. Right click the TestCase class and select **Run As**.
3. Select **JUnit Test** or **Java Application**.

   The unit test is executed and the debugger will break as appropriate, allowing for full debugger functionality, including variable inspection and code stepping.

Related Topics

Understanding Unit Testing

Working with the Action Processor

Modeling Entities

Understanding Unit Test Property Files

You use a set of property files to set up a unit test (both are property file and follow the Java property file format):

- **testdata** file (for example, **TestExample.testdata**)
- **testinfo** file (for example, **TestExample.testinfo**).

Note: The **testinfo** file is optional. Design Studio uses defaults if it is not present.

Testdata file

The **testdata** format for naming the input parameters is similar to that within a work order. However, you must populate the test data with atomic action labels (and not service action labels). Run the unit test as if the parameters have been previously defaulted.

Apply the defaults that are normally set by the SARM (based on what is configured in the atomic action) as if they had been applied in the test data (the processor runs after those defaults have been set). The unit test data should be based on data that has already been defaulted and based on names relating to the atomic action label (and not the service action label).

When you fill in the test data for compounds or incoming repeating elements, use square brackets to indicate the index for a compound as in the following example.

```
# Example Action Processor input property file
NETID=ERIC-SDP_3-6-2-HOST
MSISDN=0701234567
FAF_LIST[1].FAF_N=0701237777
FAF_LIST[1].TSC=O
FAF_LIST[1].RCO=1
FAF_LIST[1].K=400
FAF_LIST[2].FAF_N=07052
FAF_LIST[2].TSC=4
FAF_LIST[2].K=100
FAF_LIST[3].FAF_N=071
FAF_LIST[3].K=500
```

Testinfo file

You can use this optional file to define the properties for which you are testing. You can also define what expected request the processor should create, the expected canned response returned to the processor, the expected exit type and whether it should be tested.

Note: If you do not define a **testinfo** file, then by default the test case only tests whether the exit type is *succeed* (that is, to confirm that the test data has gone through).

```
# Example Action Processor test info property file
request.check=true
request.value=Test Message
response.value=Test Response
# Exit Type values:
# SUCCEED
# FAIL
# RETRY
# MAINTENANCE
# SOFT_FAIL
# DELAYED_FAIL
# STOP
exittype.check=true
exittype.value=SUCCEED
```

If you wish to have multiple request and response values in your test, you can specify multiple values in the **testinfo** file. Add a dot separated numeric suffix to the value (starting at 1).

If your request or response has multiple lines or special character, follow the standard Java property guidelines.

```
# Example Action Processor test info property file
request.check=true
request.value.1=Test Message 1
request.value.2=Test Message 2
response.value.1=Test Response 1
response.value.2=Test Response 2
# Exit Type values:
# SUCCEED
# FAIL
# RETRY
# MAINTENANCE
# SOFT_FAIL
# DELAYED_FAIL
# STOP
exittype.check=true
exittype.value=SUCCEED
```

Related Topics

Understanding Unit Testing

Working with the Action Processor

Modeling Entities

Configuring a Unit Test

To configure a unit test:

1. Select **File**, select **New**, then select **File**.
2. Create a file ***name*.testdata**.

   For example, you might create a file called **TestExample.testdata**.

   > **Note:** Place this file in a subfolder of the action processor implementation package named **test**.

3. Enter the text for the file.

   The file format is a Java property file, so each entry specifies the parameter and its value.

4. Repeat steps 1 and 2 as necessary to create a second file ***name*.testinfo**.

   For example, you might create a file called **TestExample.testinfo**.

Related Topics

Understanding Unit Testing

Working with the Action Processor

Modeling Entities

Working with Java NE Connection Handlers

The NE Connection Handlers with Java implementation manage the connections with network elements based on the communication parameters in an NE Template.

When implementing Java NE Connection Handlers, see the following topics:

- About Java NE Connection Handlers
- Creating New NE Connection Handlers
- Generating a Telnet NE Connection Handler Implementation
- Generating a Custom NE Connection Handler Implementation

About Java NE Connection Handlers

The Java implementation NE Connection Handler needs to implement the `IConnectionHandler` interface, which provides a common interface for interacting with connections and requires few methods to be written.

Different types of NE Connection Handlers can be created:

- Telnet: When you create a new telnet NE Connection Handler, it generates code for telnet connections. This extends the telnet connection to support the interface. The NE Connection Handler editor indicates where additional code is required.
- Custom: Create this NE Connection Handler if the connections are not telnet. Custom Connection Handlers generate a skeleton to implement the `IconnectionHandler` and extends the base NE connection class. The NE Connection Handler editor indicates where additional code is required.

Related Topics

Creating New NE Connection Handlers

Generating a Telnet NE Connection Handler Implementation

Generating a Custom NE Connection Handler Implementation

Modeling Entities

Creating New NE Connection Handlers

You use the NE Connection Handler Wizard to create new NE Connection Handler entities.

To create a new NE Connection Handler entity:

1. Select **Studio**, select **New**, select **Activation**, then select **NE Connection Handler**.

   The NE Connection Handler Wizard appears.

2. Select the project for this element and enter a name for the entity.
3. (Optional) Select a location for the entity.

   By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

   a. Click the **Folder** field **Browse** button.
   b. Navigate to the directory in which to save the entity.
   c. Click **OK**.

4. Click **Finish** to create the NE Connection Handler.

Related Topics

Working with Java NE Connection Handlers

Modeling Entities

Generating a Telnet NE Connection Handler Implementation

You need to generate a Telnet Network Element Connection Handler implementation if you want to extend a telnet connection to support the interface.

To generate a Telnet NE Connection Handler Implementation:

1. Create an NE Connection Handler with the NE Connection Handler Wizard.

   See "Working with Java NE Connection Handlers" for more information.

2. In the Studio Projects view, double-click the entity to open the NE Connection Handler editor.

3. In the editor, enter a description and select **Java Connection Handler** as the NE Connection Handler type.

4. Click **Add**.

   The **Vendor, Technology**, and **Software Load** fields are populated.

5. Click **New**.

   The Studio Activation Java Connection Handler Wizard appears.

6. Ensure that **Telnet** appears in the **Connection Type** field.

   Note: Ensure that a dot does not precede the package name. If a dot precedes the package name, remove it.

7. Click **Finish**.

   The code is generated ready for implementation.

   Note: The code is generated once but not synchronized (that is, it will not be rewritten and the developer owns the generated class).

Related Topics

Working with Java NE Connection Handlers

Modeling Entities

NE Connection Handler Editor Editor Tab

Generating a Custom NE Connection Handler Implementation

Generate a custom NE Connection Handler implementation if you want to extend the base NE connection class of a connection other than telnet.

To generate a custom NE Connection Handler implementation:

1. Create an NE Connection Handler with the NE Connection Handler Wizard.

   See "Working with Java NE Connection Handlers" for more information.

2. In the Studio Projects view, double-click the entity to open the NE Connection Handler editor.
3. In the editor, enter a description and select **Java Connection Handler** as the NE Connection Handler type.
4. Click **Add**.

   The **Vendor**, **Technology**, and **Software Load** fields are populated.
5. Click **New**.

   The Studio Activation Java Connection Handler Wizard appears.
6. Select **Custom** in the **Connection Type** field.

   Note: Ensure that a dot does not precede the package name. If a dot precedes the package name, remove it.

7. Click **Finish**.

   The code is generated ready for implementation.

   Note: The code is generated once but not synchronized (that is, it will not be rewritten and the developer owns the generated class).

Related Topics

Working with Java NE Connection Handlers

Generating a Telnet NE Connection Handler Implementation

Modeling Entities

NE Connection Handler Editor Editor Tab

Working with User-Defined Exit Types

User-defined exit types are values that describe the manner in which ASAP atomic actions complete. You can create your own values to describe exit scenarios specific to your own cartridge solutions.

When implementing user-defined exit types, see the following topics:

- About User-Defined Exit Types
- Creating New User-Defined Exit Types
- Configuring User-Defined Exit Types
- Extending User-Defined Exit Types
- Examples: User-Defined Exit Types

About User-Defined Exit Types

User defined exit types may be configured based on the base exit types explained in the table in "Extending User-Defined Exit Types" and other attributes like the following:

- Service action

- Action action
- Vendor
- Technology
- Software load
- Pattern

Related Topics

Working with User-Defined Exit Types

Modeling Entities

Creating New User-Defined Exit Types

Use the User-Defined Exit Type Wizard to create a user-defined exit type.

To create a new user-defined exit type:

1. Select **Studio**, then select **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Activation**, then select **User Defined Exit Type**.
3. Select the project for this element and enter a name for the entity.
4. (Optional) Select a location for the entity.

   By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

   a. Click the **Folder** field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.
5. Click **Finish** to create the user-defined exit type.

Related Topics

Working with User-Defined Exit Types

Modeling Entities

Configuring User-Defined Exit Types

You can configure user-defined exit types using the User Defined Exit Type editor.

To configure a user-defined exit type:

1. In the Studio Projects view, double-click a User-Defined Exit Type entity to open the User Defined Exit Type editor.
2. In the User Defined Exit Types area, click **Add**.

   This enables the fields in the User Defined Exit Types Detail area of the editor and populates those fields with default values.
3. In the **Pattern** field, enter a value.

   For example, enter SUCCESS, DENIED, RESOURCE BUSY, and so on.
4. Select the corresponding base exit type.
5. Enter the User Defined Exit Type for this pattern.

