

# Customer Adaptation Development Guide for Resource Activation

## 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

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Purpose and Scope	1
1.2	Target Groups	1
1.3	Typographic Conventions	1
1.4	Prerequisites	1
<b>2</b>	<b>Dynamic Activation Overview</b>	<b>3</b>
2.1	Limitations	4
<b>3</b>	<b>JDV Overview</b>	<b>5</b>
3.1	Northbound Interfaces	6
3.1.1	CAI Interfaces	6
3.1.2	CAI3G Interface Files	7
3.2	JDV Configuration Files	7
3.2.1	javadataview-descriptor.xml	7
3.2.2	managedobjects-descriptor.xml	7
3.2.3	looseerror-descriptor.xml	8
3.3	Business Logic Files	8
3.3.1	JDV Factory File	8
3.3.2	JDV File	9
3.3.3	Provisioning BL Files	9
<b>4</b>	<b>JCA Overview</b>	<b>11</b>
<b>5</b>	<b>Connector Framework Overview</b>	<b>13</b>
<b>6</b>	<b>NE Management Overview</b>	<b>15</b>
<b>7</b>	<b>CA Application Types</b>	<b>17</b>
7.1	CUDB Layered Data NE Provisioning	18
7.1.1	CUDB Layered Data Provisioning Overview	18
7.1.2	Data Model	19
7.1.3	CA DLA Wizard and Base Projects	21
7.2	Monolithic Data NE Provisioning	22
7.2.1	Monolithic Data Provisioning Overview	22
7.2.2	Data Model	23
7.2.3	CA Monolithic Wizard and Design Base Projects	24
7.2.4	Migration of Existing JDVs	25
7.3	Cluster Strategy	28
7.3.1	Cluster Strategy CA Development Overview	28
7.4	Subscriber View Provisioning	29



7.4.1	Subscriber View Overview	29
7.4.2	Data Model	31
7.4.3	CA Subscriber View Wizard and Design Base Projects	33
7.5	Monolithic Data NE Provisioning for non UDC Solutions	33
7.5.1	Monolithic Data Provisioning Overview	33
7.5.2	CA Design Base Projects	34
7.5.3	Developing Monolithic Data NE Provisioning for non UDC solutions	34
<b>8</b>	<b>Preparing for CA Development Environment</b>	<b>37</b>
8.1	JDK	37
8.2	Maven	37
8.2.1	Maven Repository	37
8.3	IDE	38
8.3.1	Eclipse	38
8.4	Wizard	39
8.4.1	Installation of Wizard	39
8.5	Initializing Design Base Projects	39
8.5.1	Importing the Project	40
8.5.2	Renaming the JDV and JCA Projects (Optional)	40
<b>9</b>	<b>CA Development Workflow</b>	<b>43</b>
9.1	CA Development for CUDB Layered Data Provisioning	43
9.1.1	Preparations	43
9.1.2	Project Creation	43
9.1.3	Interface Specification	49
9.1.4	Define the JDV Access Control Model	51
9.1.5	Handling the JDV Business Operations	51
9.1.6	Transforming Request and Response Data	52
9.1.7	BL Implementation for CUDB Layered NE Provisioning	53
9.1.8	Checking Whether CUDB Is in the Backup Process	54
9.1.9	CUDB Data Inconsistency Alarm	54
9.1.10	CUDB Data Inconsistency During the Create and Delete Operation	54
9.1.11	Implementation of Customized Error Code	55
9.2	CA Development for Monolithic Data Provisioning	55
9.2.1	Preparations	55
9.2.2	Project Creation	56
9.2.3	Interface Specification	59
9.2.4	Define the JDV Access Control Model	60
9.2.5	Define JDV Resource Key Attribute (Optional)	60
9.2.6	Define JDV Loose Error Handling (Optional)	61
9.2.7	Handling the JDV Business Operations	63
9.2.8	Transforming Request and Response Data	63
9.2.9	BL Implementation for Monolithic NE Provisioning	64
9.2.10	Implementation of Customized Error Code	64
9.3	CA Development for Java Connector Architecture	65



9.3.1	Preparations	65
9.3.2	Project Creation	65
9.3.3	Configuring the Resource Adapter Description File	68
9.3.4	Implementing ManagedConnectionFactory	68
9.3.5	Implementing ManagedConnection	69
9.3.6	Implementing StatisticsReporter for Dashboard	70
9.4	CA Development for Cluster Strategy	71
9.4.1	Implementing CA Cluster Strategy by Wizard	72
9.4.2	Cluster Strategy APIs	84
9.4.3	Customizing Off-the-shelf Cluster Strategy	86
9.5	CA Development for Subscriber View	87
9.5.1	Preparations	87
9.5.2	Project Creation	87
9.5.3	Interface Specification	92
9.5.4	Define the JDV Access Control Model	93
9.5.5	Handling the JDV Business Operations	93
9.5.6	Transforming Request and Response Data	94
9.5.7	BL Implementation for Subscriber View	94
9.5.8	SV Mapping and Ignoring Error Messages	95
9.5.9	SV Data Inconsistency During Create Operation	95
9.5.10	Implementing CAI3G DC	96
9.5.11	Implementation of Customized Error Code	97
<b>10</b>	<b>Building and Deployment</b>	<b>99</b>
10.1	Building, Deploying, and Undeploying a CA JDV	99
10.1.1	Building JDV	99
10.1.2	Deploying JDV	100
10.1.3	Undeploying JDV	102
10.1.4	Updating JDV	102
10.2	Building, Deploying, and Undeploying a CA JCA	103
10.2.1	Building JCA	104
10.2.2	Deploying JCA	104
10.2.3	Undeploying JCA	105
10.2.4	Updating JCA	106
10.3	Building, Deploying, and Undeploying a CA Cluster Strategy	107
10.3.1	Packaging and Deployment	107
10.4	Add 3pp Jar Files to the System	109
<b>11</b>	<b>Configuring Dynamic Activation for CA Provisioning</b>	<b>111</b>
11.1	Checking JDV Information	111
11.2	Configuring NE	111
11.3	Configuring Cluster Strategy	111
11.3.1	Configuration and Testing	111
11.4	Configuring Routing	112
11.5	Configuring Loose Error Handling (Optional)	112



11.6	Granting Authority to Provisioning User	112
11.7	Configuring Logging	113
11.8	Manage Custom Processing Log Settings	113
11.8.1	Deploy New Processing Log Settings	113
11.8.2	Undeploy Processing Log Settings	113
11.8.3	List Custom Processing Log Settings	114
<b>12</b>	<b>Testing and Provisioning over CAI3G</b>	<b>115</b>
12.1	Testing Tools for Sending Requests – SoapUI	115
12.2	LDAP Simulator – OpenLDAP	115
12.3	Application Data Browser – Apache Directory Studio	115
12.4	HTTP Test Server	115
<b>13</b>	<b>Appendix A – CUDB Data Model in CUDB Configuration Files</b>	<b>117</b>
13.1	XML Tags in CUDB Configuration Files	117
13.1.1	LDAP Attribute Entry	117
13.1.2	LDAP Attribute staticattr	117
13.1.3	Element attr	117
<b>14</b>	<b>Appendix B – Version History</b>	<b>119</b>
	<b>Reference List</b>	<b>121</b>



# 1 Introduction

This document describes the processes of developing Customer Adaptation (CA) in Ericsson Dynamic Activation (EDA).

## 1.1 Purpose and Scope

This document is intended for the persons developing provisioning Business Logic (BL) with Java™ Data Views (JDVs), outbound connectors or Cluster Strategy in Dynamic Activation.

CA developers and solution architects can develop new JDVs or outbound connectors for operator-specific layered and monolithic applications in Dynamic Activation. In a monolithic architecture, the traffic logic and data are stored on the same node. Data Layered Architecture (DLA) allows traffic logic and data storage to be stored separately on different nodes. Dynamic Activation supports these customized JDVs and outbound connectors.

This document only supports the development of new JDVs and outbound connectors. JDVs and outbound connectors developed based on this manual are called CA JDVs and CA outbound connectors. For CAs on existing JDVs and outbound connectors, contact your local Ericsson representative.

## 1.2 Target Groups

The target groups for this document are as follows:

- CA developers
- Solution architects

## 1.3 Typographic Conventions

Typographic conventions are described in the *Library Overview*, Reference [1].

## 1.4 Prerequisites

The readers of this document must meet the following prerequisites:

- Knowledge of Dynamic Activation
- Knowledge of Linux™
- Knowledge of the Java language, Eclipse, and Maven







## 2 Dynamic Activation Overview

The purpose of Dynamic Activation is to host and serve provisioning BLs. A typical Dynamic Activation workflow is shown in Figure 1. For more information about Dynamic Activation, see *Function Specification Resource Activation*, Reference [2].

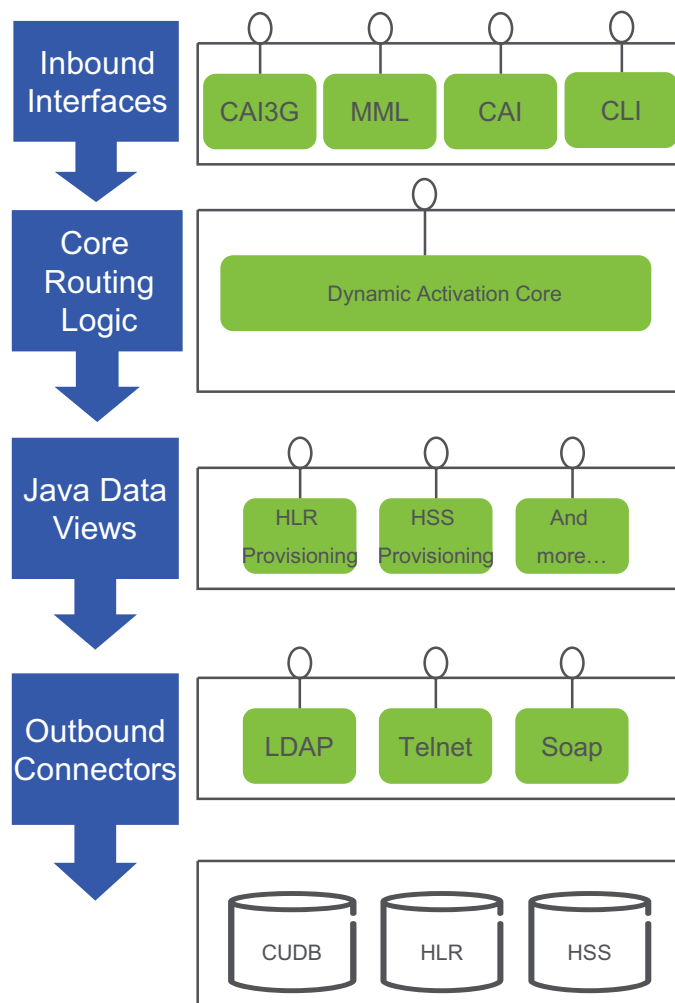


Figure 1 Dynamic Activation Overview

### Inbound Interfaces

The Inbound Interfaces are the access protocols towards Dynamic Activation. The supported protocols are Customer Administration Interface Third Generation (CAI3G) (version 1.2), Man-Machine Language (MML), Customer Administration Interface (CAI), and Command-Line Interface (CLI).



