

User Guide for Designer Studio

Ericsson Dynamic Activation 1

USER GUIDE

Copyright

© Ericsson AB 2017. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing. Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Introduction	1
1.1	Purpose and Scope	1
1.2	Target Groups	1
1.3	Prerequisites	1
1.4	Typographic Conventions	1
2	Important Release Note	3
3	Functional Overview	5
3.1	Concepts	6
3.2	Solution Overview	10
3.2.1	Designer Studio	11
3.2.2	Relation to Dynamic Activation System	11
3.3	Key Functions	12
3.3.1	Data Model Transformation	12
3.3.2	Execution Plan	13
3.3.3	Data Integrity	13
4	Designer Studio Use Cases	15
5	Installing Designer Studio	17
5.1	Configuration	18
6	Upgrading Designer Studio	19
6.1	Preparation	19
6.2	Upgrading from Multi Activation 16.0 CP2	19
6.3	Upgrading from Earlier than Multi Activation 16.0 CP2	20
6.3.1	Install the Latest Version of Designer Studio	20
6.3.2	Migrate the Service Models Manually	21
6.4	Roll Back to the Previous Version	22
7	Updating Southbound Interface	23
7.1	Update for Resource Configuration Services	23
7.2	Update for Off-the-shelf Supported Subscriber Activation Services	25
7.3	Update for Customer Adaptations and Existing Service Models	25
8	Preparing Northbound Interface	27



8.1	WSDL and XSD Examples	27
9	Configuring a Service Model	29
9.1	Initiate a New Service Model	29
9.2	Add Task to an Operation	29
9.2.1	Delete or Rename a Task	30
9.3	Assign Parameters for a Task	31
9.3.1	Slide-in Panel	32
9.3.2	Parameter Assignment Setting	34
9.3.3	Parameter Assignment Condition	36
9.4	Define Response for a Task	37
9.5	Define Condition for a Task	38
9.6	Define Error Handling for a Task	41
9.6.1	Accept Error Codes	41
9.6.2	Enable Rollback	42
9.7	Error Handling Task	43
9.7.1	Condition After	45
9.8	Update of Northbound Interface for a Service Model	47
9.9	Service Model Examples	48
9.9.1	Example FTTH	49
9.9.2	Example Voice over LTE	51
10	Optional Operations	55
10.1	Export a Service Model	55
10.2	Copy, Rename, or Delete a Service Model	55
11	Deploying and Undeploying a Service Model	57
11.1	Prerequisites	57
11.2	Deploy a Service Model	57
11.3	Undeploy a Service Model	58
12	Service Model Error Response	59
13	Verifying a Service Model	63
14	Maintenance of Designer Studio	65
14.1	Troubleshooting A Service Model	65
14.2	Clean up Temporary Files	66
15	Help Center	67
15.1	Learn More	67
15.2	User Guide	67
15.3	Support	67



16	Appendix	69
16.1	Known Limitations and Solutions	69
16.2	Transformation Expression Functions	69
16.2.1	CONCAT	69
16.2.2	SUBSTR	69
16.2.3	CHOICE	70
16.2.4	LOOKUP	70
16.2.5	MATCH	71
16.2.6	SKIP	71
	Reference List	73





1 Introduction

1.1 Purpose and Scope

This document provides the user with information about the Designer Studio solution.

Note: The instructions in this document are based on Windows environment. It can differ for other personal computer or server environment.

1.2 Target Groups

The target groups for this document are as follows:

- Solution Architect
- Solution Integrator
- Business Configuration Engineer

1.3 Prerequisites

Users of the Designer Studio must:

- Have general understanding of the Ericsson Dynamic Activation (EDA) product, for detail, refer to *Product Overview*, Reference [1].
- Have general understanding of the CAI3G interface, for details refer to *Generic CAI3G Interface 1.2*, Reference [5] and *CAI3G Implementation*, Reference [6].

1.4 Typographic Conventions

Typographic conventions are described in *Library Overview*, Reference [2].





2 Important Release Note

In the current release, service models cannot be deployed directly from Designer Studio to the Dynamic Activation system.

Service models deployment must be performed manually. For instruction, see Section 11 on page 57.





3 Functional Overview

Designer Studio is a graphical configuration tool for designing services that reflects the business need of northbound systems. Designer Studio is built upon a technology innovation that separates service realization objective from execution logic. The service execution logic is a platform feature of Dynamic Activation. It offers dynamic execution plan and built-in rollback. The user only needs to define service realization objective by using Designer Studio. Designer Studio does not require users to have programming skills.

Figure 1 illustrates the business logic architecture for Dynamic Activation. Details of the layers are described in *Customization - Architectural Overview*, Reference [7]. Service models configured with Designer Studio implement solution on the Service Realization layer. Services implemented on network abstraction and network implementation layers are the building blocks for configuring a service model. Other service models can also be used as building blocks for a service model.

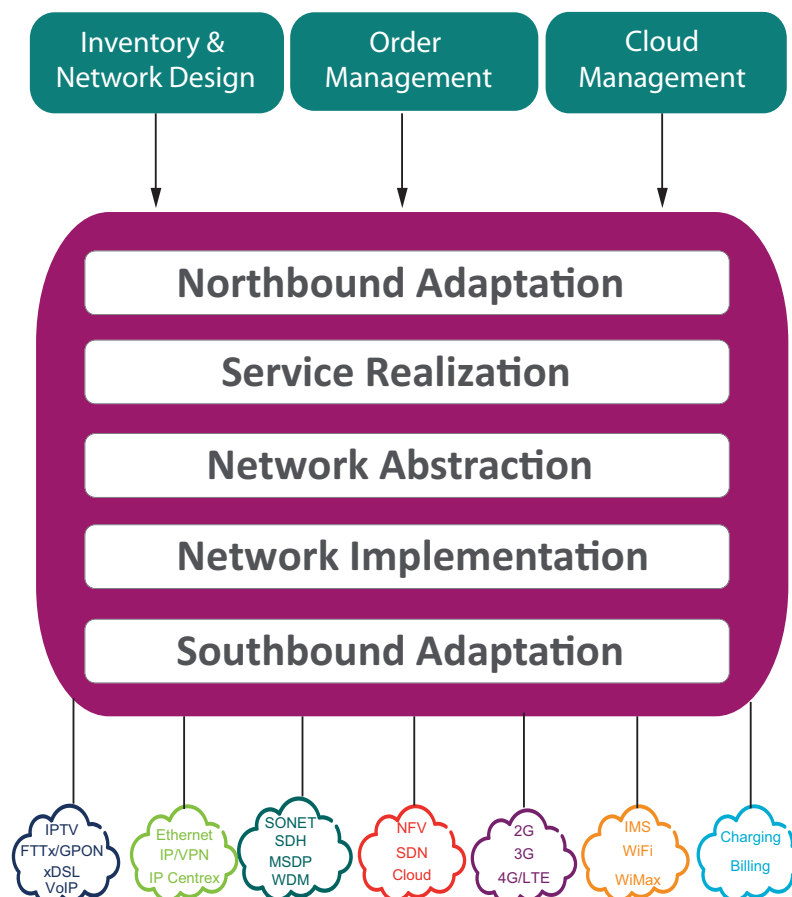


Figure 1 Business Logic Architecture

3.1 Concepts

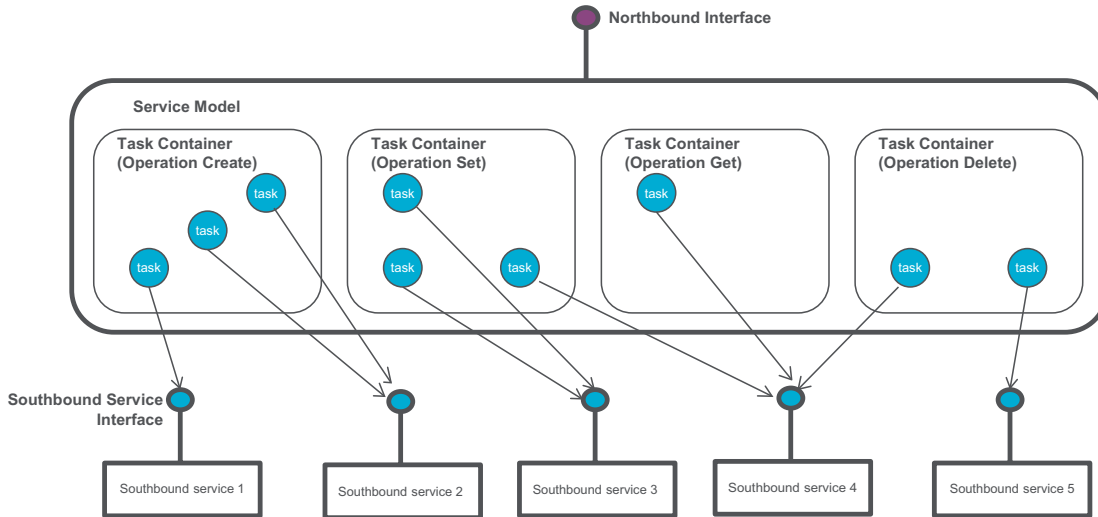


Figure 2 Concept Relations

Service model

A service model (area 1 in Figure 3) implements a service on the service realization layer. A service on this layer is technology independent and vendor neutral. The purpose of such services is to ease integration towards northbound systems. A service model consists of a maximum of four task containers. Each task is corresponding to one of the operations defined in the Northbound Interface of the service (Create/Set/Get/Delete). A service model operation container consists of one or more tasks, each calling for a southbound service.

Northbound Interface

A Northbound Interface exposes the service implemented by a service model. The Northbound Interface is described with a CAI3G based Web Services Description Language (WSDL) file and one or more XML schema files. A call to the service is referred to as a northbound request.

The content of the Northbound Interface is shown in area 5 in Figure 4.

**Southbound service**

A southbound service is a relative term addressing a service that is hidden by the northbound system. It could be a service implement on network implementation layer, on network abstraction layer or even a service on the service realization layer when deemed necessary. A southbound service must be a service that can be executed by Dynamic Activation, either standard product feature or customer adaptation.

Southbound interface

A southbound service is exposed by its southbound interface, which is a web service and is described with:

- A CAI3G based WSDL file.
- one or more XML Schema Definition Language (XSD) files.

The content of the southbound interface of the task is shown in area **6** in Figure 4.

Task Container

A task container hosts all tasks for a given operation of the service model. The number of task containers generated for a service model is based on the number of operations defined in the Northbound Interface WSDL file. Operation types supported in CAI3G are Create/Delete/Get/Set. One or more tasks can be added to each task container. For example, a Create task container can contain tasks of any of the Create/Delete/Get/Set types.

The overview of all tasks for the task container and their dependencies is shown in area **2** in Figure 3.



Task

A task (area 4 in Figure 3) is an action to be taken to realize a part of the objective of a northbound request. Typical tasks are describes as below and illustrated in Figure 2.

- A task is an operation of a southbound service.

For example, task container `Create` includes a create task to create `SouthboundService1`.

- Multiple tasks in a task container can call for different southbound service.
- Tasks from different task containers can call for one southbound service (`SouthboundService4` in Figure 3).

A task consists of three key attributes: parameter assignments, conditions, and error handling.

- Parameter assignments: The parameters of a task are inherited from the selected southbound service schemas. Parameters of the task are to be assigned with parameters of a source. The source is either the northbound service or another task in the same service model operation container. Assigning parameter is a mean to transform data model between the source (northbound requests or other tasks) and the southbound service, and to define default profile. Parameter assignment conditions can be defined for each task parameter. The assigned parameters of a task implicitly define the objective of the task.
- Task condition: The conditions of a task regulate that whether and when a task is to be executed for a particular northbound request.
- Error handling: Two methods are supported for handling runtime errors of a task.
 - Accept error codes: Allow Service Model continuing execution even when certain errors or any errors occur on a task, without triggering any rollbacks.
 - Enable rollback: Rollback is crucial in maintaining data consistency from a service perspective. Dynamic Activation comes with default rollback implementation.



Error handling task

Error handling tasks are executed when error occurs in other tasks. This is by defining condition error response from other tasks. Multiple conditions can be defined for an error handling task. Built-in rollback of a task and error handling tasks can be combined to ensure rollback of a northbound request.

Dependencies

There are two types of dependencies (area 3 in Figure 3) for a task: parameter dependency and condition dependency. The dependencies influence the execution order of the tasks for each northbound request.

- Parameter dependency is defined by assigning task parameter to a source parameter. Often the source is the Northbound Interface but it can also be response of another task of type Get.
- Condition dependency is defined on per task basis. Condition can be set on other tasks or the Northbound Interface.

Figure 3 and Figure 4 show where in the GUI the concepts are reflected.