For example, you might enter **AA1_SUCCESS**.

6. Select **File**, then **Save**.

> **Note:** Use the **Service Action** and **Atomic Action** fields when creating Service Cartridges.

Related Topics

Working with User-Defined Exit Types

Modeling Entities

Working with Custom Action Processors

Working with Java NE Connection Handlers

Extending User-Defined Exit Types

Extending User-Defined Exit Types

In ASAP, cartridge error messages are categorized and generally assigned one of the following base exit types:

Atomic Action Exit Type	Description
SUCCEED	The atomic action executed successfully. The NEP successfully completes the atomic action command and the SARM provisions the next atomic action on the work order.
FAIL	Hard error. The atomic action failed. ASAP fails the entire work order and notifies the SRP of the WO Failure Event. If rollback is configured on the work order, the SARM rolls back any previously completed atomic actions.
RETRY	The atomic action will be retried later. While the atomic action is waiting to be retried, the connection to the network element can be used to provision other atomic actions destined to that network element (during Maintenance Mode, however, no other atomic actions are transmitted to the network element). If the atomic action command does not complete after the final retry, it is failed. The number and frequency of retry attempts is governed by work order properties. If these properties are not defined on the work order, the retry attempts is governed by **ASAP.cfg** configuration properties.
RETRY_DIS	A base exit type. The atomic action is not failed, but will be retired at a later time. ASAP disables the port that was originally used for the connection and forwards any future atomic action commands to other connections associated with the network element. This base exit type should be used when a connection is found to be in abnormal state and must be manually reset.
SOFT_FAIL	The atomic action has encountered a minor error, but the provisioning of the order completed successfully. For example, this exit type may occur when attempting to add a feature to a line that already has that option.
MAINTENANCE	The atomic action command failed because the network element is currently unavailable to receive provisioning requests. On receiving this atomic action response, ASAP moves the atomic action from the In Progress Atomic Action queue, and then marks the status of the network element as Maintenance.

Atomic Action Exit Type	Description
DELAYED_FAIL	The atomic action failed during provisioning. The SARM skips any subsequent atomic actions associated with the service action and continues provisioning at the next service action. It then fails the order after all subsequent service actions have finished processing. Rollback and delayed failure are incompatible because the intent of this base type is to continue to provision subsequent service actions; rollback would reverse successfully provisioned service actions. Delayed failure should be used only when there are no dependencies on subsequent service actions. If dependencies exist, the subsequent provisioning actions will fail.
STOP	The atomic action has stopped processing. While the work order is in a stopped state, ASAP only accepts requests to resume or cancel this work order.

For specific solutions, designers working with delivered cartridges will likely be required to customize the user-defined exit types provided in the cartridges for the following reasons:

- Unusual connection behavior in which connections may be dropped in the middle of activations and need to be reset.
- Various service actions and atomic actions must influence the base exit types that are assigned.
- Network elements have specific preassigned static maintenance windows in which they are not available to receive activation commands.

Related Topics

Configuring User-Defined Exit Types

Working with User-Defined Exit Types

Examples: User-Defined Exit Types

When working with user-defined exit types, see the following examples:

- Example: Unstable Network Element Connections
- Example: Configuration of Context Sensitive Exit Types
- Example: Exit Type Rationalization

Example: Unstable Network Element Connections

Problem: On an Ericsson network element during activation (after a successful connection and login to the network element has already occurred) the login to the network element is randomly terminated. As an atomic action may be in progress against the network element at the time the connection was dropped it must be placed back in the queue for later activation and the connection must be re-established.

Solution: Configure a user- defined exit type with the RETRY_DIS base type that is triggered when the login prompt is detected during normal activation. This will allow for the atomic action to be retried at a later time after instructing ASAP to disable the current connection. If there is only one connection to the network element then ASAP will eventually re-enable the connection and re-login.

Related Topics

Examples: User-Defined Exit Types

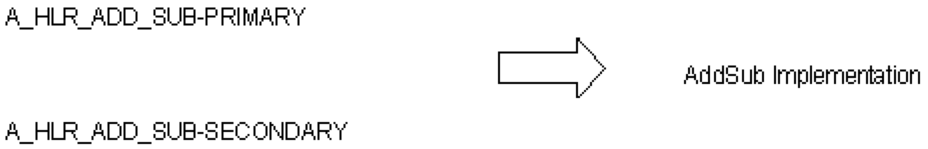
Working with User-Defined Exit Types

Modeling Entities

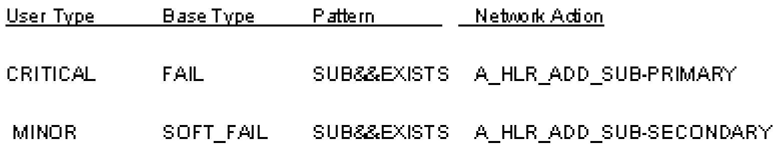
Example: Configuration of Context Sensitive Exit Types

Problem: The customer has a network in which each HLR (referred to as a primary HLR) has a backup HLR (referred to as a secondary HLR). Services must be activated on both HLRs but if activations fail on primary HLRs the work order must be failed; if activations fail on secondary HLRs they must be soft failed.

Solution: Create different atomic actions that map to the same implementation. Configure two user-defined exit types that include the atomic action names in the configuration. Configure the base type for the primary atomic action with FAIL. Configure the base type for the secondary atomic action with SOFT_FAIL. The service model for this configuration is shown in the following diagram:



The user-defined exit type configuration is shown as follows:



User Type	Base Type	Pattern	Network Action
CRITICAL	FAIL	SUB&&EXISTS	A_HLR_ADD_SUB-PRIMARY
MINOR	SOFT_FAIL	SUB&&EXISTS	A_HLR_ADD_SUB-SECONDARY

In this example, whenever the response from the network element contains the strings `SUB` and `EXISTS` and the atomic action is A_HLR_ADD_SUB-PRIMARY, then a failure is triggered. Whenever the response from the network element contains the strings `SUB` and `EXISTS` and the atomic action is A_HLR_ADD_SUB-SECONDARY then a soft failure is triggered.

Related Topics

Examples: User-Defined Exit Types

Working with User-Defined Exit Types

Modeling Entities

Example: Exit Type Rationalization

Problem: There are too many exit type entries with similar attributes present in the configuration, resulting in potentially high maintenance costs.

Solution: Where possible, collapse multiple exit type rows. For example, collapsing rows that have identical attributes other than the software load may be possible when the network element responses remain the same across software loads. A prime example of when exit type rationalization should occur is when multiple delivered cartridges are employed in the solution for the same network element. Because the user-defined exit types in delivered cartridges always contain the vendor, technology, and software load attributes to ensure uniqueness, exit type rationalization is generally possible.

Related Topics

Examples: User-Defined Exit Types

Working with User-Defined Exit Types

Modeling Entities

Working with Custom Action Processors

Custom action processors cater to a specific purpose that is outside the scope of the cartridge action processors.

When working with custom action processors, see the following topics:

- About Custom Action Processors
- Creating Custom Action Processors

About Custom Action Processors

Solution designers can create custom action processors to perform actions outside of the scope of the action processor in the cartridge. For example, a custom action processor may be necessary when a certain piece of data is required by a subsequent atomic action but is not provided by the upstream system. Generally a custom action processor will produce but not consume data. It simply generates data for use in subsequent spawning logic, for use in the generation of an API/MML command, or to query the switch for a piece of data that is to be passed back to upstream systems. Custom action processors generally do not talk to the network element but do some special processing of data, such as running a special algorithm to encrypt a piece of data or performing special formatting of data that may not have been handled by the upstream system or by the cartridge action processor.

Related Topics

Working with Custom Action Processors

Modeling Entities

Creating Custom Action Processors

To create a custom action processor:

1. Create a new atomic action that will trigger the custom action processor.

   Include the parameters required for the atomic action.

2. Create a new action processor and link it to the new atomic action.

   Creating a new action processor will involve some type of coding (either a state table or Java processor with code generation method).

3. Link the new atomic action into your service model wherever it is needed.

Related Topics

Working with Custom Action Processors

Modeling Entities

Understanding the Java Processor Class

Understanding Java with Code Generation

Creating Model Elements

Working with Retry Properties

Retry properties are set to allow retry of commands at the NE instance level and atomic action level.

When working with retry properties, see the following topics:

- About Retry Properties
- Example 1: Configuring Retry Properties at the Network Element Instance Level
- Example 2: Configuring Retry Properties at the Atomic Action Level

About Retry Properties

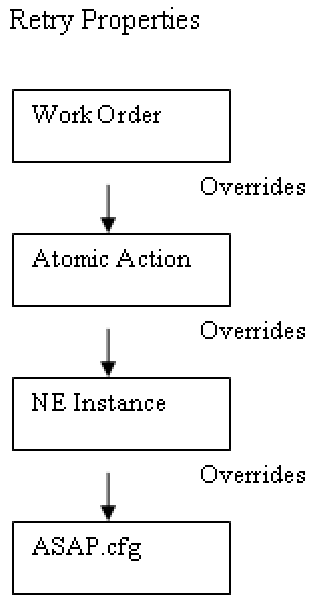
When a response from a network element is received that is mapped to a RETRY user-defined exit type (UDET), the Number of Retries value informs the system how many times the command should be retried before it is failed, and the Retry Time Interval is the amount of time between each retry.