### **Core Routing Logic**

The Core Routing Logic is the core of the product. This is where Access Control, Authorization, Routing of requests, and other core services are applied.

### **Java Data Views**

The JDV layer contains the core of the BL. This is where the workflow of a provisioning request is defined. The JDV has several components, such as service JDV, resource JDV and cluster strategy and so on. These components use the protocol adapters in the Outbound Connectors layer or communicate with other JDVs.

### **Outbound Connectors**

The Outbound Connectors handle the protocol-specific operations towards the NEs. They can be built, based on two different frameworks, Java Connector Architecture (JCA) and Connector Framework. The JCA framework is only used for connectors used for layered Resource Activation within UDC solution. For all other cases, the Connector Framework must be used.

The Dynamic Activation parts that support adaptations are in the JDV and Outbound Connector layers.

## **2.1 Limitations**

The following limitations exist in the CA development of Dynamic Activation:

- The CA of Dynamic Activation supports CAI and CAI3G 1.2 for inbound interfaces.
- The CA supported by Dynamic Activation have some restrictions for layered applications. For details, see Section 10.1.2 on page 100.
- Dynamic Activation supports a limited number of southbound protocols, for example, CUDB LDAP, and HLR MML interface. Other southbound interfaces have to be built as a separate CA.
- If developer needs to implement a CA on inbound interfaces, refer to *Northbound Interface Adapter Customization Development Guide for HTTP-Based Protocol*, Reference [7].



### 3 JDV Overview

The provisioning BL is developed within the JDV in Dynamic Activation. The JDV is a component programmed in Java and can be deployed dynamically on a running server. Basically the JDV component is a Java Archive (JAR) file consisting of Java code and some configuration data. Figure 2 shows the JDV structure and workflow.

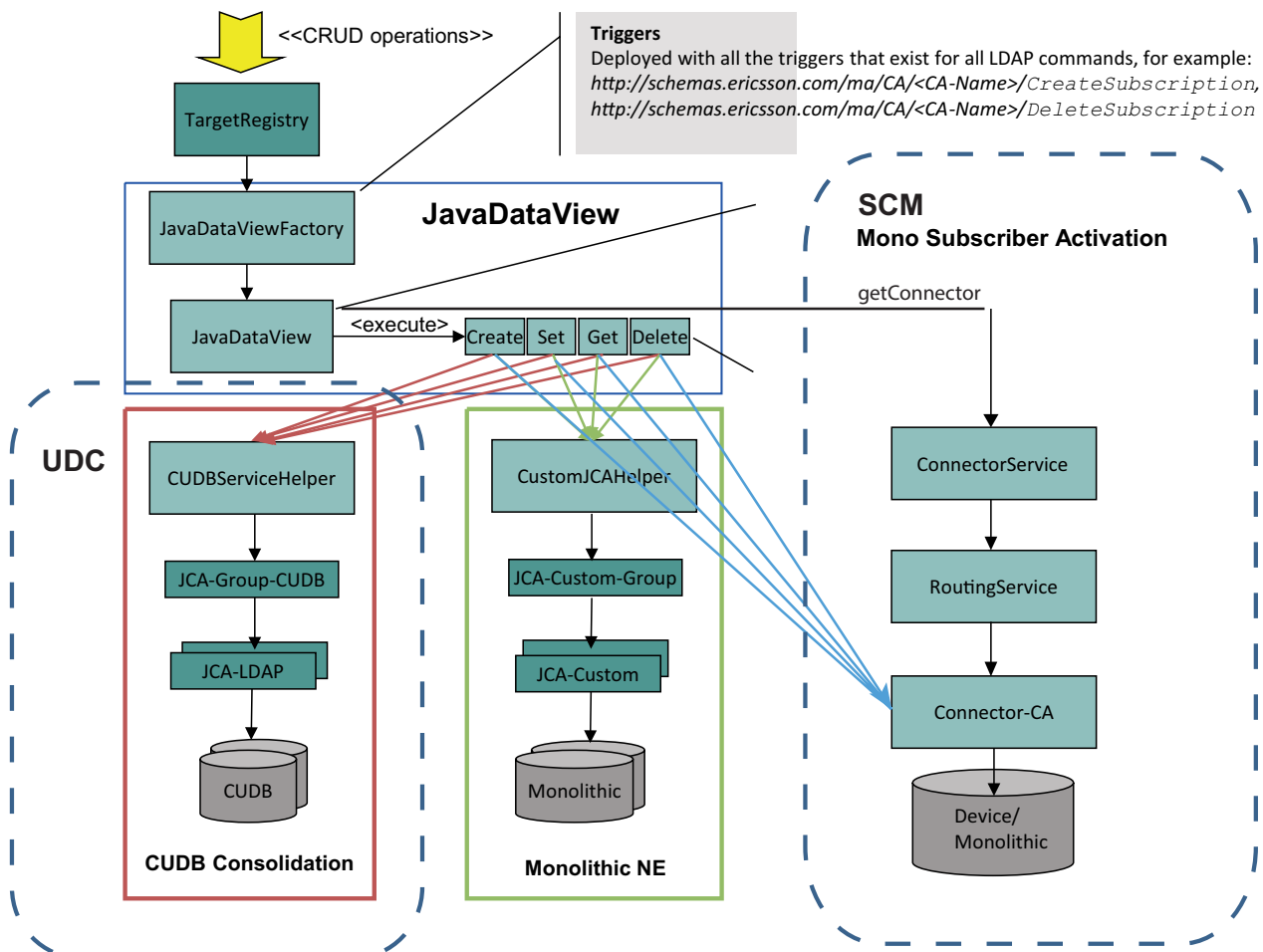


Figure 2 JDV Architecture

When a JDV is deployed, Dynamic Activation registers all the supported Managed Object (MO) operations as triggers in the **TargetRegistry** component. An incoming request for a certain MO operation is routed to the correct JDV BL according to the information in the **TargetRegistry** component.

The JDV component contains the following two important parts:

- The JDV factory object



The JDV factory object creates a JDV instance including sets of data-view definitions in the `javadataview-descriptor.xml` configuration file. Each deployed JDV component has a single JDV factory instance.

- The JDV object

The JDV object handles the incoming request. The incoming request received by the JDV object is a `Request` object. In CA JDVs, the `org.w3c.dom.Document` class represents the incoming request data. The JDV object reads the trigger information from the `Request` object and dispatches the request to the right BL.

Each MO operation in a JDV object is handled by a specific provisioning BL, such as `Create`, `Get`, `Set`, and `Delete` (CRUD). The following sections provide detailed information about the CA JDV development.

The following table contains the main directories and files in a common CA JDV project.

*Table 1 CA JDV Directories and Files*

Directory or File	Content
<code>src/main/resources/META-INF</code>	Configuration files for JDV
<code>src/main/resources/META-INF/web-service_provisioning_cai3g1.2</code>	Web Services Description Language (WSDL) and schema files for JDV
<code>src/main/java/com/ericsson/jdv/&lt;CA-Name&gt;</code>	JDV and JDV factory objects  This directory can contain common definition files, such as common constants and provisioning errors.
<code>src/main/java/com/ericsson/jdv/&lt;CA-Name&gt;/handler</code>	Detailed provisioning BL files
<code>pom.xml</code>	The configuration file used in Maven for JDV project

## 3.1 Northbound Interfaces

CAI and CAI3G interfaces are supported as northbound interfaces.

### 3.1.1 CAI Interfaces

To provision over CAI northbound interfaces, the MO part in trigger must be capital letters and MO must use the namespace `http://schemas.ericsson.com/ma/cai/1.0/`. For more information, see interface specifications in Section 9.1.3 on page 49, Section 9.2.3 on page 59, or Section 9.5.3 on page 92.



### 3.1.2 CAI3G Interface Files

WSDL and schema files are the interface files related to a specific JDV BL and are stored in the `src/main/resources/META-INF/web-service_provisioning_cai3g1.2` directory of a CA JDV project.

These interface files are used for CAI3G inbound request validation.

## 3.2 JDV Configuration Files

JDV configuration files are stored in the `src/main/resources/META-INF` directory of a JDV project.

### 3.2.1 `javadataview-descriptor.xml`

The `javadataview-descriptor.xml` is the main configuration file for a specific CA JDV. This configuration file includes the following:

- The definition of the JDV factory object
- The triggers in the JDV project
- The NE type of the JDV project

**Note:** For subscriber view, skip the configuration of NE Type.

- Other initial configuration items

In this configuration file, the `JavaDataView` tag includes the name and trigger lists that define a JDV object. One trigger represents one operation of a specific MO. The `JavaDataViewFactory` bean is defined in this file as well.

### 3.2.2 `managedobjects-descriptor.xml`

This `managedobjects-descriptor.xml` file defines an MO descriptor that contains a specific MO handled by the JDV object. The MO is a type of service representation in the JDV object.

An MO descriptor contains the following definitions:

- The MO that contains specific triggers in a subscription.
- The trigger that contains an operation, which is executed in a subscription.
- The operation that is handled by the provisioning BL, such as `Create` or `Delete` operation in a subscription.
- The attribute that is handled by the provisioning BL.



The `managedobjects-descriptor.xml` file is a typical example that contains the above four definitions. The file is located in the `src/main/resources/META-INF/` folder. The `managedobjects-descriptor.xml` is the base for the Access Control on both operations level and attribute level. All attributes defined in the operations are displayed in the **Access Control** tab in GUI. For more information on how to use GUI, see Section 11 on page 111.