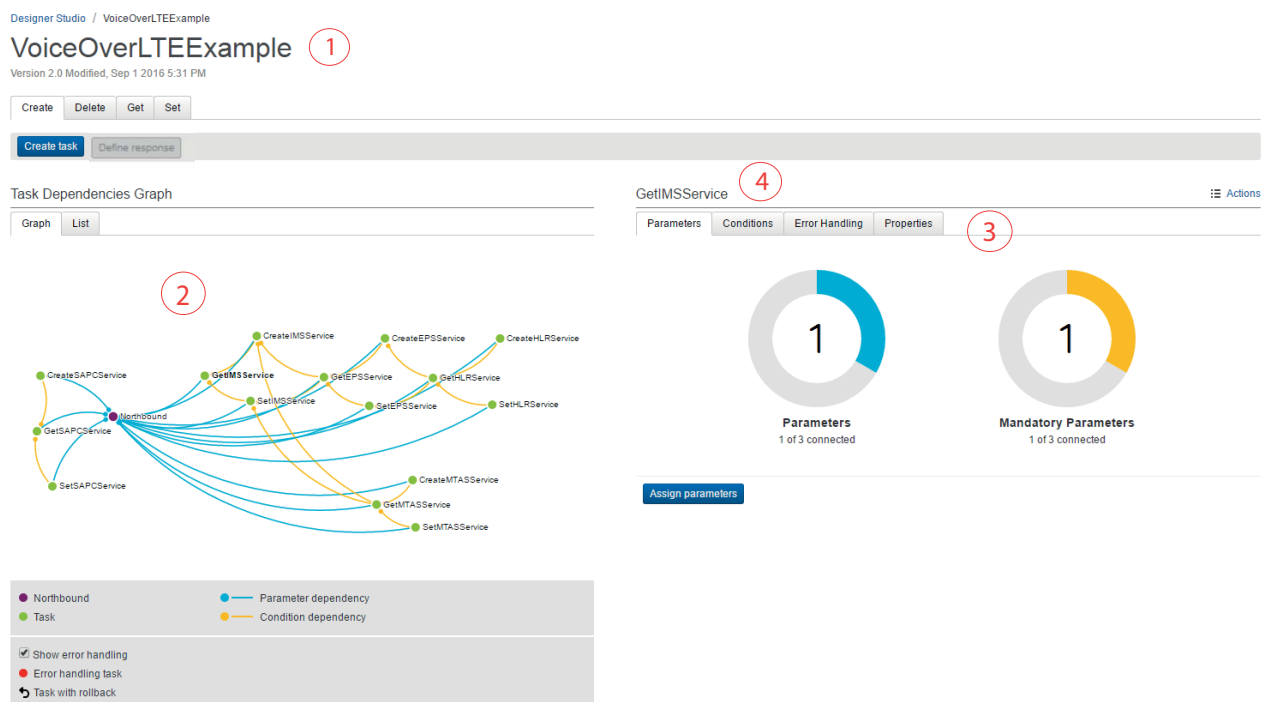


Figure 3 Concepts in GUI part 1

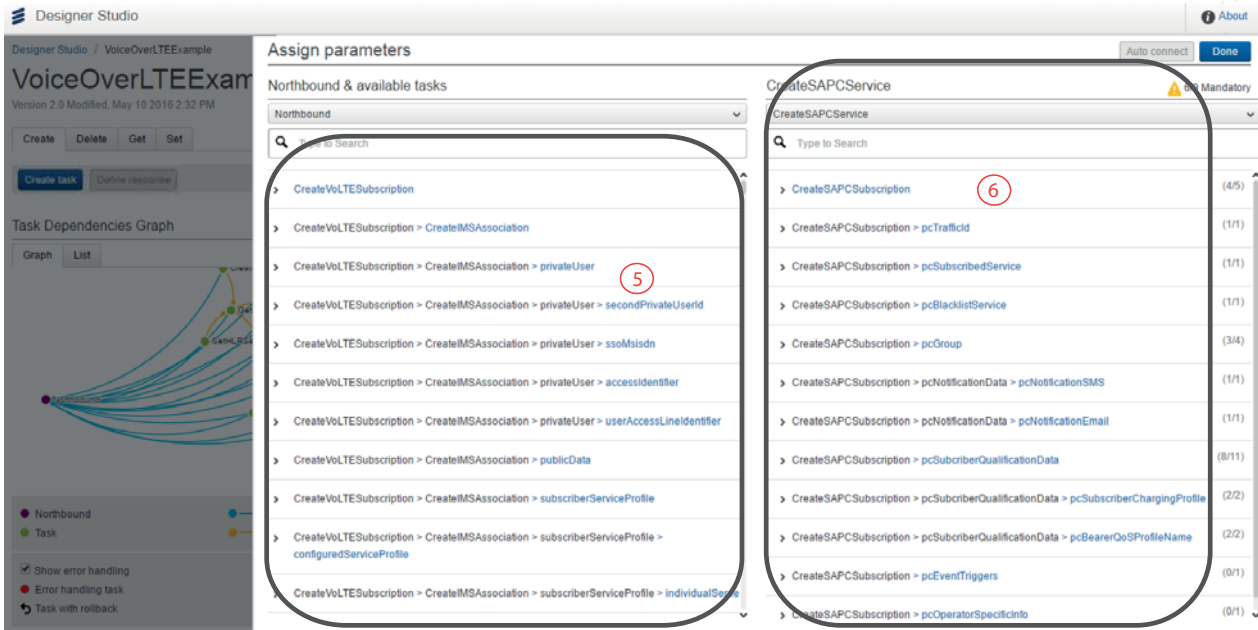


Figure 4 Concepts in GUI part 2

3.2 Solution Overview

The overall solution to manage a service model is illustrated in Figure 5. For information about functionality in Dynamic Activation system, refer to *Function Specification Dynamic Activation Execution Environment*, Reference [3].

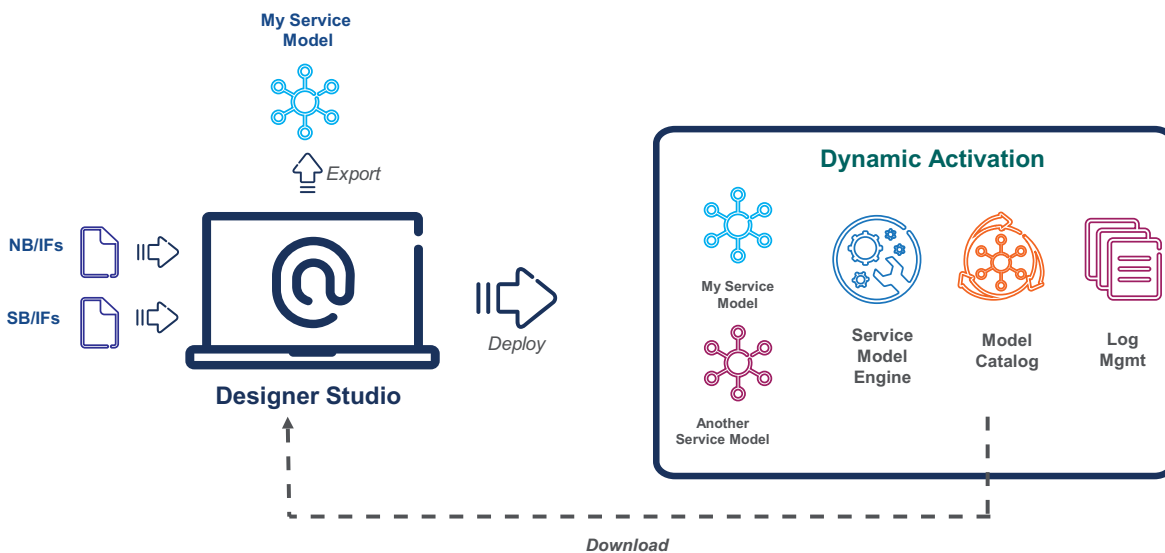


Figure 5 End-to-End Solution for Service Model

The following subsections provide detailed functionality of the solution.



3.2.1 Designer Studio

Designer Studio is a web application and is pre-bundled with an embedded runtime environment. It can be installed independently from the Dynamic Activation system.

Designer Studio comes with download function that downloads southbound interfaces function from the model catalog in the Dynamic Activation system.

Designer Studio supports the so called top-down approach for the service modeling. This means that the WSDL and schema files must be prepared before configuring a service model.

When configuring a service model, it is possible to:

- Aggregate southbound services in a service model by defining tasks.
- Select only the desired functionality of a southbound service by assigning only a subset of parameters of the southbound interface.
- Transform data model between a northbound service and southbound services, for example, transform subscriber id to MSISDN.
- Define default or override value for parameters.
- Specify conditions for tasks and for parameter assignments.
- Define acceptable error codes or configure rollback for error handling of a task.
- Define error handling task.

In the current release, Service models cannot be deployed directly from Designer Studio to a Dynamic Activation system.

3.2.2 Relation to Dynamic Activation System

3.2.2.1 Dynamic Activation Software Licence

Service models created in the Designer Studio require an SW Advanced license to run on the Dynamic Activation system.

3.2.2.2 Model Catalog

WSDL and schemas for Resource Configuration southbound services are stored in the model catalog in the Dynamic Activation system. The model catalog can be accessed directly from Designer Studio.

The WSDL and schemas for other southbound services in the standard product can be found in Customer Product Information library and can be uploaded manually.



3.2.2.3 Service Model Engine

The Designer Studio application depends on the Service Model engine in the Dynamic Activation system. Service models created in the Designer Studio application must be run on the corresponding version of Dynamic Activation system as shown in Table 1.

Table 1 Service Model Engine Dependency

Designer Studio	Compatible to
3.0.4 or later ⁽¹⁾	Dynamic Activation 1

(1) Contact the local Ericsson representative for version information.

Service model engine is a dedicated component in Dynamic Activation that executes the deployed service models. This engine is a highly efficient logic tailored for service realization. It provides the following main functions:

- Decide on per request basis the task execution pattern based on the parameter dependencies and task conditions define in the service model. A task is executed when the conditions are met. This allows parallel execution of tasks when possible.
- Perform rollback as defined in the service model. For details of rollback solution, see Section 3.3.3 on page 13.

3.2.2.4 Log Management

The Log Management GUI in Dynamic Activation system shows the northbound requests and their responses. For each northbound request, it also shows that the commands sent to the network elements or devices and their response.

This GUI can be used to verify that the Service Model is working as intended.

3.3 Key Functions

This section describes the key functional capabilities offered by the Designer Studio solution and the fundamental principles.

3.3.1 Data Model Transformation

Transformation of the Northbound Interface data model to one or more southbound interface data models is realized by parameter assignments of tasks. For details of parameter assignment, see Section 9.3 on page 31.

The available transformation functions are:

- Assign parameter with different name between the target and source.
- Define source value with default for a target parameter.



- Define fixed value for a target parameter.
- Define transformation value for a target parameter
- Define condition for parameter assignments.

For other transformation need, the work-around solution is to implement the logic as a business logic to deploy on Dynamic Activation system.

3.3.2 Execution Plan

Execution plan for a northbound request is determined in runtime, based on the following factors:

- Parameter dependences defined in the service model.
- Task conditions defined in the service model.
- Parameters and values provided in the northbound request.
- Execution result of tasks with dependencies.

The first two factors are configured by user in design time in Designer Studio. Thus the graph of a service model illustrates dependencies between tasks, not the execution plan itself.

Note: Execution plan is optimized for achieving high performance. Dependencies, such as parameter conditions or task conditions, force service model to execute in a specific way when there are business or NE technical needs.

3.3.3 Data Integrity

Data integrity, in the provisioning context, means that a northbound request is either executed completely or no changes are done at all.

To ensure data integrity in the end-to-end provisioning process, all the services involved in the process must implement function to handle error scenario. Such functions are addressed as loose error handling, fault tolerance and rollback in Dynamic Activation. For more information about these functions, refer to *Function Specification Resource Activation*, Reference [3].

Rollback is provided by the Designer Studio to facilitate the data integrity strategy. Default rollback behavior is implemented in the Service Model Engine in Dynamic Activation system. User decides, at configuration time in Designer Studio, whether to enable rollback behavior for a task. When enabled, the Service Model Engine triggers default rollback if the task successes but another task fails owing to an unacceptable error.

If the default rollback behavior is not sufficient, error handling tasks can be added to the service model. Condition for an error handling task is one or more error responses of other tasks.



Default rollback and additional error handling tasks can be combined to ensure rollback of a northbound request.

For configuration of acceptable error codes, see Section 9.6.1 on page 41.

For configuration of rollback, see Section 9.6.2 on page 42.

For configuration of error handling task, see Section 9.7 on page 43.



4 Designer Studio Use Cases

Typical use cases when working with the Designer Studio are the following:

- Installing the latest version of Designer Studio, see Section 5 on page 17.
- Upgrading Designer Studio to be compatible with the latest Dynamic Activation, see Section 6 on page 19.
- Managing southbound interface, see Section 7 on page 23.
- Prepare Northbound Interface, see Section 8 on page 27.
- Configuring service model, see Section 9 on page 29.
- Exporting service model, see Section 10.1 on page 55.
- Deploying service model, see Section 11 on page 57.
- Verifying service model, see Section 13 on page 63.
- Maintenance and troubleshooting of Designer Studio, see Section 14 on page 65.





5 Installing Designer Studio

Note: Supported web browsers to access the Designer Studio GUI are:

- Chrome
- Firefox

1. Make sure to have the Designer Studio software accessible.
2. Java 8, or later, must be installed before the Designer Studio is started.

Note: Ensure that a proper 32-bit or 64-bit version is installed correctly.

If JDK is used, the `<JAVA_HOME>` environment variable must be set to the JDK installation. If JRE is installed, the `<JRE_HOME>` environment variable must be set to the JRE installation. If both `<JAVA_HOME>`, and `<JRE_HOME>` variables exist, the `<JRE_HOME>` is used.

3. Make sure that the port allocated for Designer Studio (default 8080) is not occupied by another application in the machine. For example, if Designer Studio is deployed together with Consistency Checker on the PC or on the same server as Dynamic Activation, port 8080 is occupied. If necessary, follow the instruction in Section 5.1 on page 18 to change the default port.
4. Unzip the file to the directory where the Designer Studio is to be installed. User must have write and execution permission to the directory. The location of the installed Designer Studio software is from now on called `DS_SW` in this document.

To start the Designer Studio application, go to `<DS_SW>\bin` folder. If using Windows as operating system, double-click the `startup.bat` file. If using another operating system, run the `startup.sh` script.

When the Designer Studio is started for the first time, a folder and a default configuration file are created under the user home directory called `.DesignStudio`. From now on in this document, the location of the `.DesignStudio` directory is called `DS_HOME`.

5. Open a web browser, access the GUI through the URL:

`http://<hostname>:<port>/designerstudio`

For example:

`http://localhost:8080/designerstudio`

6. To shut down the Designer Studio application, close the Designer Studio web browser. Go to `<DS_SW>\bin` folder. If using Windows as operating



system, double-click the `shutdown.bat` file. If using another operating system, run the `shutdown.sh` script.

5.1 Configuration

Designer Studio works out of the box without any additional configurations. However, it is possible to change the default configurations.

If a self-signed certificate is used between the Designer Studio instance and the Dynamic Activation server, do the following:

1. Locate the `config.properties` file under `<DS_HOME>`.
2. Change the value of `ds.http.relaxedssl` to `true`.

To change path to, for example, the designer studio work space, edit the `config.properties` file located in `<DS_HOME>`.

To change the port of the Designer Studio server, edit the `Connector port="8080"` tag in the `server.xml` file located in `DS_SW\conf`.

All configuration changes require restart of the Designer Studio software.




6 Upgrading Designer Studio

This section instructs how to upgrade the Designer Studio to the latest version, so the created service models can run on the latest Dynamic Activation system.

6.1 Preparation

Before upgrading the Designer Studio, perform the following procedure:

1. Close the internet browser window that is running the Designer Studio GUI.
2. Clear the internet browser caches. For example, do the following in the Chrome:
 - a. Click  on the top right corner of the Chrome window.
 - b. In the displayed menu, select **More tools>Clear browsing data**.
 - c. In the popped dialog, ensure that the **Cached images and files** check box is ticked.
 - d. Click **Clear browsing data**.
3. Shut down the Designer Studio as follows:
 - a. Go to the `<DS_SW>\bin` folder.
 - b. For Windows, double-click the `shutdown.bat`.