As different network elements often have different retry requirements, it is necessary to provide a flexible retry mechanisms that enables retry properties to be specified at the network element instance level and at the atomic action level (this is in addition to the ability to configure a single set of system-wide retry properties, which apply to all atomic actions and all network elements that trigger a retry).

Flexible retry configuration in ASAP enables specification of retry properties in the following locations:

- **ASAP.cfg:** This configuration file contains values for the Number of Retries and Retry Interval which will be used whenever a retry occurs, on any network element or atomic action, if no other values are configured elsewhere.
- **Work Order:** If the Number of Retries and Retry Time Interval are specified on a work order, the values will override those defined elsewhere in the system (including the **ASAP.cfg** file, atomic action level, or network element instance level).
- **Atomic Action:** If you specify the Number of Retries and Retry Interval at the atomic action level, if a retry is encountered on any of the action processors mapped to that atomic action, the values you specify will be used. These values will override those defined at the network element instance level and at the **ASAP.cfg** level.
- **Network Element Instance:** If you specify the Number of Retries and Retry Interval on the Network Element editor, any command triggering a retry against this network element instance will use the retry values you specify. These values will override those defined at the **ASAP.cfg** level.
- **NE Template:** If you specify the Number of Retries and Retry Interval on the NE Template editor, any network element created from the template will inherit the retry values you specify.

- **Dynamic NE Template:** If you specify the Number of Retries and Retry Interval on the Dynamic NE Template editor, any network element instances dynamically created using the template will inherit the retry values you specify. These values will override those defined at the **ASAP.cfg** level.



Related Topics

Working with Retry Properties

Example 1: Configuring Retry Properties at the Network Element Instance Level

Example 2: Configuring Retry Properties at the Atomic Action Level

Understanding the Java Processor Class

Understanding Java with Code Generation

Understanding Java Libraries in Design Studio

Working with User-Defined Exit Types

Example 1: Configuring Retry Properties at the Network Element Instance Level

A specific vendor's network element often responds with a FUNCTION BUSY message, meaning that it cannot presently process commands and that the command should be retried at a later time (there is not necessarily any problem with the command itself, but the load on the network element is too large at this particular moment). Best practices dictate that a command will eventually succeed if tried 3 times with a 10 second interval between tries. To ensure that the command is properly retried, the service modeler should configure the retry properties at the network element instance level (see the following procedure). The work order will fail only if the configured Number of Retries is exceeded.

To configure retry properties at the network element instance level:

1. In the User Defined Exit Type editor, update the user-defined exit type configuration entry that corresponds to the FUNCTION BUSY response to specify an exit type of RETRY when this response message is encountered.

2. Modify the retry properties for any existing network element instances of that type.

   To do this, update the retry values in the Network Element editor for each network element instance as follows:

   - In the **Number of Retries** field, enter **3**.
   - In the **Retry Interval** field, enter **10**. (seconds)

3. Modify the retry properties for any existing Dynamic NE Template used for network element instances of that type.

   To do this, update the retry values in the Dynamic NE Template editor as follows:

   - In the **Number of Retries** field, enter **3**.
   - In the **Retry Interval** field, enter **10**. (seconds)

4. Ensure that all network element templates, network elements, and dynamic network element templates that were changed have been saved.

   After saving, you can deploy the configuration to an ASAP environment for testing.

Related Topics

Working with Retry Properties

Modeling Entities

Example 2: Configuring Retry Properties at the Atomic Action Level

When trying to change the LEN on a specific vendor's network element, the network element responds with an INVALID STATE error message if the customer line is in use. In this scenario, best practices dictate that ASAP retry the atomic action 10 times with an interval of 300 seconds between each attempt before a failure is be generated. The following example demonstrates how the service modeler configures the retry properties at the atomic action level to meet this criteria.

1. In the User Defined Exit Type editor, update the user-defined exit type configuration entry that corresponds to the INVALID STATE response to specify an exit type of RETRY when this response message is encountered.

2. When examining this network element's retry requirement, there are two options that would support the requirement:

   a. Modify the retry properties for the NE template (so that the configuration is carried over to any new network element instances that are created), for each network element instance of that type, and for each Dynamic NE template of that type.

   b. Modify the retry properties for the specific service action (change LEN). In this example, assuming the change LEN atomic action is specific to the vendor equipment in question (either a common atomic action mapping to only one vendor and technology, or a vendor and technology-specific atomic action mapping to a single action processor), and assuming the retry behavior specified for this requirement is unique to the atomic action (change LEN), then simply update the retry properties for the atomic action.

Note: Option a) requires multiple updates (to the NE Template, each network element instance, and each Dynamic NE Template). Option b) requires a single update.

3. Modify the retry properties for the change LEN atomic action.

   Update the retry values in the Atomic Action editor as follows:

   a. In the **Number of Retries** field, enter **10**.

   b. In the **Retry Interval** field, enter **300**. (seconds)

   Note: To update the retry value in an editor field, activate the field by selecting the corresponding check box. Retry values have no digit limit but must be positive integers. Retry values can be 0 if overriding the **ASAP.cfg** configured retry values is required.

4. Save changes to atomic actions.

   You can now deploy the configuration deployed to an ASAP environment for testing.

Related Topics

Working with Retry Properties

Modeling Entities

Configuring Network Element Instance Throughput Control

Configuring User-Defined Exit Types

Working with Network Elements

Working with Network Element Instance Throughput Control

Throughput control mechanism controls the number of transactions per unit of time. This mechanism ensures that networks elements are not overloaded.

When working with network element instance throughput control, see the following topics:

- About Network Element Instance Throughput Control
- Configuring Network Element Instance Throughput Control

About Network Element Instance Throughput Control

To prevent certain types of network elements from becoming overloaded, it may be necessary to control the volume of transactions that are being sent from ASAP. A central throughput control mechanism enables you to configure a specific throughput per unit of time for network element instances, which ensures that no more than a specific number of transactions are sent to the network element per unit of time.

Related Topics

Configuring Network Element Instance Throughput Control

Modeling Entities

Configuring Network Element Instance Throughput Control

Consider the following scenario:

It has been discovered that the throughput limitations of a specific network element (that responds to ASAP asynchronously) require that no more than 20 transactions per second can be sent to the network element. Otherwise, some response messages are not generated and are therefore never received by ASAP. To prevent overloading and ensure the network element generates all required response messages, the service modeler configures throughput controls for this network element instance as described below.

To configure the throughput control for a network element instance:

1. In the NE Template editor, modify the throughput properties used to create new network element instances.

   When modifying the properties used to create new network element instances, you ensure that any future network element instances use the appropriate throughput properties. To do this, update the throughput values in the NE Template editor as follows:

   a. In the **Throughput** field, enter **20** as the number of transactions.

      Valid **Throughput** field values range from 1 - 9999.

   b. In the **Transactions Per** field, enter **Seconds** as the unit of time.

2. In the Network Element editor, modify the throughput properties for any existing network element instances of that type.

   Update the throughput values as follows:

   a. In the **Throughput** field, enter **20** as the number of transactions.

      Valid **Throughput** field values range from 1 - 9999.

   b. In the **Transactions Per** field, enter **Seconds** as the unit of time.

3. In the Dynamic NE Template editor, modify the throughput properties for any existing Dynamic NE Template used for network element instances of that type.

   Update the throughput values as follows:

   a. In the **Throughput** field, enter **20** as the number of transactions.

      Valid **Throughput** field values range from 1 - 9999.

   b. In the **Transactions Per** field, enter **Seconds** as the unit of time.

4. Save all modified network element templates, network elements, and dynamic network element templates.

   You can now deploy the configuration to an ASAP environment for testing.

Related Topics

Working with Network Element Instance Throughput Control

Working with Retry Properties

Modeling Entities

Working with Network Elements

Packaging and Deploying Activation Cartridges

NE Connection Handler Editor

Use the NE Connection Handler editor to add or remove NE Connection Handlers. You can create a connection Handler class that manages the connection with an network element based on the communication parameters in an NE Template. In the Studio Projects view, double-click an NE Connection Handler entity to open the NE Connection Handler editor. You can create the editor using NE Connection Handler Wizard.

When working with the NE Connection Handler editor, see the following topics:

- NE Connection Handler Editor Editor Tab
- NE Connection Handler Editor Blueprint Tab

NE Connection Handler Editor Editor Tab

Use the **Editor** tab for defining the parameters for an NE Connection Handler. In the Connection Handler Detail area, you can define the class for the NE Connection Handler entity.

Field	Use
Description	Specify a description for the NE Connection Handler editor.
Connection Handler Type	Select any one of the following Connection Handler types from the list: ■ Java Connection Handler A Java class that contains the connection logic related to network element connection. The Java Connection Handler is recommended for use. ■ Login State Table Connection Handler A programming type related to state table. ■ Logoff State Table Connection Handler A programming type related to state table. ■ Generic Connection Handler A programming type which is neither related to Java class nor state table.
Vendor, Technology, and **Software Load**	In the Connection Handlers area, on clicking **Add**, displays the vendor, technology, and software load for the NE Connection Handler entity in the cartridge project.
Class	In the Connection Handlers area, displays the class you have selected or created in the Connection Handler Details area.
Add	Click to add a Connection Handler. As there are different ways of communicating like SOAP, Telnet, and so on, you can add or select a Connection Handler implementation for the Connection Handler.
Remove	Click to clear a Connection Handler from the editor.
Open	Click to open the code of the Connection Handler class. This button is enabled only when the code is available.