The attributes in MO identities can be defined as JDV Resource Key attribute (which is optional) for routing purpose. For detailed information about the JDV Resource Key attribute, see Section 9.2.5 on page 60.

### 3.2.3 looseerror-descriptor.xml

The `looseerror-descriptor.xml` file is the configuration file for the loose error handling function (which is optional). This configuration file contains the following definition:

- Loosable errors
- Pre-defined loose error rules

The file is located in the `src/main/resources/META-INF/` folder. The `looseerror-descriptor.xml` is the base for Loose Error Handling. All rules and errors configured in this file are displayed in the **Loose Error Handling** tab in GUI. For more information on how to use GUI, see Section 11 on page 111.

For detailed information about the definition in this file, see Section 9.2.6 on page 61.

## 3.3 Business Logic Files

The following types of BL files are supported in a JDV project:

- The JDV factory file
- The JDV file
- The provisioning BL files

### 3.3.1 JDV Factory File

The JDV factory class is a subclass of the `com.ericsson.pas.javadataview.JavaDataViewFactory`, which is located in the API-PAS library.

The JDV factory file `<CA name>JavaDataViewFactory.java` is located in `src/main/java/com/ericsson/jdv/<CA name>` directory.



The JDV factory configuration file `javadataview-descriptor.xml` defines the JDV factory object. Each deployed CA JDV has only one JDV factory object, and this configuration file must be defined for each CA JDV to generate new JDV instances.

### 3.3.2 JDV File

The JDV class is a subclass of the `com.ericsson.pas.javadataview.JavaDataView`, which is located in the API-PAS library.

The JDV file `<CA name>JavaDataView.java` is located in `src/main/java/com/ericsson/jdv/<CA name>` directory.

The JDV object receives the incoming request, which is a `Request` object. The JDV object routes the `Request` object to the corresponding provisioning BL based on the trigger and operation of the `Request` object.

### 3.3.3 Provisioning BL Files

The provisioning BL files contain the source code for handling the incoming requests. Most commonly one specific provisioning BL file handles one MO operation. The example BL file for the `Create` operation can be found in `src/main/java/com/ericsson/jdv/<CA Name>/handler/CreateBusinessLogic.java`.

For example, the provisioning BL can interact with Centralized User Database (CUDB) by handling the incoming requests and calling API methods in `com.ericsson.jdv.cudb.datamodel.CUDBServiceHelper`. The `CUDBServiceHelper`, which is located in the LIB-JDV-Public library, provides a formal way to `Create`, `Get`, `Set`, and `Delete` the data in CUDB.

For detailed API information about CA JDV development, see Section 9 on page 43.







## 4 JCA Overview

The Figure 3 shows the CA JCA architecture and workflow.

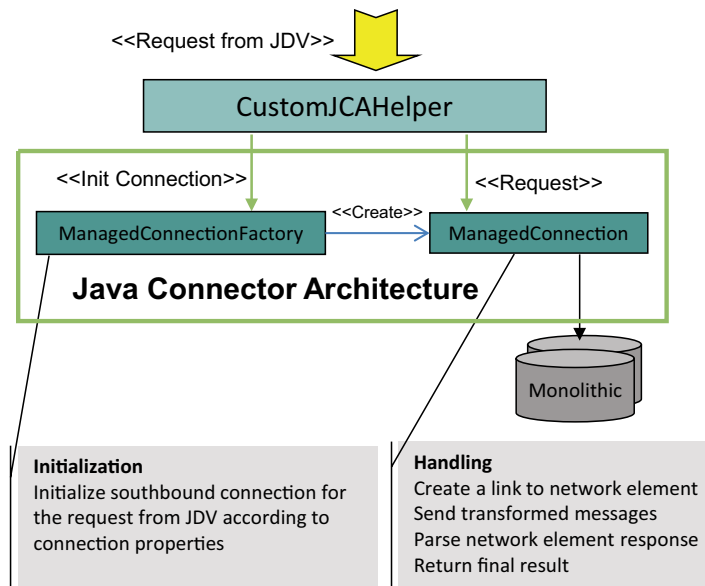


Figure 3 JCA Overview

The southbound adapters in Dynamic Activation are based on Java Connector Architecture (JCA).

The `LIB-JDV-Public` library provides the most basic Common Client Interface (CCI) and abstract Service Provider Interface (SPI) class implementation of JCA connector interfaces. CA developers need to extend these abstract SPI classes and implement CA-specific logic in CA JCA projects.

A CA JCA link contains the following three important parts:

- Resource adapter description file `ra.xml`

Define a resource adapter description file for each JCA link implementation. In this file define the JCA name, version, and the resource adapter classes information. The resource adapter classes information is used to load a JCA link into the Dynamic Activation system during deployment. Each CA JCA project has its own configuration properties that must be defined in this resource adapter description file according to the CA JCA requirements.

- `ManagedConnectionFactory`

Provide a `ManagedConnectionFactory` class that is extended from `com.ericsson.jca.common.spi.AbstractManagedConnectionFactoryImpl` (located in the `LIB-JDV-Public` library) for each CA



JCA link. This class is used to validate the Connection configuration, generate new Connection instances when a new request is received and also handling the heartbeat functionality. Heartbeat is needed for handling communication problems between the Network Element (NE) and Dynamic Activation. When an NE is unreachable, Dynamic Activation does not send traffic to the node. The heartbeat detects when the NE is reachable again, and Dynamic Activation can reach it for traffic once again.

- `ManagedConnection`

The `ManagedConnection` class is the key class of a CA JCA link. The protocol-specific implementation is located in this class. It includes writing to the processing log in Dynamic Activation. All connectors must implement processing log points. The class is extended from `com.ericsson.jca.common.spi.AbstractManagedConnectionImpl` (located in the LIB-JDV-Public library) and implements the `executeMappedRecord` method to handle requests.

The Table 2 contains the main directories and files in a CA JCA project.

*Table 2 Main Directories and Files in a CA JCA Project*

Directory and Files	Contents
<code>src/main/java/com/ericsson/jca/&lt;CA-Name&gt;/spi</code>	CA JCA Java classes
<code>src/main/rar/META-INF</code>	Resource adapter description file <code>ra.xml</code>
<code>pom.xml</code>	Maven configuration file for the JCA project

A complete executable design base project `JCA-Custom3` is available in Dynamic Activation distribution. The `JCA-Custom3` project works together with the `JDV-Custom3-Resource-Provisioning` project.



## 5 Connector Framework Overview

The Figure 4 shows the connector framework architecture.

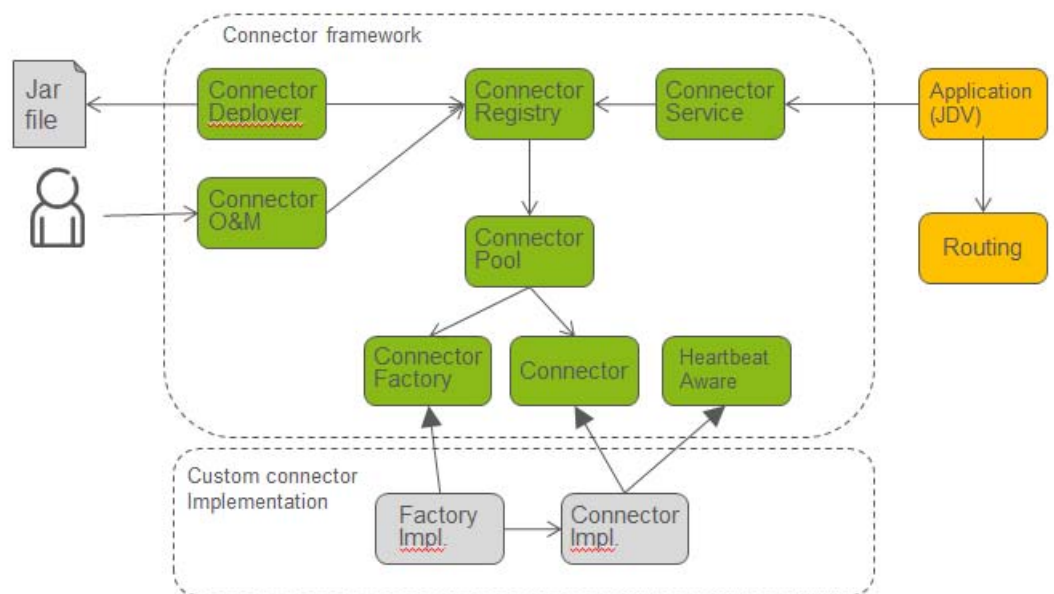


Figure 4 Connector Framework

A connector needs to implement some interfaces in the API. The most important interfaces are `ConnectorFactory`, `Connector`, and `HeartbeatAware`. The `HeartbeatAware` interface is needed if regularly heartbeat is wanted. Heartbeat will only be performed in one of the following cases:

- When the connector is created for the first time.
- If a connection failure occurs when interacting with the connector.

The connector framework will handle fault management. An alarm will be raised upon connection failure and ceased when the connection is restored.

The most important libraries are `ema-connector-fw-interfaces-<version>.jar` and `API-PAS-<version>.jar`. These libraries are available in `/usr/local/pgngn/dve-application-<version>/lib/` on an installed system.

The complete connector framework API is available in `ema-connector-fw-interfaces-<version>.jar`. The connector's implementation of interface `ConnectorFactory`, must use the following annotations available from this file: `@ConnectorDescription` `@ConfigurationParameter`



The business logic will be able to fetch a connector through the `ConnectorService` (`API-PAS-<version>.jar`).

A `JavaDataViewFactory` must implement the interface `ConnectorServiceAware` (`API-PAS-<version>.jar`) to be able to access the `ConnectorService` interface.

The file `dve-connector-fw-impl-<version>.jar` contains the implementation of `ema-connector-fw-interfaces-<version>.jar` and can be used as reference material.

The CA Package `CXP9029435-<version>.tar.gz` file is available on SW Gateway and contains two example projects:

- `CF-CustomHTTP-Connector.jar` is an example of how to implement a connector, based on the connector framework.
- `JDV-CustomHTTP-Provisioning.jar` is an example of business logic using the `CF-CustomHTTP-Connector`.

The example projects are located in the sub-package `ca-sourcecode-<version>.tar.gz`.

## 6 NE Management Overview

Dynamic Activation provides two types of NE management: Cluster Strategy and NE Group.

Figure 5 shows the request flow for the JDVs, the cluster strategy and NE Group.