For other operating system, run the `shutdown.sh` script.
4. Create a folder for archive, and copy the current `<DS_SW>` and `<DS_HOME>` folders to the archive folder.
5. Ensure that a proper 32-bit or 64-bit version of Java 8 is installed.
6. Ensure the environment variable `<JAVA_HOME>` or `<JRE_HOME>` is set to Java 8 installation folder. If both variables are set, `<JRE_HOME>` is the one that takes effect.

6.2 Upgrading from Multi Activation 16.0 CP2

To upgrade the Designer Studio of Multi Activation 16.0 CP2 to the latest version:

1. Make sure to have the Designer Studio software accessible.



2. Unzip the file to the directory where the Designer Studio is to be installed. User must have write and execution permission to the directory. The location of the unzipped folder is called `<DS_SW>` from now on.
3. Start the Designer Studio as follows:
 - a Go to the `<DS_SW>\bin` folder.
 - b For Windows, double-click the `startup.bat`.

For other operating system, run the `startup.sh` script.
4. Open the Designer Studio GUI in a web browser through the URL, for example:
`http://localhost:8080/designerstudio`

If all existing service models are visible on the **Service Models** page, the Designer Studio is upgraded successfully.

6.3 Upgrading from Earlier than Multi Activation 16.0 CP2

Because of major improvements on efficiency and robustness, the Designer Studio is not compatible backward to Service Model engines in Multi Activation 16.0 CP2 or earlier release. Therefore manually migrating the existing service models is required.

6.3.1 Install the Latest Version of Designer Studio

1. Make sure to have the Designer Studio software accessible.
2. Unzip the file to the directory where the Designer Studio is to be installed. User must have write and execution permission to the directory. The location of the unzipped folder is called `<DS_SW>` from now on.
3. Start the Designer Studio as follows:
 - a Go to the `<DS_SW>\bin` folder.
 - b For Windows, double-click the `startup.bat`.

For other operating system, run the `startup.sh` script.
4. Open the Designer Studio GUI in a web browser through the URL, for example:
`http://localhost:8080/designerstudio`

In the newly installed Designer Studio, the **Service Models** page shows an empty service model list, and a new button **Import service models**.



Designer Studio / Service Models

Service Models

[Update southbound interfaces](#)
[Add new service model](#)
[Import service models](#)

i No Service model have been created yet

[Add a new Service Model](#) [Import a Service Model](#)

6.3.2 Migrate the Service Models Manually

To migrate the existing service models to the upgraded Designer Studio manually, do as follows:

1. In the **Service Models** page, click **Import a service models**.
2. In the popped dialog, click **Choose Files**, select one or more service model files (*.tar.gz) in the <DS_HOME>\workspace\servicemodels\ folder, and then click **Import**.

Note: Hands on adjustment of the service model can be required. Therefore Ericsson recommends importing one service model at a time.

A report is generated recording any changes made during the import process. Figure 6 shows an example.

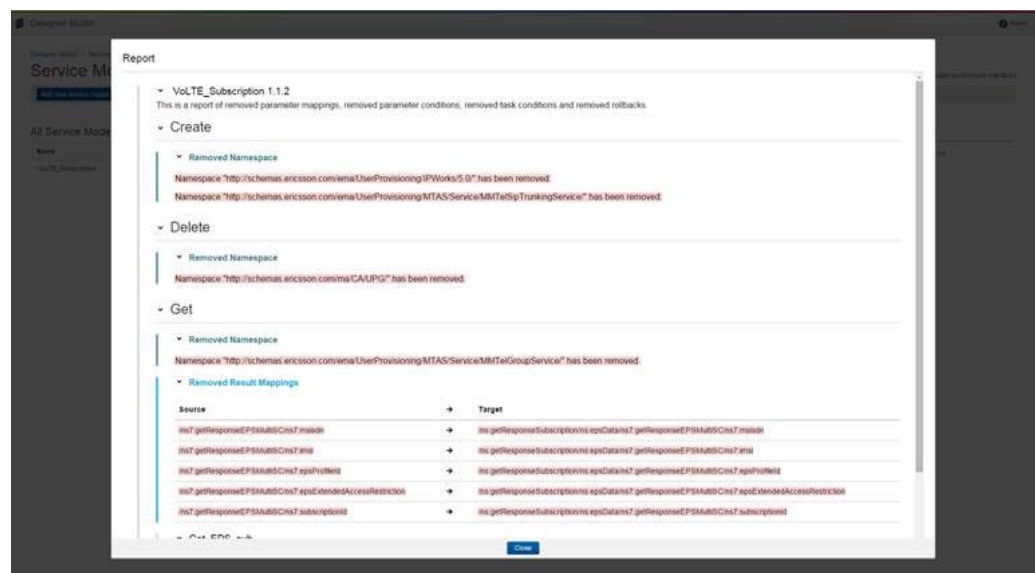


Figure 6 Import Report Example

3. Check the report and save it if significant changes were made during the import.



4. If the report says that any parameter is removed, do the following:
 - a In the **Service Models** page, select the newly imported service model, and click **Edit**.
 - b Check the tasks of the service model, and manually re the parameters if necessary.
 - c Click the **Designer Studio** link to return to the **Service Models** page.

Note: Namespaces not used in any configuration can be removed without impact on service models. Therefore such removal report can be ignored.

6.4 Roll Back to the Previous Version

To roll back the Designer Studio to the version before upgrading, do as follows:

1. Close the internet browser window that is running the Designer Studio GUI.
2. Clear the cache of the browser.
3. Shut down the Designer Studio application.
4. Create an archive folder, and move the *DS_SW* and *DS_HOME* folders to an archive folder.
5. If the previous Designer Studio uses Java 7, change the environment variable *<JAVA_HOME>* or *<JRE_HOME>* back to Java 7.
6. Restore the *DS_SW* and *DS_HOME* archived in Section 6.1 on page 19.



7 Updating Southbound Interface

Designer Studio uses an update wizard to update the southbound interfaces of the following:

- Off-the-shelf supported Subscriber Activation services
- Multi-vendor network element customer adaptations and existing service models
- Resource Configuration

Figure 7 shows how to open the update wizard.

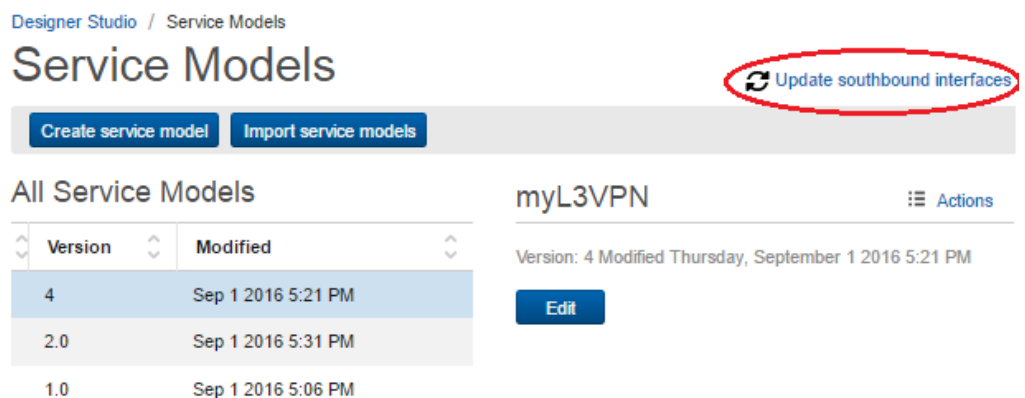


Figure 7 Open Update Southbound Interfaces Wizard

The update wizard creates a backup of Designer Studio workspace, that is, all contents under `<DS_HOME>\workspace`. This backup is stored in the default download folder. The workspace can be rolled back by overriding the workspace with this backup if necessary. The backup is empty the first time this function starts.

7.1 Update for Resource Configuration Services

This procedure replaces the existing content with new ones, and affect configurations in existing service models.

1. In Designer Studio, click **Update southbound interfaces** to open the update wizard.
2. Follow the wizard instruction, and in **2 Select interface location**, select **Download from VIP OAM HOST**.



Update southbound interfaces

1 Backup ✓

2 Select interface location

3 Update interfaces

4 Generate reports

Choose method to update schemas from

☒ Download from VIP OAM HOST
This method updates southbound interfaces for service configuration services. These southbound interfaces can be downloaded from the runtime system via the VIP OAM HOST. This will replace the existing content with new one.

☐ Use downloaded zip file
This method updates southbound interfaces for off-the-shelf supported subscriber activation services. The zip file can be downloaded from the Designer Studio User Guide. This will replace the existing content with new one.

☐ Use local files
This method updates southbound interfaces for customer adaptations and existing service models. The interface files (.xsd, .wsdl, .xml) must be made available on the local machine. This will replace the existing files that have the same names with new files.

Cancel

← Previous

Next →

3. Follow the wizard to finish updating the southbound interfaces.

Update southbound interfaces

1 Backup ✓

2 Select interface location ✓

3 Update interfaces

4 Generate reports

Update from VIP OAM host

VIP OAM host

hostname or ip

Southbound interfaces will be downloaded from the model catalog in Multi Activation.

Cancel

← Previous

Next →

4. Go through the report that is created by the wizard, and update the service models accordingly if necessary.
5. When the updated southbound interface affects a service model, for all tasks that has rollback enabled, uncheck the rollback setting and check it again.



7.2 Update for Off-the-shelf Supported Subscriber Activation Services

This procedure replaces the existing content with new one, and affect configurations in existing service models.

1. Save the zip file, [Multi_Activation_WSDL_and_XSD_files.zip](#), to a local folder.
2. In Designer Studio, click **Update southbound interfaces** to open the update wizard.
3. Follow the wizard instruction, and in **2 Select interface location**, select **Use downloaded zip file**.

Update southbound interfaces

1 Backup ✓

2 Select interface location

3 Update interfaces

4 Generate reports

Choose method to update schemas from

☐ Download from VIP OAM HOST
This method updates southbound interfaces for service configuration services. These southbound interfaces can be downloaded from the runtime system via the VIP OAM HOST. This will replace the existing content with new one.

☒ Use downloaded zip file
This method updates southbound interfaces for off-the-shelf supported subscriber activation services. The zip file can be downloaded from the Designer Studio User Guide. This will replace the existing content with new one.

☐ Use local files
This method updates southbound interfaces for customer adaptations and existing service models. The interface files (.xsd, .wsdl, .xml) must be made available on the local machine. This will replace the existing files that have the same names with new files.

Cancel Previous Next

4. Follow the wizard to update the southbound interfaces by using the downloaded zip file.

7.3 Update for Customer Adaptations and Existing Service Models

This procedure replaces the existing files that have the same `MOType` and the same filename with new files, and affect configurations in existing service models.

1. In Designer Studio, click **Update southbound interfaces** to open the update wizard.



2. Follow the wizard instruction, and in **2 Select interface location**, select **Use local files**.
3. Follow the wizard to update the southbound interfaces by using the local .xsd and .wsdl files.

Update southbound interfaces

1 Backup ✓

2 Select interface location ✓

3 Update interfaces

4 Generate reports

Update from local storage

Upload local schemas (.wsdl, .xsd).

Attention: Make sure that all schemas that are not wsdl-files are linked.

Choose Files

Files uploaded

✓ MNP.xsd	✕
✓ MNP.wsdl	✕

Cancel

← Previous

Next →



8 Preparing Northbound Interface

Northbound interface, for example WSDL and XSD files, must be prepared before creating a service model in Designer Studio. Such files can be edited using any XML editor or text editor.

Note: The MO name and MO namespace **MUST** be unique for a Dynamic Activation system.

A valid WSDL file can be constructed in many ways. However, the examples included in this chapter are the verified practice. Ericsson recommends using these example files as the template for constructing own Northbound Interfaces.

8.1 WSDL and XSD Examples

An example of WSDL and XSD files that describe the provisioning interface, can be found in the downloadable zip file. The zip file contains the following files:

- `myService.wsdl` – defines the MO Type, MO Ids, and operations.
- `myService.xsd` – defines the MO Attributes
- `myService_types.xsd` – defines the common data types
- `PGFault.xsd` - defines the PGFault

Download this zip file by following the instruction below:

1. Save the zip file, [Northbound_Interface_Templates](#), to a local folder.
2. Unpack the zip file.
3. Use an appropriate XML editor to read and update the templates.





9 Configuring a Service Model

This section contains information that is important when configuring a service model.

9.1 Initiate a New Service Model

When creating a service model, upload the WSDL file and its dependent XSD files as prepared in Section 8 on page 27.

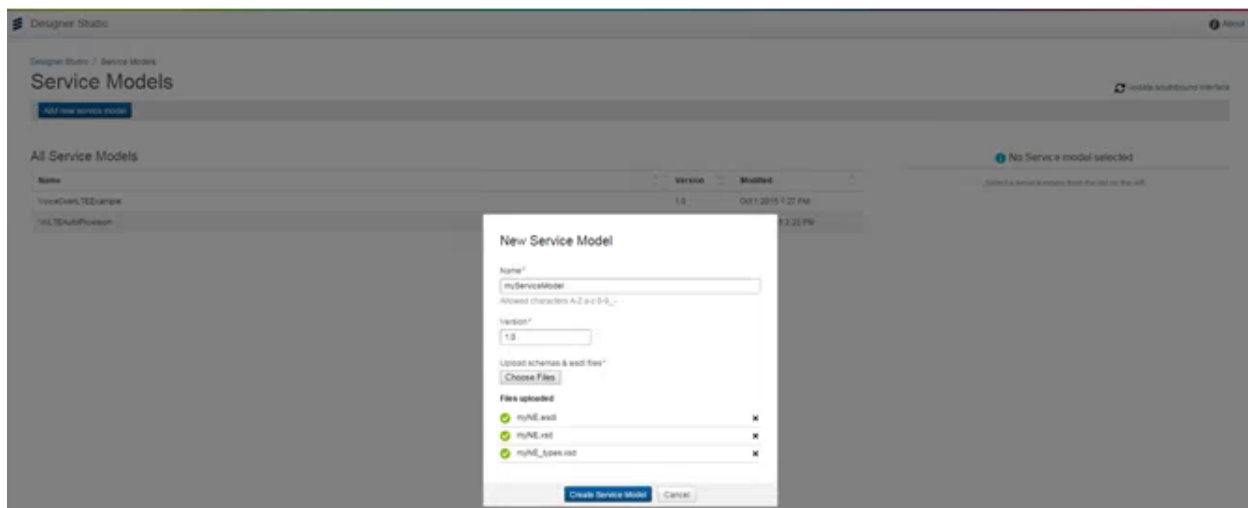


Figure 8 Upload Files

Designer Studio initiates a new service model skeleton based on the supported operation types (create, get, set, and delete) specified in the WSDL file.