Field	Use
New	Click to create a connection class. The connection class contains the skeleton of the connection logic based on the connection type selected on the Studio Activation Java Connection Handler Wizard. **Note: New** is grayed out for Login State Table Connection Handler and Logoff State Table Connection Handler types.
Select	Click to select an existing class.

Related Topics

NE Connection Handler Editor

NE Connection Handler Editor Blueprint Tab

Working with Java NE Connection Handlers

NE Connection Handler Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the NE Connection Handler entity. This tab is read-only.

Related Topics

NE Connection Handler Editor

NE Connection Handler Editor Editor Tab

Working with Java NE Connection Handlers

User Defined Exit Type Editor

Use the User Defined Exit Type editor to add or remove user-defined exit types. In the Studio Projects view, double-click a user-defined entity to open the User Defined Exit Type editor. You can create the editor using the User Defined Exit Type Wizard.

When working with the User Defined Exit Type editor, see the following topics:

- Working with User-Defined Exit Types
- User Defined Exit Type Editor Editor Tab
- User Defined Exit Type Editor Blueprint Tab

User Defined Exit Type Editor Editor Tab

Use the **Editor** tab for configuring the values of the user-defined exit types. In the User Defined Exit Type Detail area, you can configure the values for a specific user-defined exit type.

User Defined Exit Types

Field	Use
Description	Specify a description for the User Defined Exit Type editor.
Vendor, **Technology**, and **Software Load**	Displays the vendor, technology, and software load for the user-defined exit type entity in the cartridge project.

Field	Use
User Exit Type and **Base Exit Type**	Displays the values, of the **User Defined Exit Type** and **Base Exit Type** fields respectively, you specified in the User Defined Exit Type Detail area. **Note:** In the User Defined Exit Types area, initially the fields display the default values.
Add	Click to add a new user-defined exit type. The default values appear in the fields.
Remove	Click to clear a user-defined exit type from the editor.

User Defined Exit Type Detail

Field	Use
Pattern	Specify a regular expression to perform a pattern search on the responses from network elements. The regular expression will allow cartridge users to associate a series of responses to a user-defined exit type. For example, a regular expression "6." can identify a pattern where any response with the character "6" followed by any number of characters will translate to the user-defined exit type related to failure.
User Defined Exit Type	Specify a value that maps atomic action exit codes to one of the predefined base exit types. The cartridges map the return codes and status values from a network element to a user-defined exit type.
Base Exit Type	Select a value that determines the product behavior like success or failure.
Select	Click to select a service action or an atomic action. **Note:** You must select a service action or an atomic action while creating Service cartridges.
Open	Click to open the selected service action or atomic action.
Clear	Click to clear from the field the selected service action or atomic action.
Description	Specify a description for the specific user-defined exit type.

Related Topics

User Defined Exit Type Editor

Working with User-Defined Exit Types

User Defined Exit Type Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the user-defined exit type entity. This tab is read-only.

Related Topics

User Defined Exit Type Editor

Working with User-Defined Exit Types

7 Working with Network Elements

To model network element (NE) models, you create and configure three elements: the NE template, the network element and the Dynamic Network template. These elements are related as they involve connections to equipment.

If you want to route atomic actions in a work order using Directory Number (DN), you must configure the DN routing details.

When working with network elements models, see the following topics:

- About Network Elements
- Configuring Network Element Models
- Working with Directory Number Routing
- DN Routing Map Editor
- Dynamic NE Template Editor
- NE Template Editor

About Network Elements

You can create and configure the following network elements:

NE Template

NE templates are similar to network elements: they exist in projects, they have vendor, technology, and software loads, and so forth. However, you can set up network element templates so that they can be copied later to create one or many specific network elements.

Network Element

Network elements represent the information needed to connect to one specific piece of equipment on the network. Separate connection pools allow more than one connection to a piece of equipment, and for each connection you can set up a different connection pool. Pools have parameters that are used by connection handlers.

Dynamic NE Template

Dynamic NE templates represent an additional method for specifying how to connect to a piece of equipment on the network. However, in contrast to a network element (where one network element is created for each piece of equipment, and a template can be used to assist with creation of multiple network elements), one dynamic NE template is created for a vendor, technology, and software load (and not for a piece of equipment). When an order comes into the system, half of the connection information is in the dynamic NE template, the other half is in the order. Therefore, the connection pool is created at runtime and is discarded when no longer needed. Consequently, you are not required to set up network elements individually. Instead, you set up only the profile and then enter the connection information on the order.

Related Topics

Working with Network Elements

Configuring Network Element Models

When configuring network element models:

1. Create an NE template that you can copy later to create a specific network element. See "Creating NE Templates".
2. Create a network element based on the NE template. See "Creating Network Elements based on an NE Template".
3. (Optional) If you are working with large network, create a dynamic NE template. See "Creating Dynamic NE Templates".

Related Topics

Working with Network Elements

Creating NE Templates

To create NE templates:

1. Select **Studio**, then **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Activation**, then select **NE Template**.

   The **NE Template Wizard** appears.
3. Select the project and enter a name for the entity.
4. Click **Finish**.

   The NE Template editor appears.
5. In the **Connections** tab, click **Add**.

   The Add Predefined Parameters dialog box appears.
6. Click **Yes** to set up a different connection pool for each required connection.

   You can add the predefined Oracle Communications ASAP communication parameters as global parameters.

   In the **All Communication Parameters** tab, depending on the network element type (Serial Port Dial-up, for example), all parameters for the type are listed by default and are displayed as Global connections. You can leave them as global connections to use the value for a parameter across all connections, or you can override parameters (a port setting, for example) for a connection.
7. To override a parameter, select it in the Connection Parameter Details list and click **Make Local**.

   The parameter will then have the connection name substituted for Global in the list. For each connection, the network element first uses local parameters if available. If none are available, the global parameters are used.

8. In the **Target Network Elements** tab, create a target network element by copying the name of the network element into the **Target NE Name** field.
9. Select **File**, then select **Save**.

   The network element is now ready for users to connect to it with their pools.

   > **Note:** There is a direct link between connection handlers and network elements. When ASAP determines what network element it needs to configure (the one it is routing to), it checks the vendor, technology, and software load of the network element and then searches for a connection handler with the same vendor, technology, and software load. The parameters it sets up go into the connection handler, which gets called to establish the connection to the pools. When the connection is established, the connection handler determines which atomic action it is processing and then checks the implementation map for a matching vendor, technology, and software load to determine which action processor to use.

10. In the **Connection Pool Name** field, enter a name (the name must be unique on the ASAP server).

Related Topics

Configuring Network Element Models

Working with Network Elements

NE Template Editor

Creating Network Elements based on an NE Template

To create a network element based on an NE template:

1. Select **Studio**, then select **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Activation**, then select **Network Element**.

   The Studio Model Entity wizard appears.
3. Select the project, enter a name for the entity, and select the NE template.
4. Click **Finish**.

   The Network Element editor appears. The pool and parameter descriptions are identical to those on the NE Template editor. See "NE Template Editor" for more information on using the Network Element editor.
5. In the **Connection Pool Name** field, enter a unique name (pool values must be unique across the ASAP server).
6. Navigate to the **Target Network Elements** tab.
7. Create a target network element by copying the name of the network element into the **Target NE Name** field.
8. Click **Save** to save the network element.

   The network element is now ready for users to connect to it with their pools.

Note: There is a direct link between connection handlers and network elements. When ASAP determines what network element it needs to configure (the one it is routing to), it checks the vendor, technology, and software load of the network element and then searches for a connection handler with the same vendor, technology, and software load. The parameters it sets up go into the connection handler, which gets called to establish the connection to the pools. Once the connection is established, the connection handler determines which atomic action it is processing and then checks the implementation map for a matching vendor, technology, and software load to determine which action processor to use.

Related Topics

Configuring Network Element Models

Working with Network Elements

Creating Dynamic NE Templates

For a large network it is easier to use dynamic NE templates because dynamic templates do not require that you keep an inventory of all equipment. Rather, you need only an inventory system to acquire the routing connection data.

To create a dynamic NE template:

1. Select **Studio**, then select **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Activation**, then select **Dynamic NE Template**.

   The Studio Model Entity wizard appears.
3. Select the project and enter a name for the entity.
4. Click **Finish**.

   The Dynamic NE Template editor appears. Only the global parameters are displayed in the **NE Instance Properties** list, as all connections have been created during runtime and a remote network element is not required. For example, if socket-based, you would add socket-based parameters only; any order that comes in with the same vendor, technology, and software load would get the global parameters from here and get the connection parameters from the order (a password or IP address, for example). A syntax on the order is set up for global parameters:

   ```
   COMM_PARAM.NE_ID.value
   ```
5. In the NE Instance Properties area, the following fields are listed:
   - **Drop Time Out (minutes)**: This field specifies the time in which a connection will drop in absence of activity, such as generation of new connection requests. You can change the time as per your requirement.
   - **Maximum connections**: This field specifies the number of connections allowed by ASAP in a connection pool. You can change the number of connections as per your requirement.
   - **Spawn Threshold (AAs)**: If there is a request for a new connection and the number of connections in a connection pool has reached a threshold, a new connection pool is created. This field specifies the threshold value after which a new connection pool is created.

- **Kill Threshold (AAs)**: If the number of connections in a connection pool goes below than the number of connections specified in the kill threshold, the connection pool is dropped.
- **Retry Count**: If a connection fails between ASAP and the network element, ASAP tries to establish the connection again. This field specifies the number of attempts that ASAP can do to reestablish the connection.
- **Retry Interval**: This field specifies the time interval between each retry attempt by ASAP. You can change the time interval as per your requirement.
- **Throughput**: This field specifies the time for network element instance to ensure that no more than specific number of transactions are sent to the network element per unit of time.