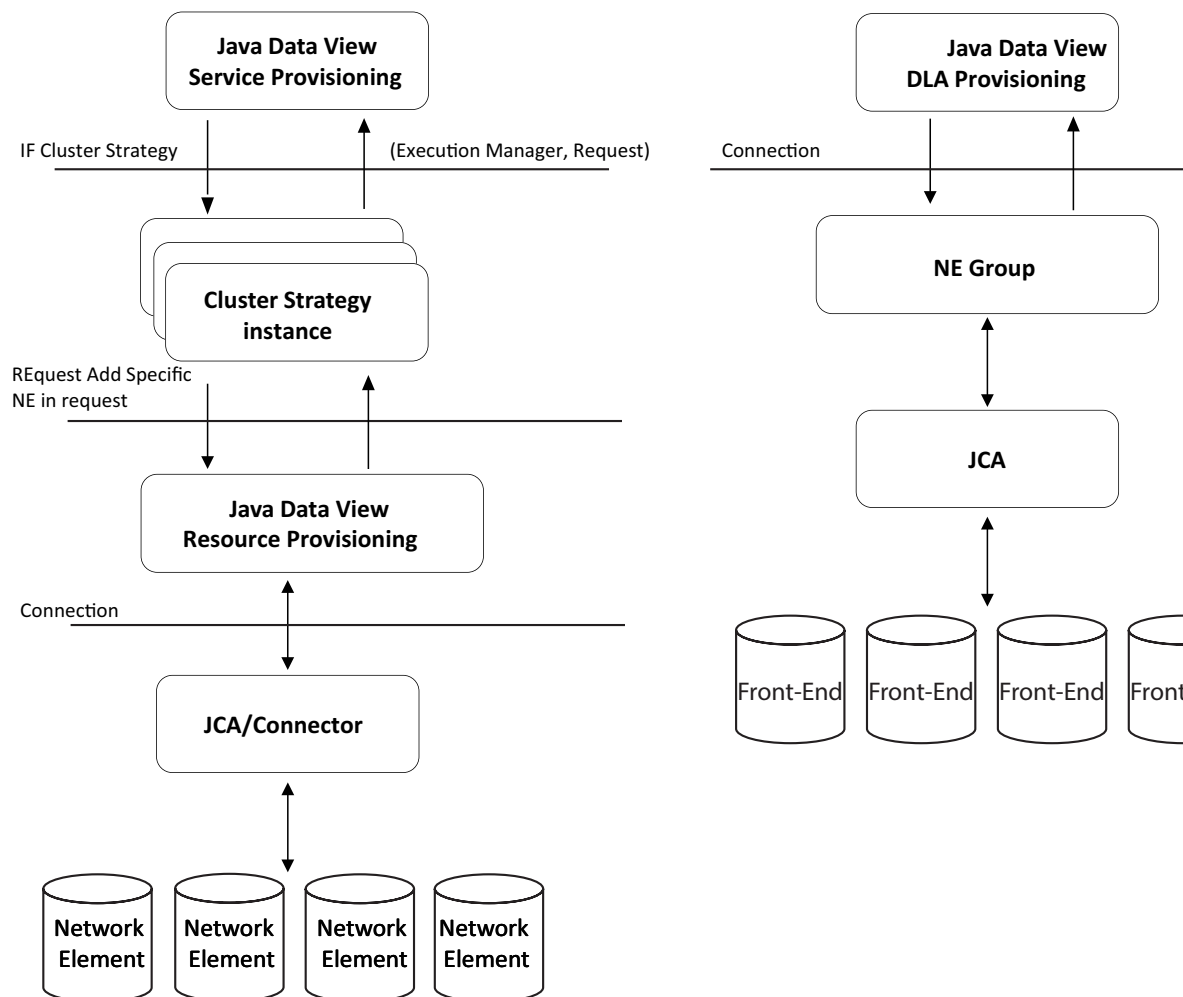


Figure 5 Customer Strategy Flow

The following cluster strategies are available in Dynamic Activation:

- ActiveActive
- ActiveActive-BestEffort
- ActiveActiveQueue



- MultipleAir
- ActivePassive
- PrimaryBackup

CA developers can develop customized cluster strategy as well.

The following NE group types are available:

- Failover
- Round-Robin

NE Groups are only available for JCA based connectors. And NE Group does not support customization.



## 7 CA Application Types

Dynamic Activation supports the following CA application types:

- A consolidated data model for an application having a shared data storage using the DLA.
- A monolithic data model for a monolithic application that has its own data storage.
- A Subscriber View (SV) data model for integrating towards other business logic components.

To develop a CA in Dynamic Activation, use the supplied Eclipse Wizard, as described in Section 9 on page 43, or start with a design base project and modify it for CA applications.

The wizard supports the creation of the following applications:

- Layered JDV
- Monolithic JDV
- JCA
- Cluster strategy
- Subscriber view JDVs

JCA, subscriber view, and cluster strategy are implementation-specific and can be created with example code from supplied basic implementations.

**Note:** The wizard does not support the creation of Connectors based on the connector framework.

Table 3 shows the CA design base projects provided and executable for above application types in the Dynamic Activation distribution.

*Table 3 CA Design Base Projects*

Category	Project Name	Description
CUDB Layered Data NE	JDV-Custom1-Provisioning	This project provides the provisioning workflow of the Custom1 service.
	JDV-Custom1-Configuration	This project provides the CUDB data model of the Custom1 service.



Category	Project Name	Description
Monolithic Data NE	JDV-Custom3-Service-Provisioning and JDV-Custom3-Resource-Provisioning	These projects provide the provisioning of monolithic NE through the southbound JCA link.
	JCA-Custom3	This project provides the southbound JCA connector link adapter.
	DVE-Custom-Cluster-Strategies	This project provides an NE Group Type.
Subscriber View	JDV-Custom4-Provisioning	This project provides the provisioning workflow of the Custom4 service.
Monolithic Data NE Provisioning for non UDC solutions	JDV-CustomHTTP-Provisioning	This project provides the provisioning of monolithic NE for non UDC solutions through the southbound connector.
	CF-CustomHTTP-Connector	This project provides the southbound connector.

All the CA design base projects provide basic provisioning implementation for different NE BL. CA developers can change or extend existing provisioning BL projects according to their specific requirements.

**Note:** The `Custom2` CA design base project is reserved for future CUDB layered data provisioning design base projects.

## 7.1 CUDB Layered Data NE Provisioning

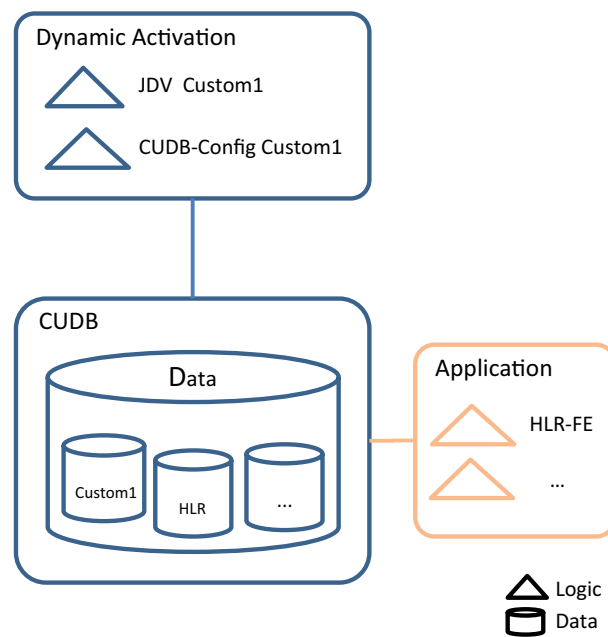
The following subsections contain information about CUDB layered data NE provisioning.

### 7.1.1 CUDB Layered Data Provisioning Overview

The CA wizard supports creation of DLA architecture-based design projects. There is also a CA design base project, `Custom1`, that supports the DLA architecture.

Figure 6 shows a typical DLA application deployment using the `Custom1` design base.





*Figure 6 Data Layered Architecture*

For more information about the customization needed in the CUDB to implement the application-specific data model, contact the local Ericsson representative.

### 7.1.2

#### Data Model

The DLA wizard supports a consolidated data model as shown in the Figure 7 using the `Custom1` design base.

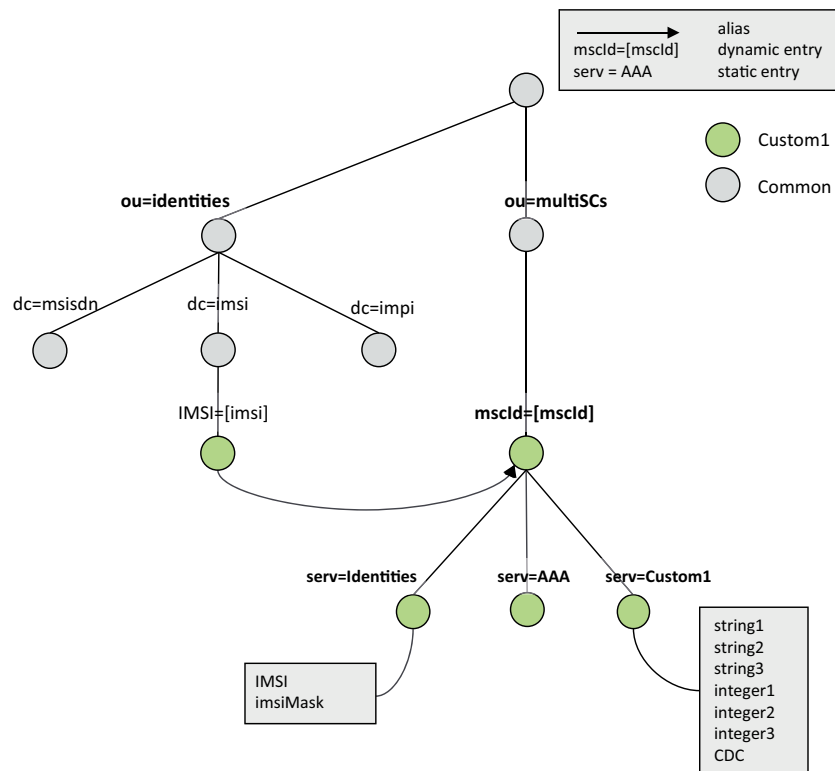


Figure 7 CA Layered Data Model

In the above data model, the `serv=Custom1` data entry is specially used by the `Custom1` application.