Define response function is active for operation type Get and passive for other operation types.

9.2 Add Task to an Operation

One or more tasks can be added to each northbound operation type. This is to combine southbound services for the service model. By adding tasks, the objective of the service model is defined.

When adding a task, the available southbound interfaces downloaded to the Designer Studio workspace are shown in the drop-down list. Select the southbound service for the specific task.

Operation types specified in the WSDL file of the selected managed object are shown for selection. Select the operation type that the task is to perform.



Each task must be given a unique name within a task container. This makes it possible to add more than one task with the same southbound interface and operation type, but with different objectives. Check the box if the task is intended for error handling purpose only.

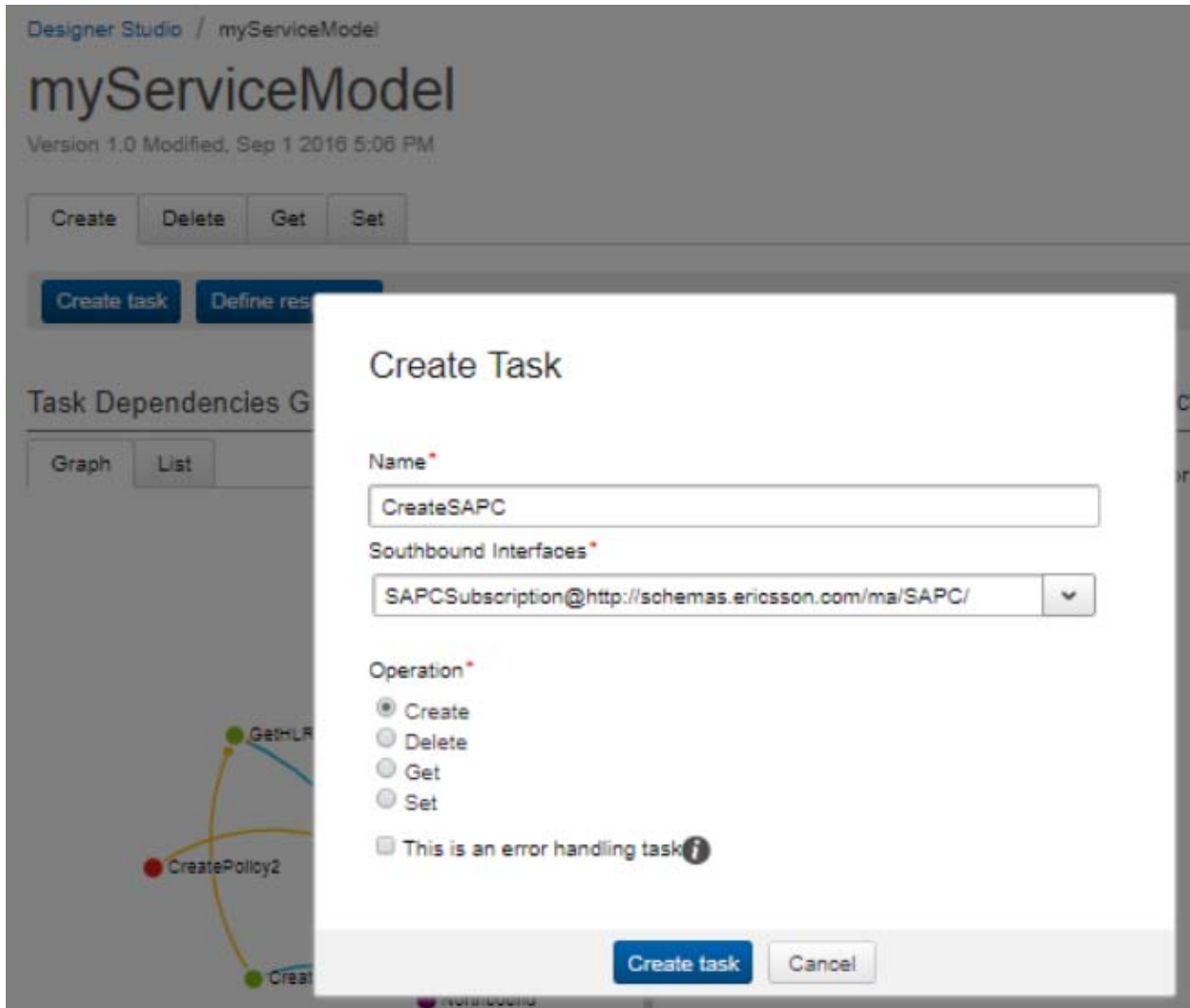


Figure 9 Create Task

9.2.1 Delete or Rename a Task

Select a task and use the **Action** link to delete or rename the task, as shown in Figure 10.



Designer Studio / myServiceModel

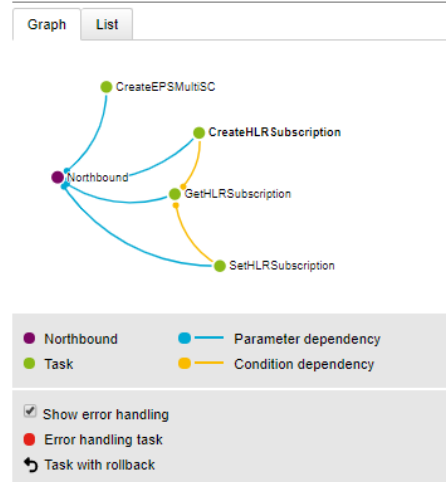
myServiceModel

Version 1.0 Modified, Sep 1 2016 5:06 PM

Create Delete Get Set

Create task Define response

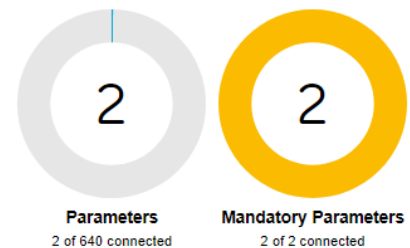
Task Dependencies Graph



CreateHLRSubscription

Actions

Parameters Conditions Error Handling Properties



Assign parameters

Figure 10 Delete or Rename a Task

9.3 Assign Parameters for a Task

Assigned parameters specify from where the task gets the data. This is achieved by identifying the source parameter for a task parameter.

Pay attention to the following of the task parameters:

- **MOID:** The MOID of the task must always be assigned.
- **Optional parameters:** which and how many optional parameters to be assigned are up to each use case.
- **Multiple value parameters:** when assigning multiple value parameters, make sure that the recurrence is compatible between the target interface and the source interface. Fixed value must not be assigned to a multiple value parameter.
- **Structured parameters:** Structured parameters are defined in CAI3G generic interface specification. When assigning structured parameters, make sure that the recurrence is compatible between the target interface and the source interface.
- **Sub-MOs:** when assigning a sub-MO, make sure that the source sub-MO has identical structure as the task sub-MO. Sub-MO operations are automatically passed through from the source to the task. For example,

for a task of set operation type, the supported operations are: adding a sub-MO, modifying an existing sub-MO, and removing an existing sub-MO.

- Empty element: in runtime, an empty element in a source (either northbound request or response of another task) is considered a valid value. Thus it is transferred to the target as is. This means that empty value is used in the target even if a value is specified in **Source value with default**.

9.3.1 Slide-in Panel

When using the parameter assignment function in the parameters tab, a slide-in panel is displayed on the page.

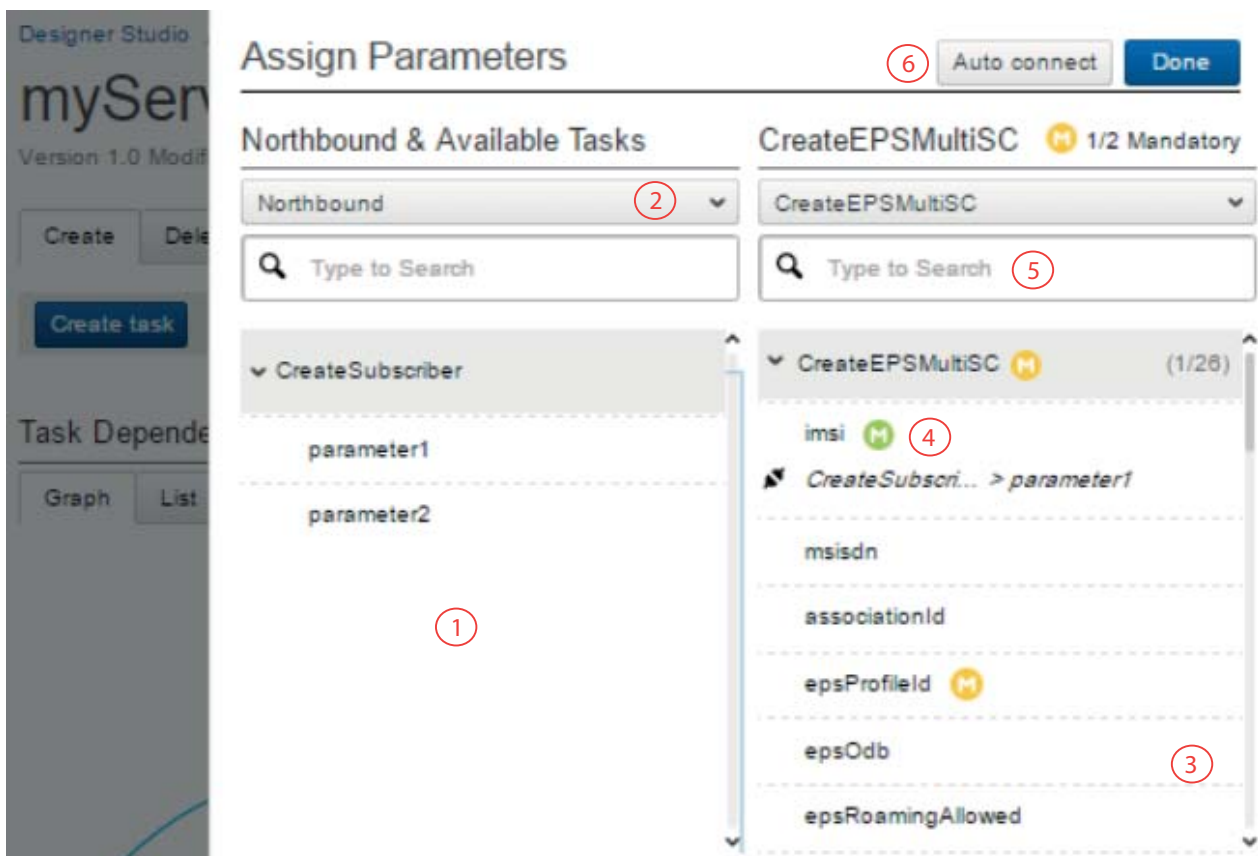


Figure 11 Slide-in Panel

In Figure 11:

Area **1** is the **source parameter column** that lists the Northbound Interface and tasks that have responses defined. The source selector (area **2**) is by default set for northbound. The parameters specified in the Northbound Interface schemas are shown in this column.

Area **3** is the current **destination parameter column**. The parameters specified in the schemas of the selected task or northbound are listed in this area.



Mandatory parameters are marked with a **M** (area 4). The color codes are:

- **M** – Indicates that this mandatory parameter is assigned.
- **M** – Indicates that this mandatory parameter has not been assigned.
- **M** – Indicates that this parameter can be assigned but not have to in the current setup. Because the interface schemer defines multiple mandatory parameter choices, and one of other choices is assigned, as shown in Figure 12.

Use the link (for example, [or one of 2 others](#)) to locate other choices quickly.

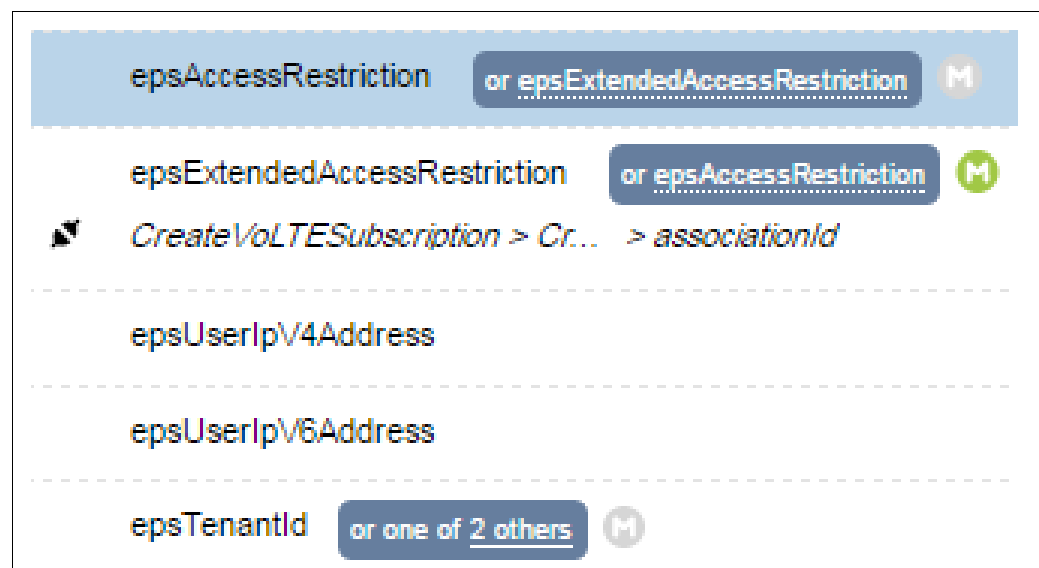


Figure 12 Mandatory Parameter Choices

Note: **M** can change to **M** dynamically if current setup changes. For example, according to the interface schemer, several sibling choices must be chosen together. If one of those parameters are assigned, the other siblings are changed to **M**.

The minimum requirement for a task to work is that **No** mandatory parameter is with a **M**.

Search function (area 5) is available for both source parameter column and destination parameter column. When entering text here, parameters that match the text and elements containing parameters match the text are shown. This function is case-sensitive.

The Auto-assignment function (area 6) is activated when an element is selected in both the source parameter column and the destination parameter column.

This function assigns the parameters in the selected element with parameters with identical name on the other column.