6. Click **Add** to add more global parameters.
7. Save the dynamic NE template.

Related Topics

Configuring Network Element Models

Working with Network Elements

Working with Directory Number Routing

Atomic actions can be routed to a network element using DN and this mechanism of routing is termed as DN routing.

When working with DN routing, see the following topics:

- About Directory Number Routing
- Creating Directory Number Routing
- Configuring Directory Number Routing

About Directory Number Routing

Directory Number is a 10 digit telephone number which is divided into three parts:

- NPA: The first three digits of a telephone number known as Numbering Plan Area (NPA) code.
- NXX: The second three digits of a telephone number known as Central Office code is also known as prefix.
- Line: The last four digits of a telephone number.

Using Oracle Communications Design Studio, DN can be configured on atomic actions in a cartridge. See *ASAP System Administrator's Guide*.

Related Topics

Creating Directory Number Routing

Configuring Directory Number Routing

DN Routing Map Editor

Working with Directory Number Routing

Creating Directory Number Routing

You can use the DN Routing Map Wizard to create a DN routing map entity.

To create DN routing:

1. Select **Studio**, then select **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Activation**, then select **DN Routing Map**.

   The DN Routing Map Wizard appears.
3. Select a project for this element and enter a name for the entity.
4. (Optional) Select a location for the entity.

   By default, Design Studio saves the entity to your default workspace location. You can deselect the **Use recommended name and location** check box and enter a folder name in the **Folder** field or select a location different from the system-provided default.

   To select a different location:

   a. Deselect the **Use recommended name and location** check box.

   b. Click the **Folder** field **Browse** button.

   c. Navigate to the directory in which to save the entity.

   d. Click **OK**.
5. Click **Finish**.

   In the Studio Projects view, in the project folder, the new DN routing map entity appears.

Related Topics

DN Routing Map Editor

Working with Directory Number Routing

Configuring Directory Number Routing

You can configure a DN routing map entity using the DN Routing Map editor.

To configure DN routing:

1. In the Studio Projects view, double-click a DN routing map entity.

   The DN Routing Map editor appears.
2. In the **Description** field, specify the description for the editor.
3. In the DN Routing Details area, click **Add**.

   A new row with default values appears.
4. Click the row to edit the DN routing details.

   This enables the fields in the Routing Map Details area for this specific row.
5. In the Routing Map Details area, for the specific row, specify the values for the **NPA, NXX, Min Line,** and **Max Line** fields.

   Note: The range for **NPA** and **NXX** is from 100 to 999 and for **Min Line** and **Max Line** is from 1000 to 9999.

6. (Optional) Click the **...** button and select an atomic action that you want to route using DN.
7. Click the **...** button and select a network element to which you want to route the atomic action.
8. Select **File**, then select **Save**.

Related Topics

DN Routing Map Editor

Working with Directory Number Routing

DN Routing Map Editor

Use the DN Routing Map editor to add or remove routing details to route atomic actions using DN. In the Studio Projects view, double-click a DN routing map entity to open the DN Routing Map editor.

When working with the DN Routing Map editor, see the following topics:

- DN Routing Map Editor Editor Tab
- DN Routing Map Editor Blueprint Tab

DN Routing Map Editor Editor Tab

Use the **Editor** tab for defining the parameters for routing an atomic action using DN. In the Routing Map Details area, you can configure the values for a DN routing map entity.

Field	Use
Description	Specify a description for the DN Routing Map editor.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load for the DN routing map entity in the cartridge project.
NPA, NXX, Min Line, Max Line, Atomic Action, and **Network Element ID**	Displays the values configured in the Routing Map Details area.
Add	Click to add DN routing details for an atomic action.
Remove	Click to clear DN routing details for an atomic action.

Routing Map Details

Field	Use
NPA	Specify a Numbering Plan Area (NPA) code. The allowed range is from 100 to 999.
NXX	Specify a Central Office code also known as prefix. The allowed range is from 100 to 999.
Min Line	Specify the lowest line number in the range. The allowed range is from 1000 to 9999.
Max Line	Specify the highest line number in the range. The allowed range is from 1000 to 9999.

Field	Use
Atomic Action	This field is optional. Select an atomic action which you want to route using DN. If no atomic action is selected, all atomic actions in the work order will be routed using DN.
Network Element ID	Select a network element to which you want to route the atomic action.

Related Topics

DN Routing Map Editor

Working with Directory Number Routing

DN Routing Map Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the DN routing map entity. This tab is read-only.

Related Topics

DN Routing Map Editor

Working with Directory Number Routing

Dynamic NE Template Editor

Use the Dynamic NE Template editor to define a dynamic NE template entity. The entity uses dynamic routing. Dynamic routing is the ability to route orders based on network and communication data provided as order parameters, rather than using preconfigured static, locally maintained data.

In the Studio Projects view, double-click a dynamic NE template entity to open the Dynamic NE Template editor.

When working with the Dynamic NE Template editor, see the following topics:

- Dynamic NE Template Editor Editor Tab
- Dynamic NE Template Editor Blueprint Tab

Dynamic NE Template Editor Editor Tab

Use the **Editor** tab for defining the values required for connecting to a network element.

Field	Use
Description	Specify a description for the Dynamic NE Template editor.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load for the dynamic NE template entity in the cartridge project.
Drop Time Out (minutes)	Select the time in which a connection will drop in the absence of an activity.
Maximum Connections	Select the maximum number of concurrent connections allowed to the host network element.
Retry Count	Select the **Retry Count** check box to enable the **Retry Count** box. From the **Retry Count** box, select the number of attempts that ASAP can do to reestablish the connection to the network element.

Field	Use
Retry Interval	Select the **Retry Interval** check box to enable the **Retry Interval** box. From the **Retry Interval** box, select the time interval between each retry attempt by ASAP.
Spawn Threshold (AAs)	Select the number of atomic action requests destined to the network element. The number must exceed the spawn threshold value to open a new connection to the network element. As the number of atomic actions waiting to be routed to the network element continues to exceed spawn threshold, ASAP continues to establish new connections to the network element till maximum connections is reached. The spawn threshold value is measured in terms of atomic actions.
Kill Threshold (AAs)	Select the number of atomic action requests that ASAP must have for a specific network element for ASAP to disconnect one or more connections to the network element. **Note:** The kill threshold value should be lower than the spawn threshold value.
Throughput	Select the **Throughput** check box to enable the **Throughput** box. From the **Throughput** box, select the minimum amount of time a transaction takes to complete for each network element instance.
Transactions Per	Select the unit of time required for transactions. **Note:** If you want to select a value, select the **Throughput** check box.
Line Type	Select a connection type that the network element will support. **Note:** The value passed through the work order takes precedence.
Parameter Label, Value, and **Description**	In the Connection Parameter Details area, specify the values for the connection details. These values are global parameters. For example, user ID and password of a Telnet session. **Note:** Ensure that the same label names and values are used in the NE connection class.
Add	Click to add global parameters.
Remove	Click to clear the global parameters from the editor.

Related Topics

Dynamic NE Template Editor

Working with Network Elements

Dynamic NE Template Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the dynamic NE template entity. This tab is read-only.

Related Topics

Dynamic NE Template Editor

Working with Network Elements

NE Template Editor

Use the NE Template editor to define an NE template entity. You can create a connection pool and add communication parameters used to manage the connections to a network element.

You can create the NE Template editor using the NE Template Wizard. In the Studio Projects view, double-click a NE Template entity to open the editor.

When working with the NE Template editor, see the following topics:

- NE Template Editor Editor Tab
- NE Template Editor Blueprint Tab

NE Template Editor Editor Tab

Use the **Editor** tab to define configuration values used to manage the connections to a network element.

Field	Use
Description	Specify a description for the NE Template editor.
Vendor, Technology, and **Software Load**	Displays the vendor, technology, and software load for the NE template entity in the cartridge project.

When working with the **Editor** tab, see the following topics:

- General Tab
- Connections Tab
- All Communication Parameters Tab
- Target Network Elements Tab

Related Topics

NE Template Editor

Working with Network Elements

General Tab

Use the **General** tab to define values for a connection pool.

Field	Use
Connection Pool Name	Specify a unique eight-lettered connection pool name. Any connection to the network element is borrowed from this connection pool.
Spawn Threshold (AAs)	Select the maximum number of atomic action requests that can be queued to be sent to the network element in ASAP at a given time. As the number of atomic actions waiting to be routed to the network element continues to exceed spawn threshold, ASAP continues to establish new connections to the network element till maximum connections is reached. See *ASAP Developer's Guide*. The spawn threshold value is measured in terms of atomic actions. The spawn threshold value must be above the kill threshold value, if multiple connections are required to a specific network element.