The `ou=identities` data entry contains below listed identities. Each CA application can only use one of the following identities:

- MSISDN
- IMSI
- IMPI

The parameters `string1`, `string2`, `string3`, `integer1`, `integer2`, and `integer3` are replaceable parameters in the `Custom1` design base project. These parameters are not possible to add in the DLA wizard and must be manually added if wanted. `Custom1` can be used as reference.

Other data entries are shared by other applications in the same CUDB.

The following table contains the `Custom1` data model parameters in CUDB.



**Table 4** *Data Model Parameters in Layered Data Provisioning*

Parameter	Description	Mandatory = M Optional = O
string1	Format - string Authority - read-write Default Value - ""	O
string2	Format - string Authority - read-write Default Value - ""	O
string3	Format - string Authority - read-write Default Value - ""	O
integer1	Format - unsigned integer Authority - read-write Default Value - ""	O
integer2	Format - unsigned integer Authority - read-write Default Value - ""	O
integer3	Format - unsigned integer Authority - read-write Default Value - ""	O

### 7.1.3 CA DLA Wizard and Base Projects

The wizard can be found in the `CXP9029435-<version>.tar.gz` package in SW GW, refer to Section 8.4 on page 39.

The DLA wizard for CUDB layered application type generates two design projects:

- `<ca-name>-Provisioning`, containing the provisioning workflow.
- `<ca-name>-Configuration`, containing the CUDB data model.

The `<ca-name>-Provisioning` project mainly supports the following features:

- Schema validation of the input `Request` object
- XSLT-based transformation from inbound to outbound request format
- CUDB identity handling
- JDV error handling, including partial execution error codes and alarms
- Support for blocking CUDB provisioning during backup



- JDV basic test code
- Java BL for more advanced tasks
- Properties set from GUI to use in Java BL

The `<ca-name>-Configuration` project contains the definition of the CUDB data model in XML format. For more information on using the wizard, see Section 9.1 on page 43.

No additional CUDB parameters can be specified in the wizard but can be manually added in the generated design project.

The following CA design base projects exist for CUDB layered application type, and are available in the Dynamic Activation distribution:

- `JDV-Custom1-Provisioning`, which contains the provisioning workflow.
- `JDV-Custom1-Configuration`, which contains the CUDB data model.

## 7.2 Monolithic Data NE Provisioning

The following subsections contain information about monolithic NE provisioning within the UDC solution. This information is not applicable for CA development for monolithic NE provisioning outside the UDC solution. JCA development is not supported for areas Resource Configuration and monolithic Resource Activation outside UDC. In these cases, the connector framework described in Section 5 on page 13 must be used.

Section 7.5 on page 33 describes how to develop Monolithic Data NE Provisioning for non UDC solutions.

### 7.2.1 Monolithic Data Provisioning Overview

The monolithic wizard supports creation of monolithic architecture-based design projects, in which the `Custom3` application data and logic are both stored in the NE. There is also a CA design base project, `Custom3`, that supports the monolithic architecture. Figure 8 shows the monolithic application deployment.

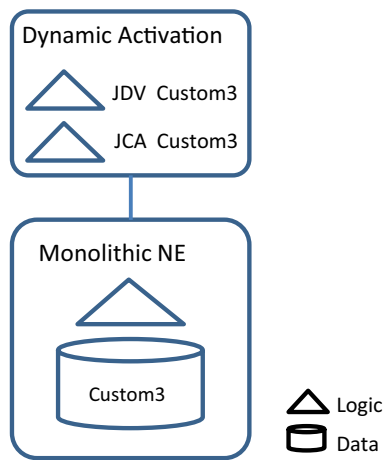


Figure 8 Monolithic NE Provisioning

## 7.2.2

### Data Model

The following table contains the data model parameters in monolithic data provisioning.

The parameters `string1`, `string2`, `string3`, `integer1`, `integer2`, and `integer3` are replaceable parameters in `Custom3` design base project. These parameters are not possible to add in the wizard and must be manually added if wanted. `Custom3` can be used as reference.

Table 5 Data Model Parameters in Monolithic Data Provisioning

Parameter	Description	Mandatory = M Optional = O
username	Format - string Authority - read-write	M
userId	Format - unsigned integer Authority - read-write	O
password	Format - string Authority - read-write	M
string1	Format - string Authority - read-write Default Value - ""	O
string2	Format - string Authority - read-write Default Value - ""	O
string3	Format - string Authority - read-write Default Value - ""	O



Parameter	Description	Mandatory = M Optional = O
integer1	Format - unsigned integer Authority - read-write Default Value - ""	O
integer2	Format - unsigned integer Authority - read-write Default Value - ""	O
integer3	Format - unsigned integer Authority - read-write Default Value - ""	O

### 7.2.3 CA Monolithic Wizard and Design Base Projects

The monolithic wizard generates two JDVs, as described below. The JCA wizard generates a southbound JCA connector link adapter.

The JCA wizard by default creates a code skeleton. However, by checking the “HTTP-based template” during creation, it can be constructed to provision an HTTP dummy server supplied in the CA development `tar.gz` file.

The JDV `<ca-name>-Service-Provisioning` project mainly supports the following features:

- Retrieves the name of the destination to which the request is forwarded.
- Maps the request to the Resource JDV.
- Decides whether the request is to be processed by a cluster strategy or directed to the resource JDV and then sends the request accordingly.
- Sends a rollback request in case of error.

The JDV `<ca-name>-Resource-Provisioning` project mainly supports the following features:

- Schema validation of the input `Request` object
- XSLT-based transformation from inbound to outbound request format
- JDV error handling, including partial execution error codes and alarms
- Monolithic error handling
- JDV basic test code
- Java BL for more advanced tasks
- Loose Error Handling



The JCA *<ca-name>* project mainly supports the following features:

- Sending and receiving messages between Dynamic Activation and target NEs
- Monitoring and reporting NE status by raising alarms
- Writing to processing log
- JCA basic test code

The following CA design base projects exist for monolithic application type, and are available in the Dynamic Activation distribution:

- JDV-Custom3-Service-Provisioning
- JDV-Custom3-Resource-Provisioning
- JCA-Custom3, which provides a southbound JCA connector link adapter. This design base project provisions an HTTP server.

These design base projects contain the provisioning workflow.

## 7.2.4 Migration of Existing JDVs

This section contains information on how to add support for a cluster strategy to an existing JDV (referred as the “resource JDV”), with the minimal number of changes on the resource JDV.

**Note:** If existing activation logic (produced prior to Multi Activation 15.0) uses Round-Robin or Failover or is not included in an NE group, there is no need to migrate to the new Service and Resource JDV structure.

Support for cluster strategy in existing JDVs is achieved by adding a JDV (referred as the “service JDV”). The purpose with adding the service JDV is to move the routing logic from the end of the whole workflow to the beginning. This enables the ResourceActivation-RequestDispatcher to know whether the request is to be dispatched to a cluster strategy or a resource JDV.

### 7.2.4.1 Service JDV

Follow the procedure below to migrate existing JDV to the new structure of the service JDV.

1. Create a new monolithic JDV using the wizard, supplying the same values used by the existing JDV, or create a JDV project based on the CA design base project JDV-Custom3-Service-Provisioning.
2. Update the `javadataview-descriptor.xml` file in the service JDV, to define the same set of triggers as in the resource JDV.



The `repositoryReference` and `repositoryOutput` parameters do not need to be defined in the service JDV since the incoming request payload is validated by the resource JDV.

3. Copy the `managedobject-descriptor.xml` file from the resource JDV to the service JDV.
4. Set the constants `NETYPE` and `NAMESPACE` in class **Constants** to the same values as `NE` type and the namespace in the resource JDV.

The declaration and registration of the `NE` type is moved to the service JDV because the routing is now performed in the service JDV.

5. Rollback is optional, but is recommended to implement for Create operation.

The CA design base project `JDV-Custom3-Service-Provisioning` contains a rollback mechanism for Create operation which can be used as is.

The rollback for failed Set and Delete operations is usually network element implementation dependent, and can have a negative effect on provisioning performance.

#### 7.2.4.2

#### Resource JDV

Follow the procedure below to migrate existing JDV to the new structure of the resource JDV.

1. Update the `javadataview-descriptor.xml` in the resource JDV, rename all the triggers to avoid conflicts with those defined in the service JDV. In `JDV-Custom3-Resource-Provisioning`, the triggers are suffixed with a `_RESOURCE`.
2. Remove the `managedobject-descriptor.xml` file from the resource JDV. This file is not needed in the resource JDV since the access control on the Managed Object is done before the request reaches the service JDV,
3. Set the second argument from the `NE` type to null when instantiating the `CustomJCAHelper`. The `NE` type is not needed since the routing is not handled in `CustomJCAHelper`.
4. Remove the registration of the `NE` type from the initialization of the Java Data View Factory in the resource JDV.
5. Remove the definition of the constant to store the `NE` type, if there is one.
6. In the resource JDV, replace the first argument passed to `CustomJCAHelper`, with the name of the network element that is stored in the request. For more details, see Section 7.2.4.2.1 on page 26

If Loose error handling is to be used (optional), see Section 7.2.4.2.2 on page 27.





#### 7.2.4.2.1 CustomJCAHelper API

When the API is invoked with the name of a network element, instead of a request object, `CustomJCAHelper` sends the request to the specified network element without performing a routing lookup.

The `NetworkElementInstance-Helper` class in the CA design base project `JDV-Custom3-Resource-Provisioning` contains an example on how the resource JDV can retrieve the name of the network element from the request object.

#### 7.2.4.2.2 Loose Error Handling

Loose Error handling is used when an error is considered as acceptable and therefore can be ignored.

Loose Error Handling can be defined by the following two ways.

- Configuring Loose Error Handling in GUI

The loose error handling can be defined by configuring the rules in **Loose Error Handling** tab in GUI. When a JDV error response matches the loose error rule condition, this error will be considered as acceptable and ignored.

To support loose error handling configuration in GUI, the `looseerror-descriptor.xml` must be defined in CA project. See the `src/main/resources/META-INF/looseerror-descriptor.xml` in the CA design base project `JDV-Custom3-Resource-Provisioning` for an example on how to define `looseerror-descriptor.xml`.

For detailed information about the definition in `looseerror-descriptor.xml`, see Section 9.2.6 on page 61.