Note: Parameters, having same name within the element that is auto assigned, must be unmapped and remapped manually. Ericsson recommends verifying the auto mapped parameters manually.

9.3.2 Parameter Assignment Setting

The following are available settings for parameter assignments:

- **Source value:** the default setting for parameter assignment, meaning that task parameter is to use the value of the source parameter. The default setting applies for parameters assigned manually or by Auto-assignment.
- **Source value with default:** use the value of source parameter if the parameter exists in the source, otherwise use the value defined in the task parameter in the GUI.
- **Fixed value:** a constant value that is always to be used for the specified southbound parameter. Therefore, an assignment to a source is not possible. Defining a fixed value for a task parameter disables existing mapping towards source parameter.

Note: For the option **Fixed value**, condition must not be set on a multiple value parameter or on a parameter belonging to a structured parameter.

- **Transformation:** a transformed value of the specified source parameter by executing a transformation expression. For example, by executing the `CONCAT(parameter1, "@ericsson.com")` function, a string text `@ericsson.com` is joined at the end of the source parameter value. Available transformation functions are described in Section 16.2 on page 69.

Note: **Transformation** cannot be combined with parameter assignment conditions.



Designer Studio
Help

Designer Studio / myServiceModel / Transformation

Transformation

createSubscription > amsisdn > bs

Model: myServiceModel | Version: 1.0 | Task: GetHLRSubscription

<
Save

Expression 1

Test expression 4
?

✓
Test was successful! Result: meat lover
×

Sources 2

+ Add new source

Source Name	Task Name	Path	Test Value 3	
parameter1	Northbound	CreateSubscriber > parameter1	<input type="text" value="hamburger"/>	Delete source

Tables 5

+ Add new table

▼ fruitPrice

 Delete table

Figure 13 Transformation Configuration

In Figure 13:

Area **1** is used for specifying an expression of a single or a composite function, which is used to transform the selected parameter. For syntax and examples of available transformation functions, click .

Area **2** is used for specifying sources of the parameter to be used in a transformation expression.

Note: If no source is specified, the transformation is not to be performed. A **Fixed value** can be used instead in such a case.

Area **3** is used for specifying a test value of a source parameter, to verify a transformation expression by clicking the button **4**.

Area **5** is used for managing tables used for the LOOKUP transformation function.



9.3.3 Parameter Assignment Condition

One or more conditions can be specified for a parameter. Conditions must be fulfilled to include this parameter assignment.

Note: Use conditions only when there is business or NE technical needs. Unnecessary condition prevents Dynamic Activation from computing optimized execution plan.

A condition can be based on criteria in the northbound request or another task for this operation type.

The available condition alternatives are:

- **Has parameter:** this alternative can be set on the northbound request or another task with defined response.
- **Does not have parameter:** this alternative can be set on the northbound request or another task with defined response.
- **Has parameter with value:** this alternative can be set on the northbound request or another task with defined response. User provides a specific value for the selected source parameter. Parameter value provided in the field can be expressed as regular expression constructed according to Java regex syntax.

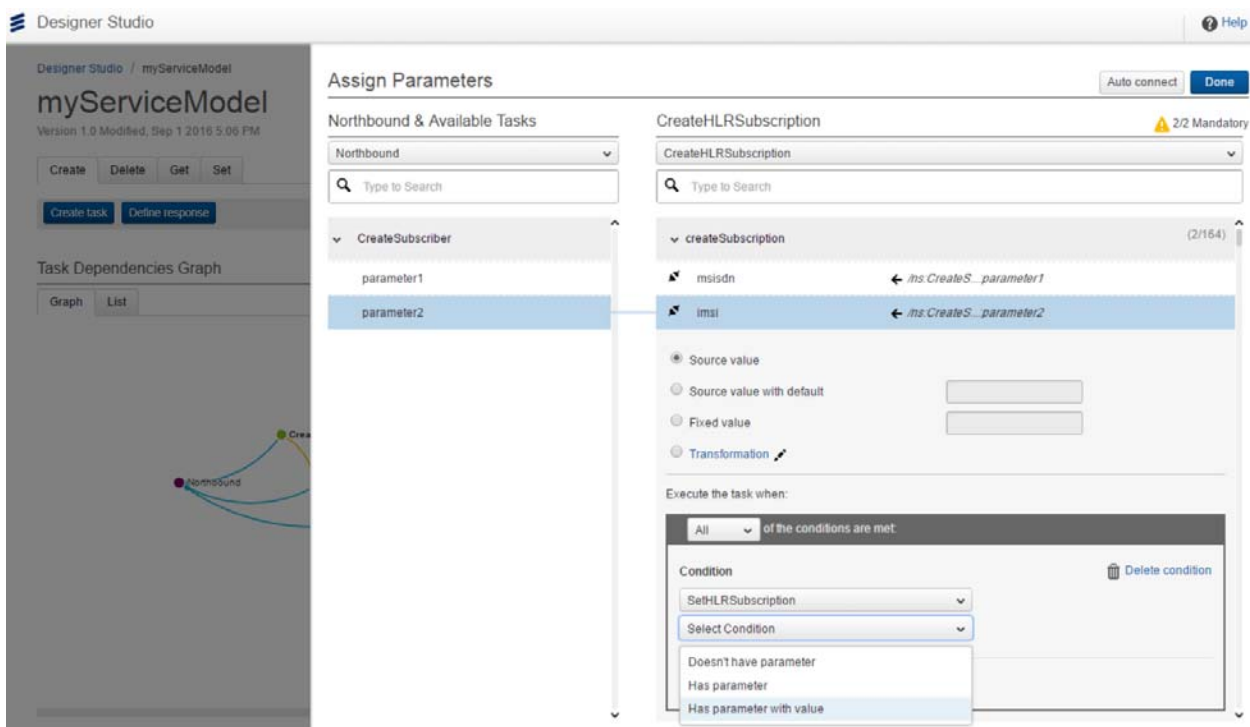


Figure 14 Parameter Assignment Conditions



Attention!

When configuring condition for mandatory parameter, if the condition is not met in runtime, the task fails because of missing mandatory parameter.

It is possible to specify the relation between the multiple conditions in a flexible manner by using condition groups. The **Show boolean expression** function shows the aggregated relation of all conditions. Figure 15 shows an example of flexible combination of conditions.

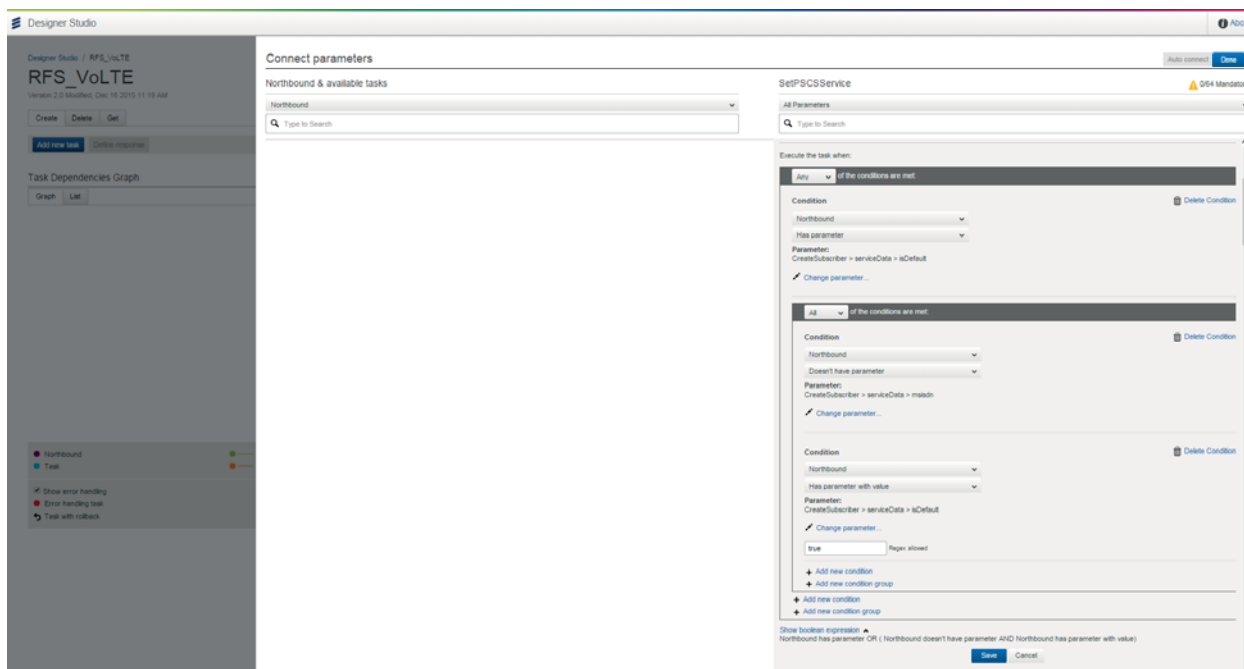


Figure 15 Flexible Combination of Conditions

9.4 Define Response for a Task

It is possible to define response for Create, Set, Get, and Delete operations. Figure 16 shows an example in the **Create** container.

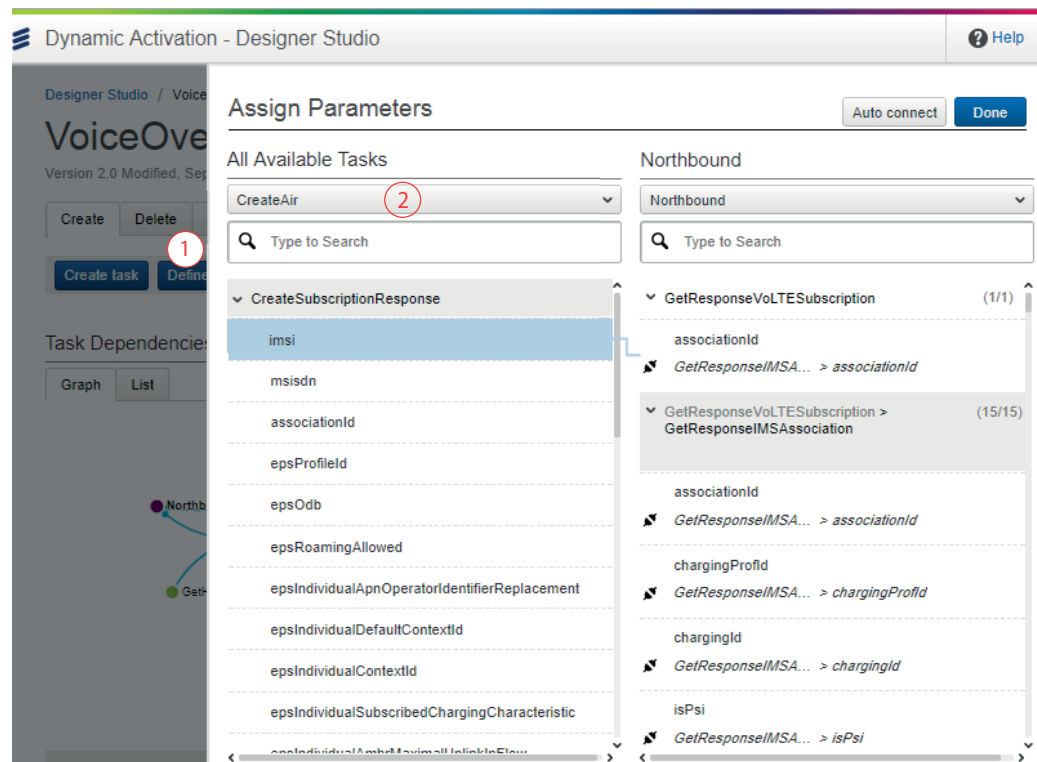


Figure 16 Define Response for a Task

In the **Create/Delete/Get/Set** container:

- The **Define response** (Area 1) is enabled when the northbound interface and at least one task contain `response` element for the respective operation in the interface WSDL file.
- All tasks with response defined in the schemas are listed in area 2. Users can configure the response mapping in the same way of assigning parameters for a task.

9.5 Define Condition for a Task

One or more conditions can be defined for a task. The conditions of a task regulate the execution plan of a given northbound request. All tasks defined for a given operation type are to be activated for the northbound request. A task, however, is only to be executed if and when the defined conditions are met.

Note: Use conditions only when there is business or NE technical needs. Unnecessary condition prevents Dynamic Activation from computing optimized execution plan.

A condition can be based on criteria in the northbound request or another task in the same task container.



myServiceModel

Version 1.0 Modified, Sep 1 2016 5:06 PM

Create

Delete

Get

Set

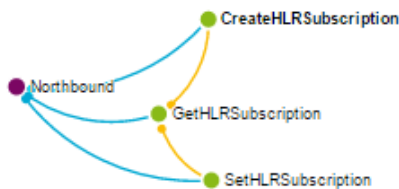
Create task

Define response

Task Dependencies Graph

Graph

List



● Northbound — Parameter dependency
● Task — Condition dependency

☒ Show error handling

● Error handling task

↺ Task with rollback

CreateHLRSubscription

[Actions](#)

Parameters

Conditions

Error Handling

Properties

Execute the task when:

All

of the conditions are met:

Condition

Delete condition

GetHLRSubscription

Has error code

Doesn't have parameter

Has error code

Has parameter

Has parameter with value

Is successfully executed

☒ Show boolean expression

Save

Figure 17 Define Condition for a Task

- **Doesn't have parameter:** this alternative can be set on the northbound request or another task with defined response.
- **Has error code:** this alternative can be set on another task for this operation type. The alternative can be used to handle expected error responses during execution. For example, for a business rule **VoLTE Auto provisioning**, the condition for a task `CreateHLRSubscription` can be defined as **Has error code 13** (IMSI is not defined) to the task `GetHLRSubscription` response – which means to check whether an HLR subscription exists before creating it.

When this alternative is selected, users can enter an acceptable error code, or select one from the drop-down list.

The acceptable-error-codes in the drop-down list can be configured in the **Error Handling** tab of the other task (the one the current task depends on). See Section 9.6.1 on page 41.

Note: If the selected acceptable-error-code is removed from the other task, the condition is automatically deleted from the current task.