Field	Use
Retry Count	Select the **Retry Count** check box to enable the **Retry Count** box. From the **Retry Count** box, select the number of attempts that ASAP perform to reestablish the connection to the network element.
Retry Interval	Select the **Retry Interval** check box to enable the **Retry Interval** box. From the **Retry Interval** box, select the time interval between each retry attempt by ASAP.
Protocol	Select a connection type that the network element will support.
Maximum Connections	Select the maximum number of concurrent connections allowed to the host network element.
Throughput	Select the **Throughput** check box to enable the **Throughput** box. From the **Throughput** box, select the minimum amount of time a transaction takes to complete for each network element instance.
Transactions Per	Select the unit of time required for transactions. **Note:** If you want to select a value, select the **Throughput** check box.
Drop Time Out (minutes)	Select the time in which a connection will drop in the absence of an activity.
Kill Threshold (AAs)	Select the maximum number of atomic action requests that must be queued in ASAP at a given time for a specific network element, for ASAP to disconnect any auxiliary connection to the network element. See *ASAP Developer's Guide*. **Note:** The kill threshold value should be lower than the spawn threshold value.

Related Topics

Connections Tab

All Communication Parameters Tab

Target Network Elements Tab

Connections Tab

Use the **Connections** tab to add connections to the connection pool, add predefined communication parameters, and if required override them with local parameters. The local parameters that appear in the Connection Parameter Details area are local to the connection selected in the NE Instance Properties area in the **Connections** tab. So, every connection in the connection pool can have parameters local to just one specific connection and can also have parameters common to all the connections in the connection pool termed as global parameters.

NE Instance Properties

Field	Use
Connection Name	Specify a name to the connection in the connection pool.
Protocol	Displays the protocol selected on the **General** tab.
ASAP System Setting	Select to display whether an ASAP environment is for development or production.
Add	Click to add a connection in the connection pool.

Field	Use
Remove	Click to clear a connection from the connection pool.
Copy	Click to create a copy of a specific connection but with a unique name.

The global and local communication parameters appear in the Connection Parameter Details area. The global parameters are common across all the connections in a pool and so are editable in the **All Communication Parameters** tab. See "All Communication Parameters Tab" for more information.

Connection Parameter Details

> **Note:** You can double-click and edit only the local communication parameters as they are local to a connection.

Field	Use
Parameter Label	Double-click to specify a parameter name.
Connection	Displays the connection name selected in the NE Instance Properties area.
Value	Double-click to specify a value.
Description	Double-click to specify a description.
Add Local	Click to add a new parameter with the connection name as displayed in the row highlighted in the NE Instance Properties area.
Remove Local	Click to clear the parameter from the editor.
Make Local Copy	Click to create a parameter which is a copy of the global communication parameter.

Related Topics

General Tab

All Communication Parameters Tab

Target Network Elements Tab

All Communication Parameters Tab

Use the **All Communication Parameters** tab to define the global communication parameters. The global communication parameters are common to all the connections in a connection pool.

The local communication parameters appear in this tab but are not editable. These are the same as can be seen in the **Connections** tab for a specific connection.

> **Note:** You can double-click and edit only the global communication parameters.

Field	Use
Parameter Label	Displays the name of the communication parameter. You can double-click to rename the parameter label name.
Connection	Displays the connection as global or local. If the parameter is local, the connection name of the connection to which the parameter belongs appears.
Value	Double-click to specify a value to the global communication parameter.
Description	Double-click to specify a description to the global communication parameter.
Add Global	Click to add a parameter as global.
Remove Global	Click to clear a parameter from the editor.

Related Topics

General Tab

Connections Tab

Target Network Elements Tab

Target Network Elements Tab

Use the **Target Network Elements** tab to define the target network elements to which you can route atomic actions.

Field	Use
Target Network Element, Atomic Action, and **Description**	Displays the values configured in the Target NE Details area.
Add	Click to add a new target network element.
Remove	Click to clear a target network element from the editor.
Target NE Name	Specify the name of a target network element to which you can route an atomic action.
Select	Click to select an atomic action to be routed through the target network element. Do not select a specific atomic action, if you want all the atomic actions to be routed to one network element.
Clear	Click to clear the atomic action for a specific target network element.
Description	Specify a description for the target network element.

Related Topics

General Tab

Connections Tab

All Communication Parameters Tab

NE Template Editor Blueprint Tab

Use the **Blueprint** tab to view the generated documentation for the NE Template entity. This tab is read-only.

Related Topics

NE Template Editor

Working with Network Elements

8

Packaging and Deploying Activation Cartridges

To deploy Activation cartridges to the Oracle Communications ASAP environment, do the following tasks:

1. Determine the elements to include in the deployable SAR file.

   See "Packaging Activation Cartridges" for more information.

2. Create a Studio Environment project.

   See "Creating Environment Projects" for more information.

3. Create an entity to define the run-time environment connection parameters.

   See "Creating Run-Time Environments" for more information.

4. Deploy and manage cartridges in the ASAP environment by adding and removing cartridges for deployment and deploying and undeploying cartridges.

   See "Deploying Cartridge Projects" for more information.

5. Map network element processors by creating and configuring NEP Map entities.

   See "Mapping Network Element Processors" for more information.

Packaging Activation Cartridges

Packaging enables you to control what elements will be included in the cartridge SAR file that you will deploy to the ASAP environment. The cartridge packaging instructions appear as a list of all types of elements that can be packaged inside the cartridge. By default, everything within the cartridge project will be included when the cartridge is built. If you want to specify specific entities to include for each type of element (service actions, atomic actions, and so forth), select the element type in the first column to display all entities of this type in the Entity column. See "Packaging Projects" for more information.

Related Topics

Understanding Java Libraries in Design Studio

Deploying Cartridges and Managing the ASAP Environment

Packaging and Deploying Activation Cartridges

Deploying Cartridges and Managing the ASAP Environment

After you create a Studio Environment project and a run-time environment, you can use the Studio Environment editor to configure the environment for a connection. See "Deploying Cartridge Projects" for more information.

Related Topics

Packaging and Deploying Activation Cartridges

Mapping Network Element Processors

To map network elements (NEs) to network element processors (NEPs), you create a new NEP Map entity and configure the entity using the NEP Map editor.

> **Note:** NEs are packaged with cartridges but are deployed and managed separately.

When mapping NEPs, see the following topics:

- Creating New NEP Map Entities
- Configuring NEP Maps

Creating New NEP Map Entities

You create new NEP Map entities using the NEP Map entity wizard.

To create a new NEP Map entity:

1. Create a Studio Environment project. See "Creating Environment Projects" for more information.
2. In the Studio Projects view, right-click and select **New**, select **Environment**, then select **NEP Map**.

   The NEP Map Wizard appears.
3. Select a Studio Environment project from the list.
4. Select an environment from the Studio Environment list.

   The Studio Environment list contains all of the `.sceEnvironment` objects in the workspace. By associating the NEP Map entity with the selected Studio environment, the entity can obtain from the environment the ASAP JMX server:port information and the `ENV_ID`, among other properties.
5. Specify an entity name.

   The system applies a default entity name prefix of `M_`.
6. (Optional) Select a location for the entity.

   By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

   a. Click the **Folder** field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.
7. Click **Finish** to create the entity in the specified location of the project.

   The system creates a new `.nepMap` object and opens the new entity in the NEP Map editor.

Related Topics

Mapping Network Element Processors

Packaging and Deploying Activation Cartridges

NEP Map Editor

Configuring NEP Maps

You configure NEP maps using the NEP Map editor, which displays a list of all network elements that you have added. After connecting to the environment, you can select a default NEP from the available list for the environment to which the elements will be sent.

When configuring NEP maps, you first connect to your ASAP environment to retrieve the NEP server names. After retrieving the server names, you can configure your NEP maps (in the Mapping Detail area of the NEP Map editor) while connected to the ASAP environment or by working offline in Design Studio.

To retrieve NEP server names:

1. Click **Connect** to connect to your ASAP environment using your WebLogic Server user name and password.

   > **Note:** To connect to your ASAP environment, you must define your connection settings. See "Studio Environment Editor Activation Connection Details Area" for more information. The NEP Map editor retrieves all NEP server names and adds them to the Default NEP list.

2. From the Default NEP list, select an NEP.
3. Click **Refresh** to retrieve new NEP server names from ASAP that you added while connected to the ASAP system.

   > **Note:** Disconnecting and reconnecting also refreshes the NEP list. After you retrieve the NEP list you can disconnect from ASAP and configure the NEP map in Design Studio. However, if you intend to deploy or undeploy network elements after configuring, do not disconnect from ASAP.

To map network elements:

1. In the Network Element Processor Map grid, select a network element.
2. In the Mapping Detail area, select a value from the **NEP Server** list.

   Your selection determines the value that displays in the NEP column in the Network Element Processor Map grid.
3. Select **File**, then select **Save**.
4. Select the element and click **Deploy**.

Note: You must define the `ENV_ID` and `ASAP_VERSION` properties for the ASAP JMX service in the Studio Environment editor. These properties are required for the NEP Map entity to generate JMX-required JNDI context strings. See "Studio Environment Editor Activation Connection Details Area" for more information.

You can also optionally enable the RESTART_ASAP_SERVERS variable in the Studio Environment editor. Enabling this variable automatically restarts the ASAP servers when you deploy a cartridge. See "Studio Environment Editor Cartridge Management Variables Tab" for more information.

5. Do any one of the following (if you have not enabled the RESTART_ASAP_SERVERS variable):
   - Restart ASAP.
   - Restart the NEP server to which the network element is mapped.
   - Execute the following command:

   ```
   asap_utils -option 113
   ```

Related Topics

Mapping Network Element Processors

Packaging and Deploying Activation Cartridges

NEP Map Editor

Studio Environment Editor Activation Connection Details Area

Use the Connection Details area, in the **Connection Information** tab on the Studio Environment editor, to provide details to connect to ASAP. See "Studio Environment Editor" for more information.