- Implementing Loose Error Handling in JDV

The loose error handling can be defined by implementing loose error handling in JDV. This is recommended to implement for delete operation to facilitate the rollback for create operation in the Service JDV.

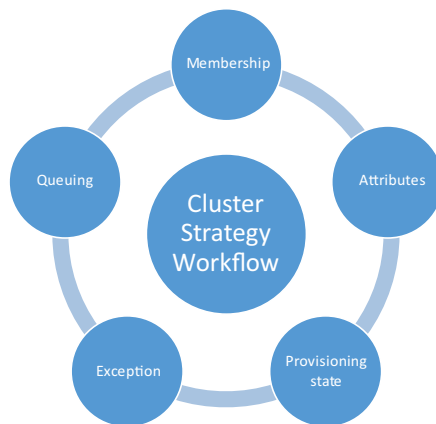
To implement loose error handling for delete operation, catch the exception thrown by the `CustomJCAHelper`. Then cease the error if it is caused by deleting non-existing data.

See class `DeleteBusinessLogic` in the CA design base project `JDV-Custom3-Resource-Provisioning` for an example on how to implement loose error handling for Delete operation.

## 7.3 Cluster Strategy

### 7.3.1 Cluster Strategy CA Development Overview

Cluster Strategy is a business logic which is used to orchestrate provisioning behavior between the service provisioning layer and resource provisioning layer.



*Figure 9 Factors of Cluster Strategy Workflow Model*

Before designing the customized cluster strategy workflow model, CA developers need to draw an activity diagram to illustrate how to orchestrate the provisioning requests of the target cluster strategy. The following factors need to be considered in the activity diagram.

- **Membership (Mandatory):** Membership means the relationship between clustered NEs by assigning different roles to each NE. In Cluster Strategy Workflow, the provisioning order of the Clustered NEs can be controlled by their roles.

For example, in a cluster, one NE has role Primary and the rest NEs have role Secondary. The request will be sent to the primary NE first, then to the secondary NEs.

To implement the membership in CA, `.json` file and JDV class file need to be modified, see Section 9.4.1.4 on page 75 and Section 9.4.1.5 on page 83.

- **Provisioning State (Optional):** Provisioning State means the logical status of an NE. In Cluster Strategy Workflow, the provisioning to an NE can be controlled by the state of the NE. And user can configure the state of NE through GUI. Also, the provisioning state is different with the connection state of the NE. For example, user can configure the provisioning state of NE to on, off, maintenance or backup on GUI.



To implement the Provisioning State in CA, `.json` file and JDV class file need to be modified, see Section 9.4.1.4 on page 75 and Section 9.4.1.5 on page 83.

- **Attributes (Optional):** Consider this factor if addition attributes are required to handle the special requirements and these attributes are applicable to cluster. For example, the request needs to be provisioned with a different logic in a specific time range and this time range is decided by user. Then, an attribute that handles this time range is needed.

To implement the addition attributes in CA, configuration file, `.json` file and JDV class file need to be modified, see Section 9.4.1.3 on page 74, Section 9.4.1.4 on page 75 and Section 9.4.1.5 on page 83.

- **Exception (Optional):** Consider this factor if the exception cases are required when clustered NE provisioning gets failed.

To implement the exception in CA, JDV class file needs to be modified, see Section 9.4.1.5 on page 83.

- **Queuing (Optional):** Consider this factor if certain requests need to be queued and retried by processing queue. The queuing condition can be defined according to the role, provisioning state, attribute and exception case.

To implement the queuing in CA, JDV class file needs to be modified, see Section 9.4.1.5 on page 83.

## 7.4 Subscriber View Provisioning

The following subsections contain information about SV provisioning.

### 7.4.1 Subscriber View Overview

Business logics components expose network applications, for example HLR or EPS. An SV can aggregate such business logics and provide high level, business-oriented provisioning interface on top of the business logic to the Business Support System (BSS). The SV is an abstract layer referring to the capabilities of the underlying resource features of the product. The SV does not contain any specific provisioning solutions.

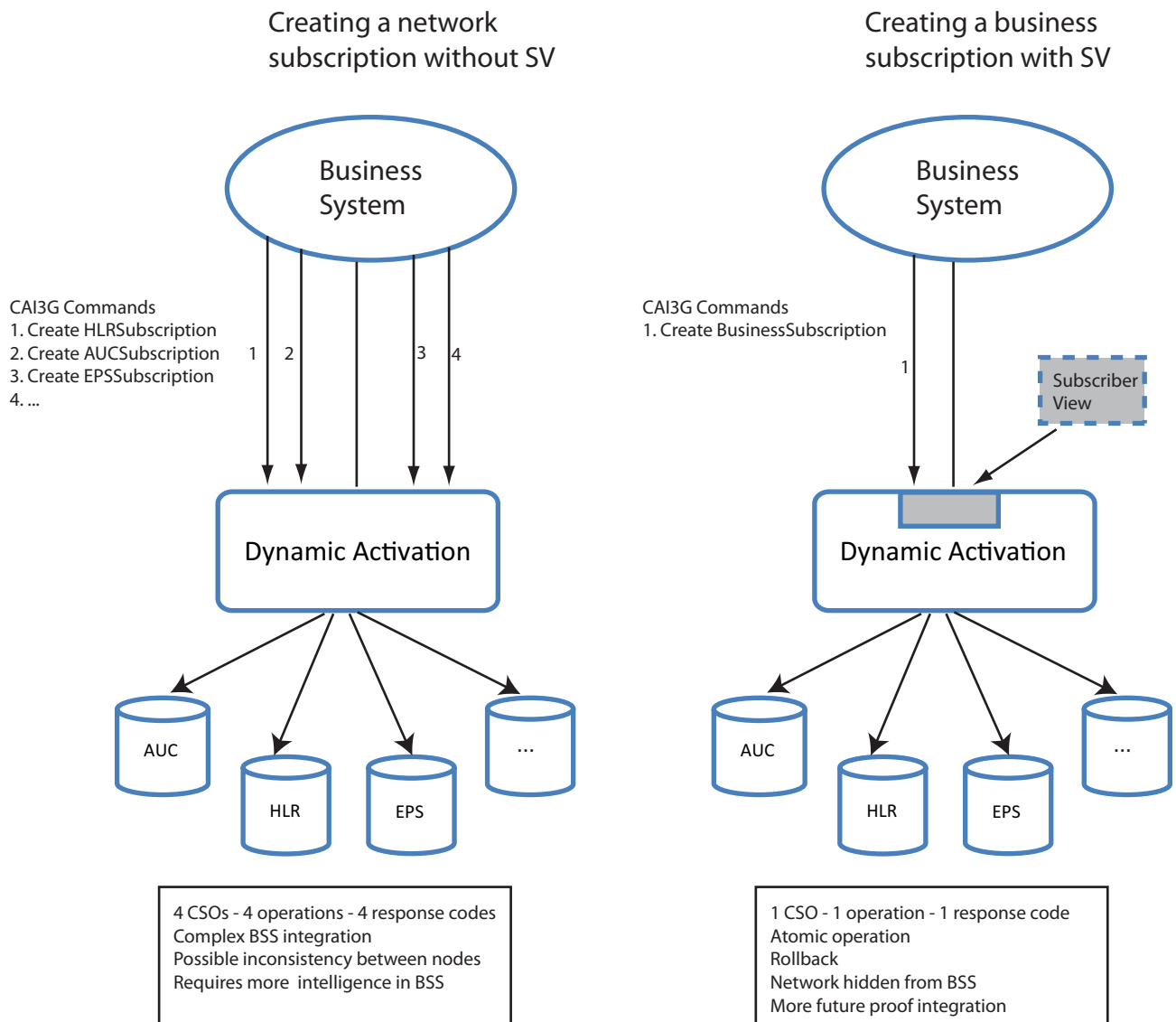


Figure 10 Overview When Using Subscriber View

The wizard supports creation of SV design projects and can be selected to include example code where the composed HLR and EPS data of the application is stored in CUDB. This setup can also be found in the CA design base project `Custom4`. Figure 11 shows a typical DLA application deployment that SV is on top of.

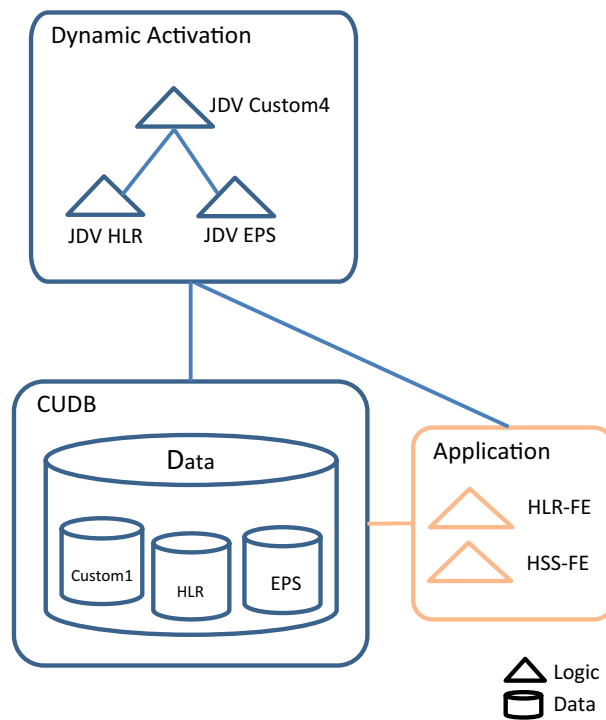


Figure 11 Subscriber View Provisioning

## 7.4.2

### Data Model

The SV wizard supports an exemplified composed data model for HLR and EPS that is shown in Figure 12 using the `Custom4` design base.

This data model is implemented in CUDB and the communication with CUDB is handled by JDV HLR and JDV EPS. The `Custom4` JDV consolidates the data model for exposure towards BSS.