- **Has parameter:** this alternative can be set on the northbound request or another task with defined response. For example, for the business rule *Upgrade IMS subscription to VoLTE*, the condition for task `SetIMSSubscription` can be defined as task `GetIMSSubscription` response having parameter `AssocID` – which means the subscriber is an existing IMS subscriber.
- **Has parameter with value:** this alternative can be set on the northbound request or another task with defined response. User provides a specific value for the selected source parameter. For example, for business rule *If a subscriber is defined as a certain type in HLR*, it is allowed to add SMS service to the subscriber. The condition for task `CreateSMSService` can be defined as task `GetHLRSub` has parameter `SType` with value 5. Parameter value provided in the field can be expressed as regular expression constructed according to Java regex syntax.
- **Is successfully executed:** this alternative is set on another task for this operation type. This alternative is useful when the execution order of tasks is of importance for the execution of the service model operation.

Note: In case task_B uses this alternative on task_A, if task_A is not executed owing to an unfilled condition in runtime, the **Is successfully executed** is still fulfilled for task_B. To prevent task_B from being executed while task_A is not, configure the same conditions of task_A for task_B.

It is possible to specify the relation between the multiple conditions in a flexible manner by using condition groups. The **Show boolean expression** function shows the aggregated relation of all conditions. For details, see Figure 15.



9.6 Define Error Handling for a Task

Two error handling methods can be defined for a task:

- **Accept error codes** – Enable continuing execution when expected runtime errors occur on the task.
- **Enable rollbacks** – Enable the default rollback behavior, which is implemented in the Service Model Engine in Dynamic Activation system, when error occurs on other tasks.

myServiceModel

Version 1.0 Modified, Sep 1 2016 5:06 PM

The screenshot shows the 'myServiceModel' web interface. At the top, there are buttons for 'Create', 'Delete', 'Get', and 'Set'. Below these are 'Create task' and 'Define response' buttons. The main area is divided into two panels. The left panel, titled 'Task Dependencies Graph', shows a dependency graph with nodes for 'Northbound', 'CreateHLRSubscription', 'GetHLRSubscription', and 'SetHLRSubscription'. It includes a legend for 'Northbound' (purple dot), 'Task' (green dot), 'Parameter dependency' (blue line), and 'Condition dependency' (yellow line). The right panel, titled 'GetHLRSubscription', has tabs for 'Parameters', 'Conditions', 'Error Handling', and 'Properties'. The 'Error Handling' tab is active, showing options to 'Accept error codes' (checked) and 'Enable rollback' (unchecked). Under 'Accept error codes', there is a list of error codes: 10, 12, and 13. Below the list is an 'Error code' input field with a plus sign button.

Figure 18 Define Error Handling

9.6.1 Accept Error Codes

When **Accept error codes** is enabled for a task, users can configure an acceptable-error-code list for the task.

- At configuration time, user can use such an error code to define other tasks conditions. See **Has error code** in Section 9.5 on page 38.
- At runtime, the Service Model continues execution if such an error code is included in the task response.

- Wildcard “*” can be used to set multiple acceptable error codes for the task. For example:

- Use “*” to set all error codes as acceptable.

- Use “13*” to set error codes starting with "13" as acceptable.

- If no acceptable-error-code list is configured, the **Accept error codes** become unchecked when the task is selected again.
- Before removing an acceptable-error-code, ensure that no other tasks use the deleted code as a **Has error code** condition.

If such a task exists, the relevant condition is deleted from that task automatically.

Note: When checking tasks that have condition dependency to the current task, ensure that **Show error handling tasks** is selected. Otherwise hidden error handling tasks can be missed.

A use case of this function, is to enable **Accept error codes** on a low-priority task (for example, `CreateSMSService`). If certain acceptable errors or any errors occur on the task at runtime, the Service Model Engine continues execution. For example, a subscription can be ready to use when core network is provisioned successfully, even the additional SMS service is not provisioned.

9.6.2

Enable Rollback

If rollback is enabled on a task, the Service Model Engine triggers rollback of the task when both of the following conditions are met:

- The task is executed successfully, but another task fails.
- The failure task response does not include any acceptable error codes, which means either of the following:
 - The **Accept error codes** is not enabled on the failure task.
 - The **Accept error codes** is enabled, but the errors that caused the failure are not in the acceptable-error-code list.

The default rollback behavior is:



- For create operation: delete the successfully created data
- For set operation: revert to the previous value of parameters. As prerequisites, the MO, which the set task operates on, must:
 - Support a `Get` operation.
 - Have a `GetResponse` defined in the schema.

When rollback is enabled for the set task, the get operation is automatically executed and the previous values are obtained from the `GetResponse`.

If the set task is assigned with sub-MOs, all changes on sub-MO instances are also rolled back.

Note: Default rollback does not support tasks that:

- Remove all assigned sub-MO entirely.
- Are assigned with nested sub-MOs or include structure elements.

If in such cases, an error handling task must be created to handle rolling back. See Section 9.7 on page 43.

Attention!

To support sub-MO rollback for a service model that is created in a Designer Studio earlier than the Dynamic Activation 1 release, you must export the service from the old Designer Studio, and then import it to the latest Designer Studio.

- For delete operation, rollback is not implemented. In order for northbound system to resend a Delete request, loose error handling must be implemented in the southbound service. For more information about loose error handling, refer to *Function Specification Resource Activation*, Reference [3].

Note: Ensure that the identifies are properly assigned before enabling rollback.

9.7 Error Handling Task

Error handling tasks are to be executed when error occurs in other tasks. This is by defining condition error response from other tasks.

Multiple conditions can be defined for an error handling task. In such a case, the error handling task is to be executed when any of the defined errors occurs.



myServiceModel

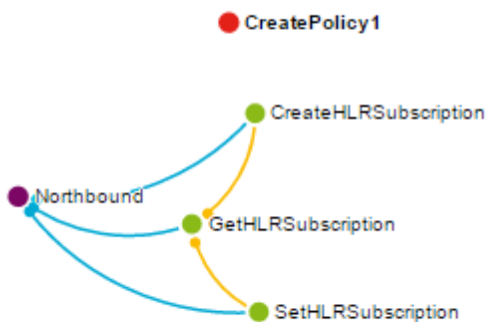
Version 1.0 Modified, Sep 1 2016 5:06 PM

Create Delete Get Set

Create task Define response

Task Dependencies Graph

Graph List



Northbound Task
Parameter dependency Condition dependency

☒ Show error handling
☐ Error handling task
☐ Task with rollback

CreatePolicy1

Actions

Parameters Conditions Error Handling Properties

Execute the task when:

All of the conditions are met:

Condition

Delete condition

CreateHLRSubscription

Has error code

Return error code:

ex 1100*, 11*00



+ Add new condition

+ Add new condition group

▼ Show boolean expression

Save

Figure 19 Error Handling Task

When a task is identified as error handling task at creation, this task differs from a normal task in the following areas:

- Available condition types are various:
 - Available conditions to Northbound:
 - Doesn't have parameter
 - Has parameter
 - Has parameter with value
 - Available conditions to another task:
 - Doesn't have parameter
 - Has error code



- `Has parameter`
- `Has parameter with value`
- `Is executed successfully`
- Available condition to another error handling task: `After`, see Section 9.7.1 on page 45.
- **Accept error codes** is not applicable.
- Rollback is not applicable.
- Error handling tasks have another color in the dependencies graph.
- Error handling tasks can be hidden from the dependencies graph.

It is possible to specify the relation between the multiple conditions in a flexible manner by using condition groups. The **Show boolean expression** function shows the aggregated relation of all conditions. For details, see Figure 15.

9.7.1 Condition After

An error handling task can use the condition `After` based on another error handling task.

The `After` alternative is only available for root level conditions having **All** operator. It is not available for **Any** operator or a child level condition inside a condition group. See Figure 20.

CreatePolicy1

Parameters

Conditions

Error Handling

Properties

Execute the task when:

All of the conditions are met: root level

Condition

CreatePolicy2

Select Condition

After

Delete condition

All of the conditions are met: child level

Condition

SetHLRSubscription

Is successfully executed

Delete condition

+ Add new condition

+ Add new condition group

+ Add new condition

+ Add new condition group

Figure 20 Condition After

As an example in Figure 20, for the current task `CreatePolicy1`:

- The `After` condition is fulfilled when either of the following occurs on the other error handling task `CreatePolicy2`:
 - The `CreatePolicy2` is executed successfully.
 - The `CreatePolicy2` is not executed because of an unfulfilled condition.
- The `After` condition is not fulfilled if `CreatePolicy2` fails.



In a use case to avoid executing `CreatePolicy1` when `CreatePolicy2` is not executed because of an unfulfilled condition, the same conditions of `CreatePolicy2` must be defined on the `CreatePolicy1`.

9.8 Update of Northbound Interface for a Service Model

It is possible to update Northbound Interface for an existing service model. Changes in the Northbound Interface can affect parameter assignments, conditions, or both in the existing service model. Thus Designer Studio automatically creates an instance of the service model. The user is asked to provide a new version number for this instance with the updated Northbound Interface. The affected parameter assignments, conditions, or both are removed in the new version. Ericsson recommends controlling all the removed items in the downloaded file and modify the updated service model manually if necessary.

The following types of changes are supported:

- Add parameters/elements: no impact on the existing service model. If the add parameters/elements are mandatory, they must be assigned for the service model to work.
- Remove parameters/elements: parameter assignments and conditions related to the removed parameters/elements are invalid and are removed in the updated service model.
- Rename parameters/elements: parameter assignments and conditions related to the renamed parameters/elements are invalid and are removed in the updated service model.
- Parameters/elements changed from mandatory to optional: if the task parameter, to which the changed parameter is assigned, is a mandatory parameter, error can occur in runtime.
- Parameters/elements changed from optional to mandatory: no impact on the existing service model.
- Rename of one namespace in the interface: Designer Studio updates the changed namespace in the updated service model.
- Change of data type for parameters/elements: Designer Studio cannot detect such changes. Consult the southbound interface schema to make sure that the assigned task parameter has compatible data type.

If the changes include renaming of more than one namespace in the Northbound Interface wsdl/xsd file, delete the service model and create a new one instead.

When prompted to upload schemas, make sure that all the documents for the Northbound Interfaces are uploaded. This means that the WSDL file and all the XSD files the WSDL file depends on.



Service Models

[Update southbound interfaces](#)[Create service model](#)[Import service models](#)

All Service Models

Name	Version	Modified
myL3VPN	4	Sep 1 2016 5:21 PM
VoiceOve...	2.0	Sep 1 2016 5:31 PM
myServic...	1.0	Sep 1 2016 5:06 PM

myL3VPN

Version: 4 Modified Thursday, September 1, 2016 5:21 PM

[Edit](#)[Actions](#)

- Export
- Update northbound schema
- Rename
- Duplicate
- Delete

Figure 21 Update Northbound Interface

9.9 Service Model Examples

The examples described in this chapter are included in the zip file in the link in the step list below. Do the following to import the examples in Designer Studio:

Attention!

The service models embedded below are examples and to be used strictly for educational purpose. Ericsson shall have no liability for any error or damage of any kind resulting from the use of these examples.

1. Close the Designer Studio GUI web browser.
2. Shut down Designer Studio.
3. Go to <DS_HOME>, relocate the `workspace` folder to another location for later use.
4. Save the zip file, [Service Model Examples.zip](#) to a temporary location.
5. Unpack the zip file.
6. Place the `workspace` folder under <DS_HOME>.
7. Start Designer Studio
8. Access the Designer Studio through a web browser and the FTTH and VoiceOverLTE service model is available in the GUI.

After the learning period, remove the `workspace` folder from <DS_HOME> and restore the original one relocated from Step 3 above.



9.9.1

Example FTTH

This service model example implements a Fiber to The Home (FTTH) service. Figure 22 shows the topology.

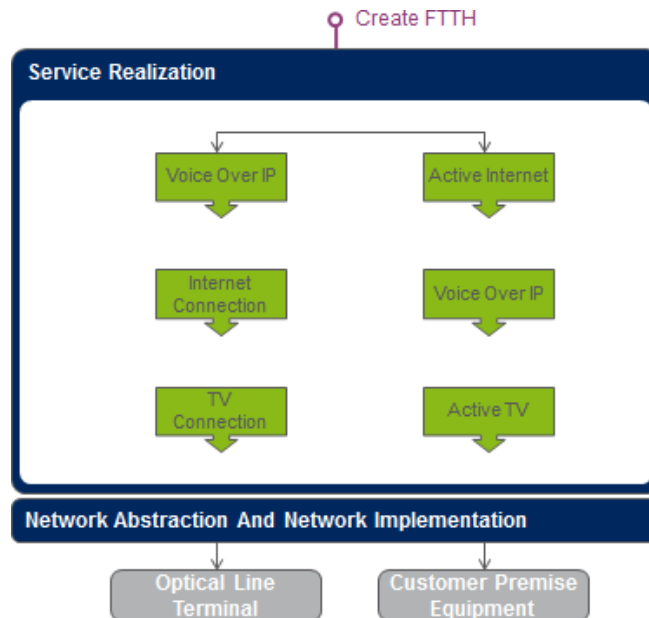


Figure 22 FTTH Service

Figure 23 shows the Create operation of the FTTH service model.