Activation Connection Details

Field	Use
Environment ID	Specify the ASAP environment ID, of an ASAP instance, specified during ASAP installation.
Activation Version	Select an ASAP version to which you will connect.
Server Address	Specify the IP address or the host name of the computer on which the Oracle WebLogic Server is installed. If ASAP is installed on the administration server, use the IP address or the host name of the administration server. If ASAP is installed on a managed server, use the IP address or the host name of the managed server.
Server Port Number	Specify the port number of the computer on which the WebLogic Server is installed. If ASAP is installed on the administration server, use the port of the administration server. If ASAP is installed on a managed server, use the port of the managed server.
JNDI Context	Displays the JNDI context that appears based on the selected Activation version. WebLogic Server uses JNDI context and the queues mentioned in this tab to communicate to ASAP.

Field	Use
Order Submission Queue	Displays the name of the queue in the WebLogic Server where the orders are submitted to ASAP.
Order Event Queue	Displays the name of the queue in the WebLogic Server where events from ASAP are received.
Order Response Queue	Displays the name of the queue in the WebLogic Server where a work order response is received.

Related Topics

Packaging and Deploying Activation Cartridges

Studio Environment Editor Cartridge Management Variables Tab

Studio Environment Editor Cartridge Management Variables Tab

Use the **RESTART_ASAP_SERVERS** variable, in the **Cartridge Management Variables** tab on the Studio Environment editor, to enable ASAP servers to automatically stop and restart after deploying a cartridge.

To enable the **RESTART_ASAP_SERVERS** variable:

1. From the Studio Environment Editor, select the **Cartridge Management Variables** tab.
2. Select the **RESTART_ASAP_SERVERS** variable from the Cartridge Management Variables list.
3. In the Environment Cartridge Management Variable Details area, click **Override**.
4. In the **Override** field, enter **true**.

Note: When this attribute is enabled, the ASAP servers restart after deploying an ASAP cartridge. This also means that you cannot deploy additional cartridges until the ASAP servers have restarted. Attempting to deploy an ASAP cartridge too quickly after deploying the initial cartridge will result in a deployment failure message.

Related Topics

Packaging and Deploying Activation Cartridges

NEP Map Editor

Use the NEP Map editor to map network elements to a NEP and deploy the network elements.

You can create the NEP Map editor using the NEP Map Wizard. In the Studio Projects view, double-click a NEP Map entity to open the editor.

Field	Use
Description	Specify a description for the NEP Map editor.
Default NEP	Select an NEP as default from the available list of NEPs in the ASAP environment.

Field	Use
Refresh	Click to retrieve a fresh list of NEP server names from the ASAP environment.
Connect	Click to connect to an ASAP environment using WebLogic Server user name and password.
Environment	Displays the name of the Studio environment selected on the NEP Map Wizard.
Add	Click to add an network element entity to the Network Element Processor Map grid.
Remove	Click to remove an network element entity from the Network Element Processor Map grid.
Deploy	Click to deploy a specific network element to the ASAP environment.
Deploy All	Click to deploy all the network elements on the NEP Map editor to the ASAP environment.
Undeploy	Click to undeploy a specific network element from the ASAP environment.
Undeploy All	Click to undeploy all the network elements from the ASAP environment.
Network Element and Cartridge	Displays the network element entity name and the cartridge name to which the network element belongs.
NEP Server	Select a NEP server for a specific network element on the Network Element Processor Map grid.

Related Topics

Mapping Network Element Processors

9

Testing ASAP Cartridges in Design Studio

An Activation test case (test work order) enables you to test, within Oracle Communications Design Studio, the activation order completion for a cartridge to verify that a work order is sent to the Oracle Communications ASAP environment. Consider the following when creating and running test cases:

- Each test case can have multiple service actions.
- It is possible to run multiple tests that are almost identical except for different parameters. Design Studio runs all of the cases in sequence.
- Test cases use OSSJ interface.

To test ASAP cartridges in Design Studio, you create an activation test case, then you run the test case.

> **Note:** Do not submit an activation test case over an SSL connection.

When working with activation test cases, see the following topics:

- Creating Activation Test Cases
- Running Activation Test Cases
- Activation Test Case Editor

Creating Activation Test Cases

To create an activation test case:

1. Select **Studio**, then select **Show Design Perspective**.
2. Select **Studio**, select **New**, select **Activation**, then select **Activation Test Case**.

   The Studio Model Entity wizard appears with the project name selected by default.
3. Enter a name for the activation test case.
4. (Optional) Select a location for the entity.

   By default, Design Studio saves the entity to your default workspace location. You can enter a folder name in the **Folder** field or select a location different from the system-provided default. To select a different location:

   a. Click the **Folder** field **Browse** button.

   b. Navigate to the directory in which to save the entity.

   c. Click **OK**.

5. Click **Finish**.

   In the Studio Projects view, the new Activation Test Case entity appears under your project folder.

6. Double-click the Activation Test Case entity to display the Activation Test Case editor.

7. Click the **Service Actions** tab.

8. In the Test Execution area, do the following:

   - In the **Test Order Primary Key Pattern** field, do the following:

     – To identify the test case with an auto-generated integer, enter **0**.

       See "About Test Case Configuration" for more information.

     – To identify the test case with the name of the test appended with a random, auto-generated, integer, enter **1**.

   - In the **Timeout (seconds)** field, enter the number of seconds for a work order to process before it is dropped from the work order processing queue.

   - From the Order Completion Status list, set the notification for a work order that notifies you whether the work order was completed or failed.

   - From the Environment list, select the activation environment for a work order in which it will get processed.

9. In the Service Actions area, click **Add** to open the Service Action Selection dialog box.

10. On the Service Action Selection dialog box, you can do any one of the following:

    - Select a service action for testing. Do any one of the following and repeat for additional service actions:

      – To view all available service actions, enter an asterisk into the search field.

      – To filter for specific service actions, enter any character or string of characters contained in the service action name to display only the service actions containing those characters.

      – Select the service action you want for the test case and click **OK**.

        The service action name and description appear in the editor.

    - Create a service action for testing. Do the following:

      a. Click **New** to create a service action entity for testing.

         The Service Action Wizard appears.

      b. Enter an action or select a previously defined action from the list (for example, ADD, MOD, DEL, or QUERY).

      c. Enter a name for the entity (for example, SUBSCRIBER, GSM-SUBSCRIBER, ROUTE, TRUNK, or LINE).

      d. (Optional) Deselect **Use recommended name and location** to manually edit the name of the entity and browse for a location.

      e. Click **Finish**.

         In the Studio Projects view, the new service action entity appears in the **Service Actions** folder.

11. Select a service action in the Service Actions area to populate the Service Actions Parameters area with associated data.

    If there is a service action parameter of type XML, then click the **XML Value** button to specify the **.xml** file associated with the parameter. See "Using XML Value in Service Action" for more information.

12. Click **Add** to enable addition of custom parameters, or select specific parameters and click **Remove** to remove them.
13. (Optional) For a service action parameter of type XML, click the **XML Value** button to specify the **.xml** file associated with the parameter.
14. Select the **Order Data** tab of the Activation Test Case editor.
15. Click **Add** in the Global Parameters area to enable addition of a global parameter.

    Enter the parameter name and value. Repeat this for any additional global parameters you want to add.

    > **Note:** A global parameter's value equals identical values of several service action parameters in the work order. For example, if several parameters listed in the **Service Actions** tab have the value **Toronto**, you can represent these by a global parameter, eliminating the need to enter a value for each separately.

16. In the Extended Work Order Properties area, click **Add** to enable addition of an extended work order property.

    Enter the property name and value. Repeat this for any additional extended work order properties you want to add.

17. Save your test case.

    You are now ready to set up and run your activation test case.

Related Topics

Running Activation Test Cases

Testing ASAP Cartridges in Design Studio

Activation Test Case Editor

Using XML Value in Service Action

To use the XML value in a service action that has the XML value parameter:

1. On the Activation Test Case editor, in the Service Actions area, add service actions.

   See "Creating Activation Test Cases" for more information.

2. In the Service Actions area, click a service action.

   In the Service Action Parameters area, the service action parameters appear.

3. Click the row which has **Type** as XML.

   The **XML Value** button is enabled.

   > **Note:** The **XML Value** button is enabled only for an XML type.

4. Create an **.xml** file.
5. In the Package Explorer view, add the **.xml** file under the **model** folder in the current cartridge project.
6. Select **Project**, then select **Clean**.
7. In the Service Action Parameters area, click the row which has an XML type.
8. Click the **XML Value** button.

   The XML Parameter Type Source dialog box appears.
9. Select the **.xml** file which contains the appropriate parameter values.
10. Click **OK**.

    In the Service Action Parameters area, the path of the **.xml** file appears in the **Value** column.

Running Activation Test Cases

After you have created an activation test case (test work order), you can run it in different environments. The purpose is to test, within Design Studio, the activation order completion for a cartridge to verify that a work order is sent to the ASAP environment.

You can run test cases for your project in 2 different ways:

- From the Activation Test Case editor (a single test case of your project can be run in a selected environment).
- From the Activation Project editor (either a single test case can be run, or multiple test cases of your project can be run at the same time in a selected environment.