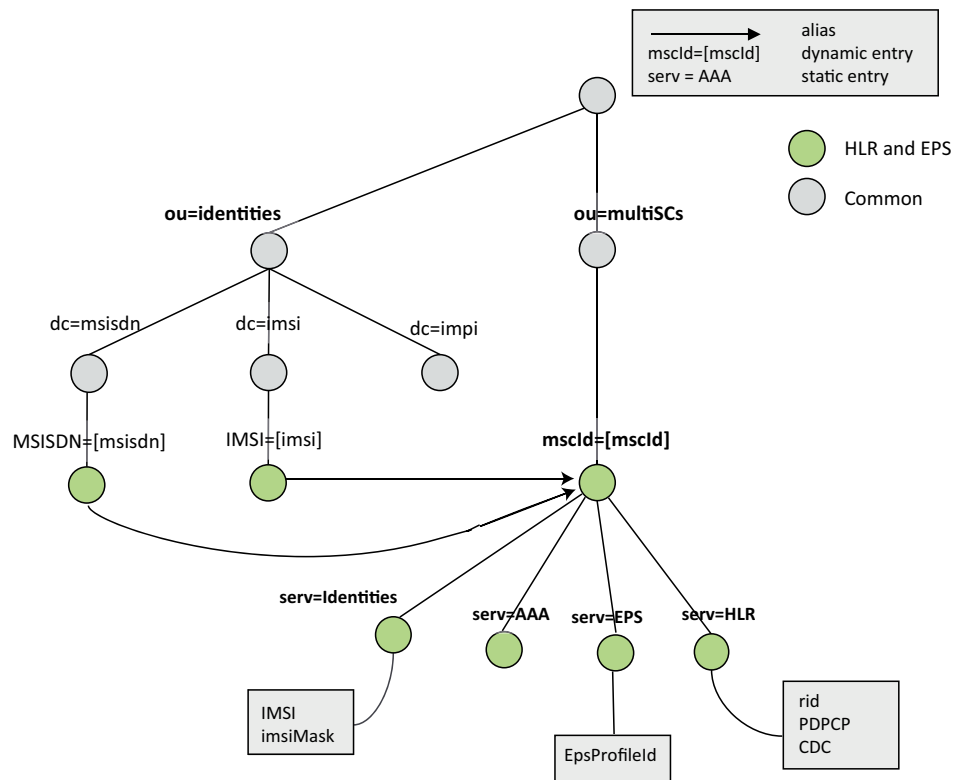


Figure 12 HLR and EPS Data Model

The parameters `rid`, `PDPCP`, and `EpsProfileId` are replaceable parameters in `Custom4` design base project depending on the actual provisioning data that SV composes.

The following table contains the HLR and EPS data model parameters in CUDB.

Table 6 Data Model Parameters in HLR Provisioning

Parameter	Description	Mandatory/Optional
<code>rid</code>	Format - integer Authority - read-write Default Value - ""	Optional
<code>PDPCP</code>	Format - integer	Optional

Table 7 Data Model Parameters in EPS Provisioning

Parameter	Description	Mandatory/Optional
<code>epsProfileId</code>	Format - string Authority - read-write Default Value - ""	Mandatory



### 7.4.3 CA Subscriber View Wizard and Design Base Projects

The SV wizard creates a code skeleton for SV base projects. It can be selected to include example code for adding HLR and EPS support.

The generated project supports the following features:

- Schema validation of the SV request
- XSLT-based transformation from SV request format to internal JDVs request format
- Execute each internal JDV request
- JDV error handling, including rollback and error mapping
- JDV basic test code
- Java BL for more advanced tasks

The CA design base project currently existing for SV application type is `JDV-Custom4-Provisioning`. This CA design base project contains the provisioning workflow and is located in the `CXP9029435-<version>.tar.gz` package in SW GW.

## 7.5 Monolithic Data NE Provisioning for non UDC Solutions

This section contains information about monolithic NE provisioning, outside the UDC solution, using the connector framework described in Section 5 on page 13.

### 7.5.1 Monolithic Data Provisioning Overview

There is a CA design base project, `CustomHTTP`, that supports the monolithic architecture outside the UDC solution. Figure 13 shows the monolithic application deployment using the `CustomHTTP` design base.

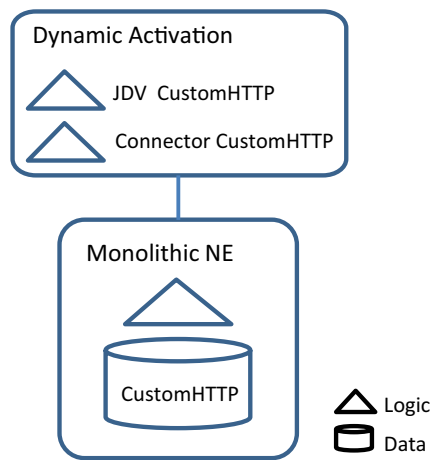


Figure 13 Monolithic NE Provisioning using the CustomHTTP Design Base

## 7.5.2 CA Design Base Projects

The following CA design base projects exist for monolithic application type, and are available in the following Dynamic Activation distribution:

- JDV-CustomHTTP-Provisioning
- CF-CustomHTTP-Connector

These design base projects contain the provisioning workflow.

## 7.5.3 Developing Monolithic Data NE Provisioning for non UDC solutions

This section contains information on how to develop connectors and JDVs based on the connector framework.

### 7.5.3.1 Connector

Follow the procedure below to create a connector:

1. Create a connector project based on the CA design base project CF-CustomHTTP-Connector.
2. The connector must have an implementation of the `ConnectorFactory` interface.

There are two types of annotations that must be used to set, for example, display name for the connector and its fields:

1. - `@ConnectorDescription` - is used to provide meta-data information on the connector. It contains the following four attributes:





Attribute	Description	Occurrence
displayName	Display name for the connector to be displayed, for example in the user interface.	Mandatory
connectorType	Connector type, for example HTTP.	Mandatory
version	Version of the connector. To be used in a multi version setup.	Optional
connectorInterface	Business interface for the connector. Used for example by the business logic when looking up and invoking a connector.	Mandatory

2. - `@ConfigurationParameter` - is used to mark fields as configuration parameters. It contains the following four attributes:

Attribute	Description	Occurrence
displayName	Display name for the field to be displayed, for example in the user interface.	Optional
description	Description for the field to be displayed or example in the user interface.	Optional
password	Indicates if the field is to be treated as a password.	Optional
mandatory	Indicates if a configuration parameter is mandatory for the connector.	Optional

3. The connector must also have an implementation of the `Connector` interface.

To enable writing to processing logs the `ProclogAware` interface must also be implemented.

To enable heartbeat behavior, the `HeartbeatAware` interface must also be implemented.

### 7.5.3.2

### JDV

Follow the procedure below to create a JDV using a connector, based on the connector framework:

1. Create a JDV project, based on the CA design base project `JDV-CustomHTTP-Provisioning`.
2. The implementation of `JavaDataViewFactory` must also implement `ConnectorServiceAware`.

It is possible to retrieve a connector instance through the connector service provided by `ConnectorServiceAware`





## 8 Preparing for CA Development Environment

CA JDVs and CA JCAs are written in Java. It is recommended to develop CA JDVs and CA JCAs with Eclipse Integrated Development Environment (IDE) and build them in Maven. The CA development package in the CXP9029435-`<version>`.tar.gz file is available on SW GW.

This package contains the following packages:

- `ca-deployables-<version>.tar.gz` - Deployable CA Design Based projects
- `ca-plugins-<version>.tar.gz` - Development tools
- `ca-repository-<version>.tar.gz` - Maven repository and settings.xml file
- `ca-sourcecode-<version>.tar.gz` - CA Design Based Projects source code
- `ca-dup-convert-tool-<version>.tar.gz` - DUP migration tools

Before the CA development, extract the tar.gz file to a local directory, such as `CA_Development`. The following subsections use this directory name as an example.

### 8.1 JDK

Download and install the latest Java Development Kit (JDK) 8 from <http://www.oracle.com/technetwork/java/javase/downloads/index.html>.

Set the environment parameters for JDK correctly, such as `JAVA_HOME` and `path`.

### 8.2 Maven

Download and install Maven 3.2.3 from <http://maven.apache.org/>.

Create and set the environment variable `M2_HOME` to the path to the Maven directory. Update the environment variable `Path` with `%M2_HOME%\bin`.

#### 8.2.1 Maven Repository

The Maven repository and the settings file, `settings.xml`, are stored in the `.m2` directory that is located at the `HOME` directory such as `C:\Users\<user name>`. The `.m2` directory is delivered in the `CA_Development` directory.



The development of CA JDVs and CA JCAs on the Dynamic Activation platform is based on the following libraries:

- `API-PAS`
- `Utils-PAS`
- `LIB-JDV-Public`

These public libraries are all backward compatible and mandatory for creating JDVs and JCAs.

These libraries can be found in the Maven repository. For more information about these libraries, see the related Java documentation in the repository.

If there is a `.m2` directory in the `HOME` directory, perform a backup of the `.m2` directory and then remove it from the `HOME` directory. Copy the new `.m2` directory from the `CA_Development` directory to the local `HOME` directory.

Configure the `settings.xml` file. Make sure the following mirror URLs exist in the `settings.xml` file:

- `http://repo1.maven.org/maven2/`
- `http://repository.sonatype.org/content/groups/forge/`

Update or remove the proxy part in the `settings.xml` file, depending if a proxy is needed or not to communicate with the URLs above.

For information about how to configure the settings file, see the Maven site <http://maven.apache.org/settings.html>.

## 8.3 IDE

All CA design base projects are general Maven projects. CA developers can decide which IDE to be used for the JDV and JCA development. The instructions in this development guide are based on Eclipse, which is the recommended IDE. Kepler is the supported version of Eclipse.

### 8.3.1 Eclipse

Follow the steps below to prepare the Eclipse environment:

1. Download and install the latest Eclipse IDE for Java EE Developers from <http://www.eclipse.org/>.

For instructions about how to develop Java projects with Eclipse, see the related Eclipse manual.

2. Follow the steps below to install the Eclipse plug-in `Maven Integration for Eclipse`.



- a Open Eclipse, and click **Help > Eclipse Marketplace and search for Maven**.
- b Click **Install Maven Integration for Eclipse** and click **Confirm**.
- c Click **Finish**.

To configure Maven in Eclipse, click **Window > Preferences**, and select **Installations** under the **Maven** category.

## 8.4 Wizard

CA developers are provided with proprietary tools available within CA development package. These tools are the following:

- Wizard - an Eclipse plug-in for generation of CA design base projects
- Customer Adaptation Code Generator - a command-line tool for generation of CA design base projects.

**Note:** The Customer Adaptation Code Generator tool is a command-line alternative for using Wizard and is available under `CA-CodeGenerator` folder in CA development package. The recommended and supported way of generating Customer Adaptations is however to use the Wizard.