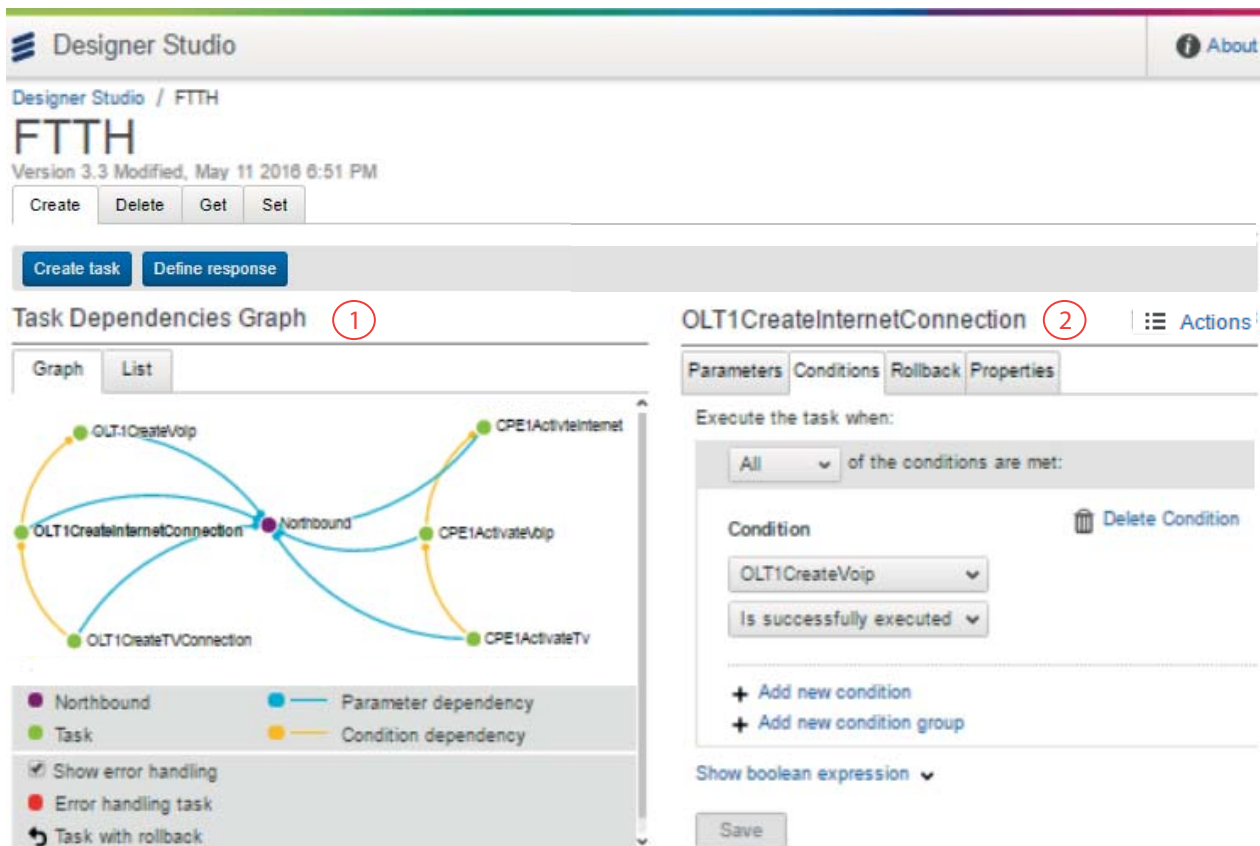


Figure 23 FTTH Example

The graph in area 1 visualizes:

- Two task tracks are configured for realizing the FTTH service.
 - One track is to configure the Optical Line Terminal (OLT).
 - One track is to configure the Customer-Premises Equipment (CPE).
 - Both tracks are to be performed at the same time.
- Within each track:
 - All task parameters are mapped to the Northbound Interface, which means there is no parameter dependency between tasks.
 - Tasks are executed in sequence, which is defined by task conditions.

Area 2 shows the condition of the `OLT1CreateInternetConnection` task as an example.



9.9.2 Example Voice over LTE

Note: This example is for educational purpose. For a complete VoLTE provisioning example for commercial purpose, refer to *VoLTE Provisioning Customer Adaptation Guide*, Reference [10].

This service model example implements a simplified version of the Voice Over LTE (VoLTE) solution as illustrated in Figure 24.

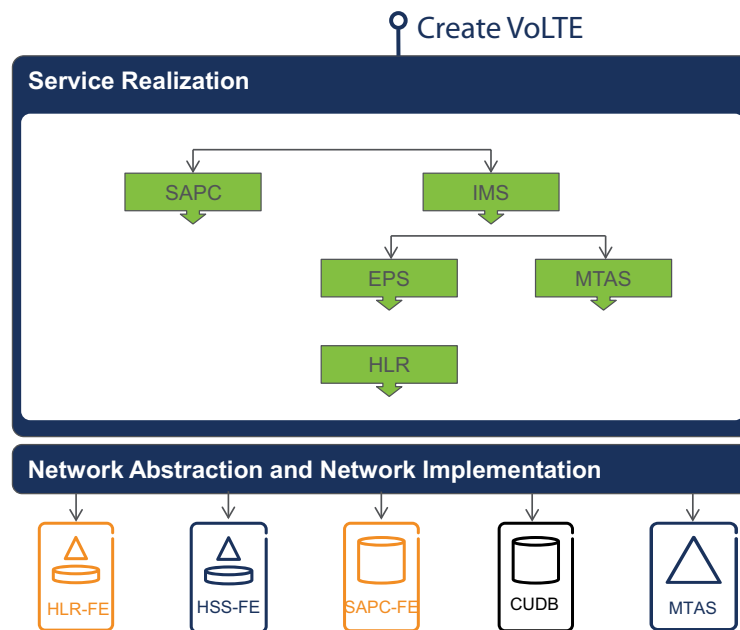


Figure 24 VoLTE Service

Figure 25 illustrates the Create operation of the VoLTE service model.

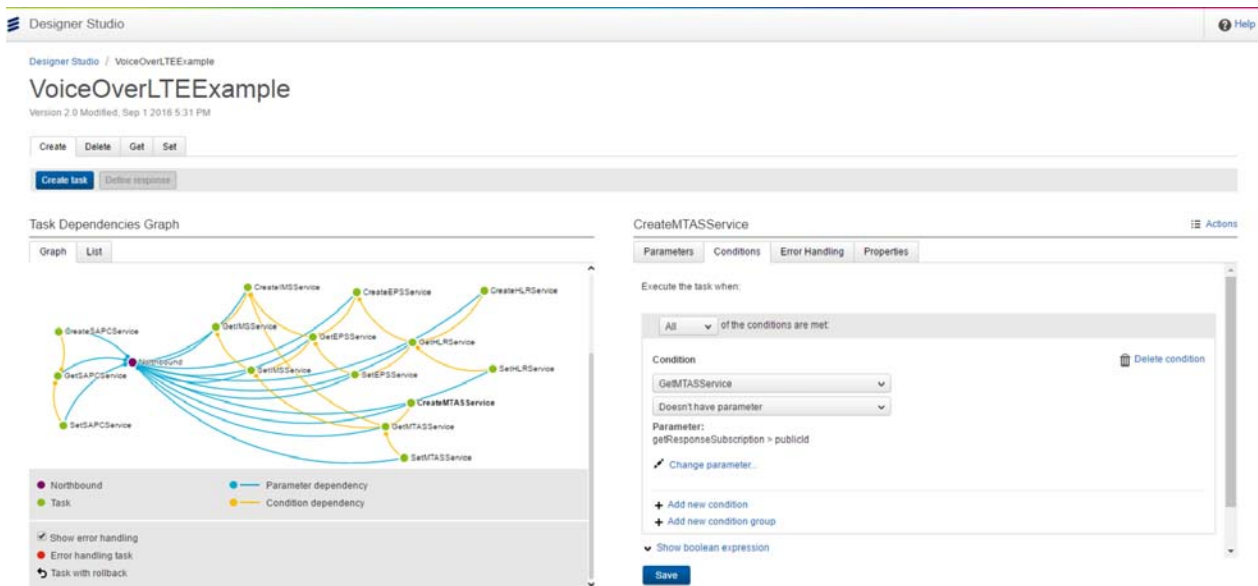


Figure 25 VoLTE Example

The graph to the left visualizes a northbound request:

- Fetch SAPC data from CUDB
 - If subscription is not found, create the subscription according to data from the request.
 - If subscription exists, modify the subscription with data from the request.
- While fetching SAPC data, the Get IMS Association task is executed at the same time
 - If subscription is not found, create the subscription according to data from the request.
 - If subscription exists, modify the subscription with data from the request.
- After IMS Association is up to date, fetch EPS subscription from CUDB
 - If subscription is not found, create the subscription according to data from the request.
 - If subscription exists, modify the subscription with data from the request.
- While fetching EPS data, the Get MTAS subscription task is executed at the same time
 - If subscription is not found, create the subscription according to data from the request.
 - If subscription exists, modify the subscription with data from the request.
- After EPS subscription is up to date, fetch HLR data from CUDB



- If subscription is not found, create the subscription according to data from the request.
- If subscription exists, modify the subscription with data from the request.

This construction is built with consideration to the CUDB data model and the application dependency between HSS and MTAS.

This example also illustrates multiple conditions for a task, as shown in the area to the right in Figure 25.

The graph of Get operation in area 1 in Figure 26 illustrates the scenario for parameter dependency between tasks. As shown in area 2, the IMSI for GetEPSService has the source from another task GetIMSService. This parameter dependency specifies implicitly that task GetEPSService is to be executed after GetIMSService is completed successfully.

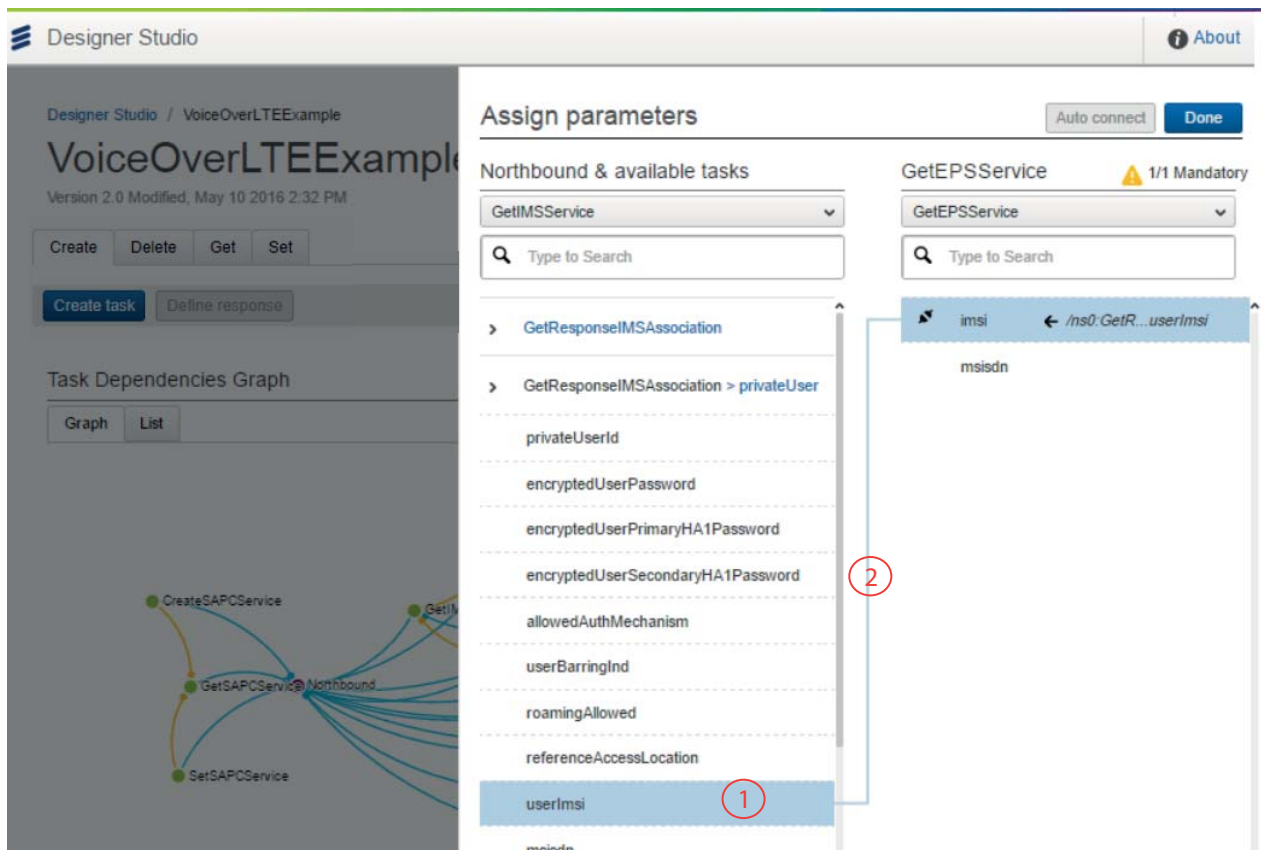


Figure 26 Get Operation





10 Optional Operations

More optional operations can be performed in the Designer Studio **Service Models** page.

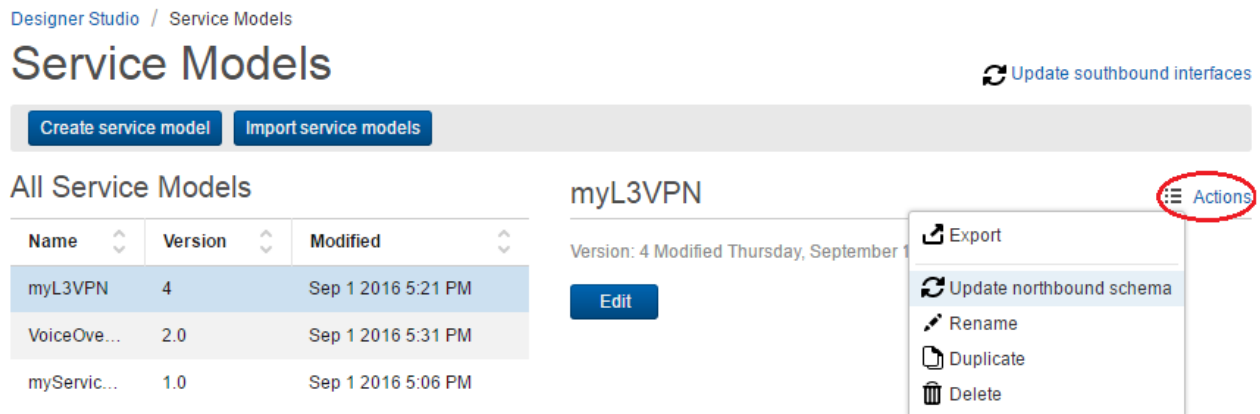


Figure 27 Optional Operations

10.1 Export a Service Model

A service model can be exported for manually deployment or correlation purpose.

To export a service model:

1. Go to **Service Models** page.
2. Select the service model to export.
3. Use the **Actions>Export** link. The service model is downloaded to the local download directory.

10.2 Copy, Rename, or Delete a Service Model

1. Go to **Service Models** page.
2. Select a service model in the list.
3. Click the **Actions** and chose one of the following links to proceed.
 - **Duplicate** (copy a service model)
 - **Rename**
 - **Delete**





11 Deploying and Undeploying a Service Model

This section describes how to deploy and undeploy a service model to the Dynamic Activation system.

A service model becomes a JDV when it is deployed on a Dynamic Activation system.

11.1 Prerequisites

Before deploying or undeploying a service model, ensure that the following conditions are met:

- Service model can only be deployed to (or undeployed from) the target runtime system by an administrative user.
- Service model created in the Designer Studio must be run on a compatible Dynamic Activation system. For information on version dependency, see Section 3.2.2.3 on page 11.
- Make sure that there is no other JDV in the target Dynamic Activation system that uses the same MO name and MO namespace.