When running activation test cases, see the following topics:

- About Test Case Configuration
- Running Test Cases from the Activation Test Case Editor
- Running Test Cases from the Project Editor

About Test Case Configuration

Every time you run a test case, a random, unique integer is generated (for example, 770250909). If in the Test Execution section of the Activation Test Case editor you leave the **Primary Key Pattern** field blank or enter *{0}*, the generated integer will be used as primary key to identify the test case (for example, 770250909). If you enter *{1}* as the pattern, the test case name will be used as primary key (for example, TESTCASE001).

If you enter *{1}-{0}* in the **Primary Key Pattern** field, the primary key be the name and generated integer concatenate (for example, TESTCASE001-770250909). You can concatenate other integers or letters (for example, a user name) with the basic patterns to build specific patterns for creation of primary keys. The following table shows some examples of patterns you could use for TESTCASE001 (and its generated integer) to create various primary keys.

Pattern	Generated integer	Primary Key
(blank)	3782695	3782695
{0}	3782695	3782695

Pattern	Generated integer	Primary Key
{1}	3782695	TESTCASE001
{1}-{0}	3782695	TESTCASE001-3782695
USER	3782695	USER
USER-{0}	3782695	USER-3782695
{1}-USER-{0}	3782695	TESTCASE001-USER-3782695

Related Topics

Running Activation Test Cases

Creating Activation Test Cases

Testing ASAP Cartridges in Design Studio

Running Test Cases from the Activation Test Case Editor

To run a test case from the Activation Test Case editor:

1. In the Test Execution area of the Activation Test Case editor, specify the following:
   - The **Test Order Primary Key Pattern** (for primary key).
   - The desired **Timeout** value for the test (default is 30 seconds)
   - The **Order Completion Status** you expect based on the parameters used (either **orderCompleteEvent** or **orderFailEvent**).

     The activation order passes the test if it meets the expected result. For example, if the expected result for the order is to fail but it completes instead, then the order has failed the test.
   - The **Environment** in which you want to run your test.
2. Click **Run** to run the test.

   A test status of PASS or FAIL appears in the Console view. Additionally, information will appear for problems encountered when running the test. If no problems are encountered and the order has passed, then Design Studio submits the work order to the ASAP environment.

Related Topics

Running Activation Test Cases

Creating Activation Test Cases

Testing ASAP Cartridges in Design Studio

Running Test Cases from the Project Editor

To run test cases from the Project editor:

1. In the Activation Project editor, click the **Testing** tab.

   All test cases you created for the project are displayed and are ready to be run at the same time.

   Alternatively, you could prepare to run one or multiple test cases by deselecting the **Include all from Project** check box, clicking **Select**, and then selecting one or more specific test cases from a selection dialog box.

2. From the Environment list, select the environment in which you want to run your tests.
3. Do any one of the following:
   - Click **Run All** to run all test cases from your project simultaneously.
   - Click **Run** to select only one test case from the list.

   A test status of PASS or FAIL appears in the Console view. Additionally, information will appear for problems encountered when running the test. If no problems are encountered and the order has passed, then Design Studio submits the work order to the ASAP environment.

   > **Note:** When selecting **Run All** to run all test cases at once, you have two options for displaying the test results:
   >
   > - Option 1: Display the results for each test case in a separate console (by leaving the **Separate Console for each Test Case** check box checked in the Project editor). You can access a specific console by clicking the down-arrow beside the **Display Selected Console** icon. This displays a list of all available consoles currently on the system, including the consoles for each test case, and enables you to view the results for any each case. For example, if you are viewing results for Test007 and want to view and compare the results for Testcase_Check, then simply click the down-arrow and select Testcase_Check from the list. The results for the selected test case will appear in a separate console.
   > - Option 2: Display the results for all test cases in one console (by deselecting the **Separate Console for each Test Case** check box in the Project editor). The results for all test cases will appear in one console.

   > **Note:** For additional information on consoles and associated icons, refer to the Eclipse online help.

Related Topics

Running Activation Test Cases

Creating Activation Test Cases

Testing ASAP Cartridges in Design Studio

Activation Test Case Editor

Use the Activation Test Case editor to test, within Design Studio, the activation work order completion for a cartridge to verify that the work order is sent to the ASAP environment. In the Studio Projects view, double-click an activation test case entity to open the Activation Test Case editor. You can create the editor using the Activation Test Case Wizard.

When working with the Activation Test Case editor, see the following topics:

- Activation Test Case Editor Service Actions Tab
- Activation Test Case Editor Order Data Tab

Field	Use
Description	Specify a description for the Activation Test Case editor.
Test Order Primary Key Pattern	Specify a pattern for the primary key that identifies a test case on ASAP server. See "About Test Case Configuration" for more information.
Timeout (seconds)	Specify the time, in seconds, for a work order to process. Beyond the specified time, the work order is dropped from the work order processing queue.
Order Completion Status	Select a notification for a work order that is sent to an ASAP user on work order completion or failure.
Environment	Select an environment for a work order to get processed.
Run	Click to execute the test case.

Activation Test Case Editor Service Actions Tab

Use the **Service Actions** tab to define service actions and their associated parameters for the test case.

Field	Use
Add	In the Service Actions area, click to add a service action to the editor.
Remove	In the Service Actions area, click to remove a service action from the editor.
Open	In the Service Actions area, click to open the service action editor for the selected service action.
XML Value	Click to specify the XML file associated with the service action parameter of XML type.
Add	In the Service Action Parameters area, click to add additional parameters for a specific service action in the Service Actions area.
Remove	In the Service Action Parameters area, click to remove service action parameters for a specific service action from the editor.

Related Topics

Activation Test Case Editor

Creating Activation Test Cases

Running Activation Test Cases

Activation Test Case Editor Order Data Tab

Use the **Order Data** tab to define global parameters and extended work order properties for the work orders that will be tested using the test case.

Field	Use
Add	In the Global Parameters area, click to add a global parameter. You can use a global parameter in place of identical values of service action parameters in the work order. See "Creating Activation Test Cases" for more information. In the Extended Work Order Properties area, click to add a property that needs to be enabled in ASAP. See *ASAP System Administrator's Guide* and *ASAP Developer's Guide*.

Field	Use
Remove	In the Global Parameters area, click to remove a global parameter. In the Extended Work Order Properties area, click to remove an extended work order property.

Related Topics

Activation Test Case Editor

Creating Activation Test Cases

Running Activation Test Cases

10

Documenting Cartridges

You can use cartridge guides to document the cartridges that you create in Oracle Communications Design Studio. The guides contain overview and technical information to assist with extending and integrating the cartridge into a customer environment.

When documenting cartridge guides, see the following topics:

- About Cartridge Documentation Guides
- Generating a Network Cartridge Guide

About Cartridge Documentation Guides

The cartridge guides, usually delivered in PDF format, are based on a template. The topics in the guides include cartridge installation and testing, atomic action and service action commands, network element templates, and user defined exit types.

Design Studio provides a cartridge guide generation feature that simplifies this documentation process. The feature becomes available whenever you create an Activation Network Cartridge project to set up an Activation Network cartridge. In addition to providing the template for the guide (with cover, table of contents, chapters, sections, headings, standard text and variables), most of the cartridge documentation is generated by Design Studio and entered in the appropriate area of the template. Developers complete the guide by adding variables and placeholders to subsections (for example, for installation and testing).

The four main subtopics generated by the Design Studio Cartridge Guide feature are:

- Cartridge Overview
- Atomic Service Description Layer (ASDL) Commands
- Service Definitions
- Configuring ASAP to Support Additional NE Instances

> **Note:** The Cartridge Guide template is created as a starting point for documentation when creating an Activation Network Cartridge project, but can be replaced with another file if desired.

Related Topics

Generating a Network Cartridge Guide

Generating a Network Cartridge Guide

When you create an Activation Network Cartridge project to set up a network cartridge, a Documentation group entity is displayed in addition to the cartridge icon under the new project in your Studio Projects view. This feature enables you to create the Cartridge Guide template. After you have built your cartridge, the feature enables you to generate most of the required cartridge documentation.

> **Note:** The Cartridge Guide template is created as a starting point for Activation Network Cartridge project documentation; you can, however, replace the template with a different file.

To generate a cartridge guide:

1. In the Studio Projects view, expand the contents of the **Documentation** group.

   The **Cartridge Guide** file and **Templates** folder appear.

2. Double-click the **Cartridge Guide** file.

   The standard template for creating cartridge guides opens in MS Word as an RTF file.

3. Scroll down the cartridge guide template until you reach the Cartridge Model section with the text *Paste Model Documentation Here*.

4. Right-click on the cartridge project and select **Copy Documentation to Clipboard** from the context menu.

5. In the Cartridge Guide template's Cartridge Model section, highlight *Paste Model Documentation Here*.

6. Right-click and select **Paste** from the context menu.

   The Design Studio-generated documentation for the cartridge now appears below the Cartridge Model heading.

Related Topics

Editing Topic Overviews in the Cartridge Guide

Documenting Cartridges

Editing Topic Overviews in the Cartridge Guide

You can edit the overview text for any of the four generated documentation topics by making changes to the appropriate overview template file. Consider the following example, which demonstrates how to change the overview paragraph below the Atomic Actions heading:

To edit the Atomic Action topic overview section:

1. In the Studio Projects view, expand the contents of the **Templates** folder.

   Several files appear, each containing overview text for the indicated topic.

2. Double-click the **Atomic Action Overview** file to display its content in the text editor.

3. Edit the overview text as needed.

   When you have completed the updates, right-click in the editor and select **Save** from the context menu to save your changes.

Related Topics

Generating a Network Cartridge Guide

Documenting Cartridges