Scenarios where the same MO name or MO namespace could exist and solutions are given as examples:

- A previous version of the same service model is already deployed on the system. If such a JDV exists, undeploy previous version of the service model as described in Section 11.3 on page 58.

For instructions on how to check existing JDVs, refer to *System Administrators Guide for Native Deployment*, Reference [8], or *System Administrators Guide for Virtual and Cloud Deployment*, Reference [9].

- The Northbound Interface, prepared in Section 8 on page 27 does not have a unique MO name and MO namespace. This means that another, already deployed JDV, uses the same name and namespace. In this case, an error is returned upon sending a provisioning request.

11.2 Deploy a Service Model

To deploy a service model, do as follows:



1. In Designer Studio, select the service model to be deployed and use the **Export** button to export the service model. The service model is downloaded to a local download directory.
2. Transfer the exported service model file, `<service_model_name>.tar.gz`, to `/home/bootloader/CArepository` on SC1 (Native) or node-1 (Virtual), of the Dynamic Activation system as an administrative user. For example, by using SFTP.
3. Change the owner and the group of the `<service_model_name>.tar.gz` file.

```
# chown actadm:activation /home/bootloader/CArepository  
/<service_model_name>.tar.gz
```

4. Run the following command to deploy the service model:

```
$ bootloader.py submodule deploy --name <service  
model file name>.tar.gz --type servicemodel --parent  
activation-orchestration-module --host all
```

11.3 Undeploy a Service Model

Service model undeployment is performed manually in the Dynamic Activation system.

To undeploy a service model, do as follows:

1. Connect to SC1 (Native) or node-1 (Virtual) of Dynamic Activation system, and log in as an administrative user.
2. Run the following command to remove a service model:

```
$ bootloader.py submodule undeploy --name <service model  
file name>.tar.gz --parent activation-orchestration-mo  
dule --host all
```




12 Service Model Error Response

Error response for a northbound request follows the CAI3G common practices. For CAI3G fault codes, refer to *CAI3G Implementation*, Reference [6]. This session describes the subordinate CAI3G errors that can be appeared in the error code element in the error message.

Table 2 Error Codes

Error Code	Error Message
1001	Invalid resource ID
1002	Invalid XPath
1003	Unrecognized namespace. No data view associated.
1004	Access denied. Invalid principal or credentials.
1005	Not authorized to perform current operation on selected data view.
1006	Invalid parameter
1009	Unsupported operation
1010	Invalid session ID
1011	Target is not found
1095	Communication error while interacting with a Network Element
1096	Time-out expired during wait for answer from Network Element
1097	Failure during processing of the request
1098	Could not process request because of resource limitation
1099	System error
1101	External error
1103	License error
2001	Format error
2002	Unreasonable value
2004	Database update error
2005	Storage shortage in log data file
50009	OPERATION FAILED, ROLLBACK WAS UNSUCCESSFUL
50010	OPERATION FAILED, ROLLBACK HAS BEEN PERFORMED SUCCESSFULLY

There are examples of the typical error response:



```
-----  
o Request failed rollback successful  
-----
```

```
<S:Body>  
<ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"  
  xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">  
  <faultcode>ns2:Server</faultcode>  
  <faultstring>This is a server fault</faultstring>  
  <detail>  
    <Cai3gFault:Cai3gFault xmlns="http://schemas.ericsson.com/cai3g1.2/"  
      xmlns:Cai3gFault="http://schemas.ericsson.com/cai3g1.2/">  
      <faultcode>4006</faultcode>  
      <faultreason>  
        <reasonText>External error.</reasonText>  
      </faultreason>  
      <faultrole>MF</faultrole>  
      <details>  
        <PGFault:PGFault  
          xmlns="http://schemas.ericsson.com/pg/1.0/"  
          xmlns:PGFault="http://schemas.ericsson.com/pg/1.0/">  
          <errorcode>50010</errorcode>  
          <errormessage>OPERATION FAILED, ROLLBACK HAS  
            BEEN PERFORMED SUCCESSFULLY</errormessage>  
          <errordetails>Successful rollback of request:  
            {http://schemas.ericsson.com/ma/CA/VoLTESubscriber/}  
            CreateVoLTESubscription in model: DesignStudio version: 3.0.4  
            Executions fail information:  
            Task with name: createIMSAssoc failed its execution with internal error c  
          </errordetails>  
        </PGFault:PGFault>  
      </details>  
    </Cai3gFault:Cai3gFault>  
  </detail>  
</ns2:Fault>  
</S:Body>
```

```
-----  
o Request failed rollback failed  
-----
```

```
<S:Body>  
<ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/  
envelope/" xmlns:ns3="http://www.w3.org/2003/05/  
soap-envelope">  
  <faultcode>ns2:Server</faultcode>  
  <faultstring>This is a server fault</faultstring>  
  <detail>  
    <Cai3gFault:Cai3gFault xmlns="http://schemas.  
ericsson.com/cai3g1.2/" xmlns:Cai3gFault=  
"http://schemas.ericsson.com/cai3g1.2/">  
      <faultcode>4006</faultcode>  
      <faultreason>  
        <reasonText>External error.</reasonText>  
      </faultreason>  
      <faultrole>MF</faultrole>  
      <details>  
        <PGFault:PGFault xmlns=  
          "http://schemas.ericsson.com/pg/1.0/"  
          xmlns:PGFault="http://schemas.ericsson.com/pg/1.0/  
ericsson.com/ema/provisioning/">  
          <errorcode>50009</errorcode>  
          <errormessage>OPERATION FAILED,  
            ROLLBACK WAS UNSUCCESSFUL</errormessage>  
          <errordetails>Failed rollback of request:  
            {http://schemas.ericsson.com/ma/CA/VoLTESubscriberError/}  
            CreateVoLTESubscription in model: DesignStudio version: 3.0.4  
          </errordetails>  
        </PGFault:PGFault>  
      </details>  
    </Cai3gFault:Cai3gFault>  
  </detail>  
</ns2:Fault>  
</S:Body>
```

Possible issues:

Task with name: createSAPCsub had its rollback logic suppressed
due to other tasks failing their rollback.
Task with name: deleteSAPCsub had its rollback logic suppressed
due to other tasks failing their rollback.
Task with name: createHLRsub failed its execution with internal
error code: 10001.
Task with name: ReCreateSAPCsub had its rollback logic suppressed
due to other tasks failing their rollback.



Task with name: ReCreateIMSAssoc failed its rollback execution
with internal error code: 14001.

```
</errordetails>
      </PGFault:PGFault>
    </details>
  </Cai3gFault:Cai3gFault>
</detail>
</ns2:Fault>
</S:Body>
```

o Request failed (rollback is not enabled for any tasks in
the service model)

```
<S:Body>
<ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
  <faultcode>ns2:Server</faultcode>
  <faultstring>This is a server fault</faultstring>
  <detail>
    <Cai3gFault:Cai3gFault xmlns="http://schemas.ericsson.com/cai3g1.2/"
      xmlns:Cai3gFault="http://schemas.ericsson.com/cai3g1.2/">
      <faultcode>4006</faultcode>
      <faultreason>
        <reasonText>External error.</reasonText>
      </faultreason>
      <faultrole>MF</faultrole>
      <details>
        <PGFault:PGFault
          xmlns="http://schemas.ericsson.com/pg/1.0/"
          xmlns:PGFault="http://schemas.ericsson.com/pg/1.0/">
          <errorcode>1101</errorcode>
          <errormessage>External error.</errormessage>
          <errordetails>Failed request:
            {http://schemas.ericsson.com/ma/CA/VoLTESubscriber/}
            CreateVoLTESubscription in model: DesignStudio version: 3.0.4
            Possible issues:
            Task with name: createIMSAssoc failed its execution with internal error
          </errordetails>
        </PGFault:PGFault>
      </details>
    </Cai3gFault:Cai3gFault>
  </detail>
</ns2:Fault>
</S:Body>
```

Example 1 Error Responses





13 Verifying a Service Model

1. Use a web service client test tool, for example Soap UI, and create a project based on your Service Model northbound WSDL file.
2. Send a LOGIN request towards the Dynamic Activation system, and retrieve the `sessionId` from the response message.
3. Generate a SOAP request that includes the retrieved `sessionId` and wanted parameter values.
4. Execute the SOAP request towards the Dynamic Activation system using endpoint `http://<VIP_Traffic_host_address>:8080/CAI3G1.2/services/CAI3G1.2`.
5. Verify the request execution results, which can be found through the **Log Management** GUI. For details of the Log Management GUI, refer to *User Guide for Resource Activation*, Reference [4].

If the service model is unnormal, see Section 14.1 on page 65 for troubleshooting.





14 Maintenance of Designer Studio

14.1 Troubleshooting A Service Model

The `consolidated.log` (or `/var/log/dve/activation-orchestration-module.log`) in the runtime system can be used to debug interaction between the service model and the business logic serving the tasks.

To troubleshoot a service model in runtime, enable trace for a short period (several minutes) on the service model:

1. Log on to one of the payload nodes.
2. Locate the `log4j.xml` file under `/usr/local/pgngn/activation-orchestration-module-<version>/config` directory.

3. Edit the `log4j.xml` file by adding

```
<category name="com.ericsson.mdv">
    <priority value="TRACE"/>
</category>
```

under the Limit categories section in the file.

4. Save the file. The change takes effect automatically after it is saved.

Attention!

Make sure to disable `trace` on live systems (operation level) after troubleshooting.

This is to avoid logging unnecessary data.

To disable the `trace` for service models:

1. Change the priority value that was added in Step 3 from `<priority value="TRACE"/>` to `<priority value="INFO"/>`.
2. Save the `log4j.xml` file. The change takes effect automatically after it is saved.



14.2 Clean up Temporary Files

If the Designer Studio GUI is not closed before shutting down of the application, garbage can remain in the temporary directory. Such files can be cleaned up manually.

Go to `<DS_SW>\temp` and, delete the files in the `temp` folder.



15 Help Center

Click **Help** to access the **Help Center**, as shown in Figure 28.

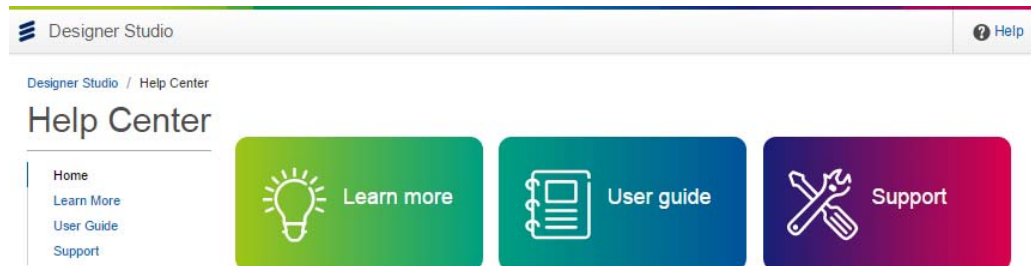


Figure 28 Help Center

15.1 Learn More

Here users can:

- Get a short description of what the designer studio is, and what it can do.
- View a tutorial video is available at the end of the page.

15.2 User Guide

Here users can access the User Guide.

15.3 Support

Here the user can download a data-collection package. This automatically packs the `workspace` folder, all logs, and a report including a problem description written by the user.





16 Appendix

16.1 Known Limitations and Solutions

This section describes walk-around solutions for the known limitations in Designer Studio.

- Update of a southbound interface does not support change of namespace.
Delete the tasks affected by the changed southbound interface, and create a new one.

16.2 Transformation Expression Functions

This section provides short descriptions on expression functions that can be used for parameter transformation.

More information on arguments of each function can be found in the GUI.

16.2.1 CONCAT

Syntax

```
CONCAT(text1, text2, [text3])
```

Description

Combines the text from multiple string values.

Example

Expression	<code>CONCAT("yourname", "@ericsson.com")</code>
Returns	<code>yourname@ericsson.com</code>

16.2.2 SUBSTR

Syntax

```
SUBSTR(text, start_num, [end_num])
```



Description

Extracts a subset of a text string, based on the start and end number of characters specified.

Note: SUBSTR always counts the leftmost character as 0 in a text.

Example

Expression	<code>SUBSTR("hamburger", 3, 7)</code>
Result	burg

16.2.3 CHOICE

Syntax

```
CHOICE(condition, value_if_true, value_if_false)
```

Description

Checks whether a condition is met, and returns one value if TRUE, and another value if FALSE.

Example

Expression	<code>CHOICE (MATCH("hamburger", ".*burg"), "meat lover", "vegan")</code>
Result	meat lover

16.2.4 LOOKUP

Syntax

```
LOOKUP(table_name, row_index, [column_index])
```

Description

Looks up a cell in a table based on specified criteria, and returns the value of the cell.

Note: If `column_index` is omitted, the first column-index is selected automatically.



Example

	5 kg	10 kg	20 kg
Apple	25	40	60
Banana	30	50	80
Pear	35	60	100

Figure 29 Example Table

Expression `LOOKUP(fruitPrice, "Banana", "10 kg")`
 Result 50

16.2.5 MATCH

Syntax

`MATCH(expression, pattern)`

Description

Determines whether a specific expression value matches a specified pattern, and returns:

- `TRUE` if it matches, or
- `FALSE` if it does not match.

Example

Expression `MATCH("hamburger", ".*burger")`
 Result `TRUE`

16.2.6 SKIP

Syntax

`SKIP()`



Description

This function is used as an argument of another function to skip the assignment of the parameter in the task.

Example

Expression	<code>CHOICE (MATCH ("TVM", "TVM"), SKIP (), "3")</code>
Result	The parameter is skipped in the task.



Reference List

- [1] *Product Overview*, 1550-CSH 109 628 Uen
- [2] *Library Overview*, 18/1553-CSH 109 628 Uen
- [3] *Function Specification Dynamic Activation Execution Environment*, 6/155 17-CSH 109 628 Uen
- [4] *User Guide for Resource Configuration*, 1/1553-CSH 109 628 Uen
- [5] *Generic CAI3G Interface 1.2*, 2/15519-FAY3020003 Uen
- [6] *CAI3G Implementation*, 26/155 19-CSH 109 628 Uen
- [7] *Customization - Architectural Overview*, 20/1553-CSH 109 628 Uen
- [8] *System Administrators Guide for Native Deployment*, 1/1543-CSH 109 628 Uen
- [9] *System Administrators Guide for Virtual and Cloud Deployment*, 3/1543-CSH 109 628 Uen
- [10] *VoLTE Provisioning Customer Adaptation Guide*, 13/1553-CSH 109 628 Uen