### 8.4.1 Installation of Wizard

Wizard is available under `CA-Wizard` folder in CA development package. Before installing the tool, make sure that Eclipse is installed (all releases from 4.3 are supported). To install Wizard, perform the following actions:

- Close Eclipse if it is running
- Copy files `com.ericsson.ca.wizard.core-1.0.0.jar` and `com.ericsson.ca.wizard.ui-1.0.0.jar` to the `plugins` directory of Eclipse installation folder.
- Start Eclipse.

To check installation status in Eclipse, open **Help > About Eclipse > Installation Details > Plug-Ins**. Searching for keyword “CA Wizard” in filter field, gives number of plugins. If no plugins are displayed in the list, installation has failed.

## 8.5 Initializing Design Base Projects

**Note:** The recommendation is to use the Wizard for generating projects. However, the design base projects are still available in the product.



This section provides procedures for importing and renaming design base projects.

### 8.5.1 Importing the Project

CA developers can start to initialize their own project by importing existing design base projects. The following projects are available:

- JDV-Custom1-Provisioning and JDV-Custom1-Configuration

These projects are for CUDB layered NE provisioning adaptations.

- JDV-Custom3-Service-Provisioning and JDV-Custom3-Resource-Provisioning

These projects are for monolithic NE provisioning adaptations.

- JCA-Custom3

This project is for connector adaptations.

- DVE-Custom-Cluster-Strategy

This project is for cluster strategy adaptations.

- JDV-Custom4-Provisioning

This project is for SV adaptations.

Before the development of actual provisioning BL, follow the steps below to import a required design base project into Eclipse as an existing Maven project. The projects of Custom1 application type can be used as an example:

1. Open Eclipse, and click **File > Import**.
2. In the **Import** dialog, select **Maven > Existing Maven Projects** and click **Next**.
3. Click **Browse** and select the **CA\_Development** directory > **JDV-Customx** project that is used.
4. Click **Next** and **Finish**.

### 8.5.2 Renaming the JDV and JCA Projects (Optional)

Follow the steps below to rename the JDV or JCA project (take the Custom1 project as an example):

1. Follow the substeps below to remove the design base project **JDV-Custom1-Provisioning** from Eclipse:
  - a Right-click the **JDV-Custom1-Provisioning** project in **Project Explorer**, and select **Delete** in the context menu.



- b Click **OK** in the **Delete Resource** window.

**Note:** Do not select the option **Delete project contents on disk**.

- 2. Follow the substeps below to set the name of the **JDV-Custom1-Provisioning** project directory to `<CA-JDV-Project-Name>`:

- a Rename the directory name of the JDV project to `<CA-JDV-Project-Name>`.
- b Replace the `artifactId` and `name` in the `pom.xml` file with your specific project name. In the following example, the values of `artifactId` and `name` are set to `JDV-Custom1-Provisioning_1`:

```
<artifactId>JDV-Custom1-Provisioning_1</artifactId>  
<name>JDV-Custom1-Provisioning_1</name>
```

- c Follow Step 1 to Step 4 in Section 8.5.1 on page 40 to import the project into Eclipse again.







## 9 CA Development Workflow

This section contains information about CA development for CUDB layered data provisioning, monolithic data provisioning, outbound connectors, cluster strategy, and SV.

### 9.1 CA Development for CUDB Layered Data Provisioning

#### 9.1.1 Preparations

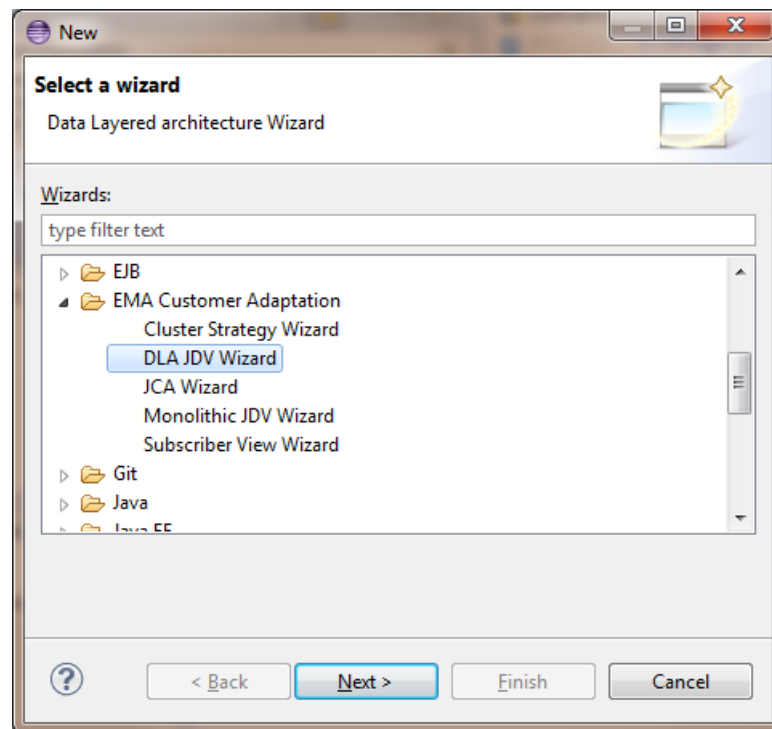
Prepare for the creation of the new CA JDV by analyzing the application to be provisioned and its data model. CA developers need to decide the application type, the application data model details, possible data model differences between the northbound and southbound interfaces, and so on.

If the application type is for a CUDB layered data model and requires changes to the CUDB data model, the work must be initialized for customization in the CUDB as well. If so, contact the local Ericsson representative.

#### 9.1.2 Project Creation

Use the wizard in Eclipse to create a new layered JDV.

1. Start Eclipse and select **File > New > Other** from the menu bar.
2. Select the type of wizard to use: **EMA Customer Adaptation > DLA JDV Wizard** and click **Next**.

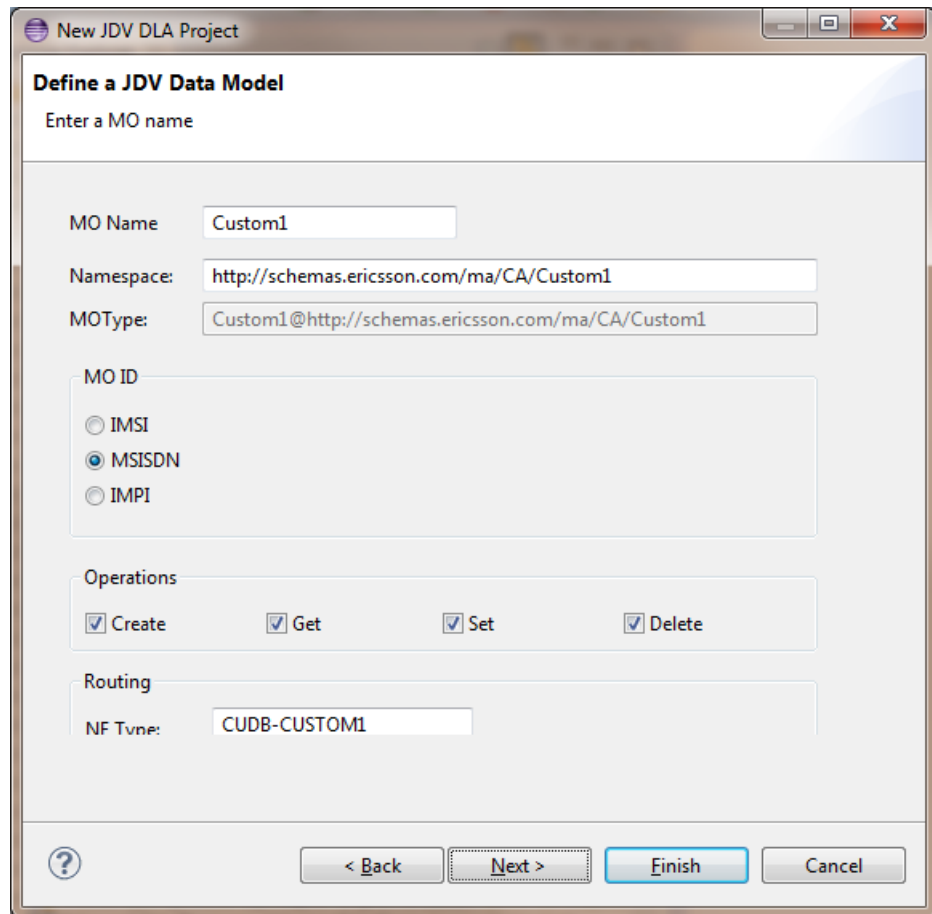


3. Specify the information for the JDV project to create and click **Next**.



Parameter	Description
Project name	The name of the project to be created for DLA JDV.
Package name	The Java package structure.
Use default location	If checked, the system default project name and location are used.
Location	The directory where the new project is stored.
Version	Version of build to be used when packaging CA.
Network	Data Layered Architecture

- Specify the information for the JDV data model and click **Next**.



**New JDV DLA Project**

**Define a JDV Data Model**

Enter a MO name

MO Name:

Namespace:

MOType:

MO ID

☐ IMSI

☒ MSISDN

☐ IMPI

Operations

☒ Create ☒ Get ☒ Set ☒ Delete

Routing

NF Type:

Parameter	Description
MO Name	The specific MO name. Must start with an alphabetical character. Must be in capital letters if using CAI northbound interface.
Namespace	The namespace to use for the MO. <ul style="list-style-type: none"> <li>For CAI, must begin with <code>http://schemas.ericsson.com/ma/cai/1.0/</code></li> <li>For CAI3G, recommend beginning with <code>http://schemas.ericsson.com/ma/CA/</code></li> </ul>
MO Type	The MO Name and Namespace together form MO Type.
MO ID	Select which identity to use, see section Section 7.1.2 on page 19.  The CA of CUDB layered data provisioning only supports one identifier per application. The identifiers are predefined as IMSI, MSISDN, and IMPI.



Parameter	Description
Operations	Select which operations to create.
Routing	Specify the name of the NE type to use.

5. Create a CUDB configuration project and click **Next**.

Parameter	Description
Service Name	Adds the new service to CUDB configuration.
Service Bit	<p>Select what service bits to use as representation for the CA application in the CUDB. Possible values are 12-15 .</p> <p>The CA of CUDB layered data provisioning supports up to four CA applications at the same time.</p>

6. Specify configuration parameters for the JDV and click **Next**.

**New JDV DLA Project**

**Add configuration parameters**

Configuration parameters:

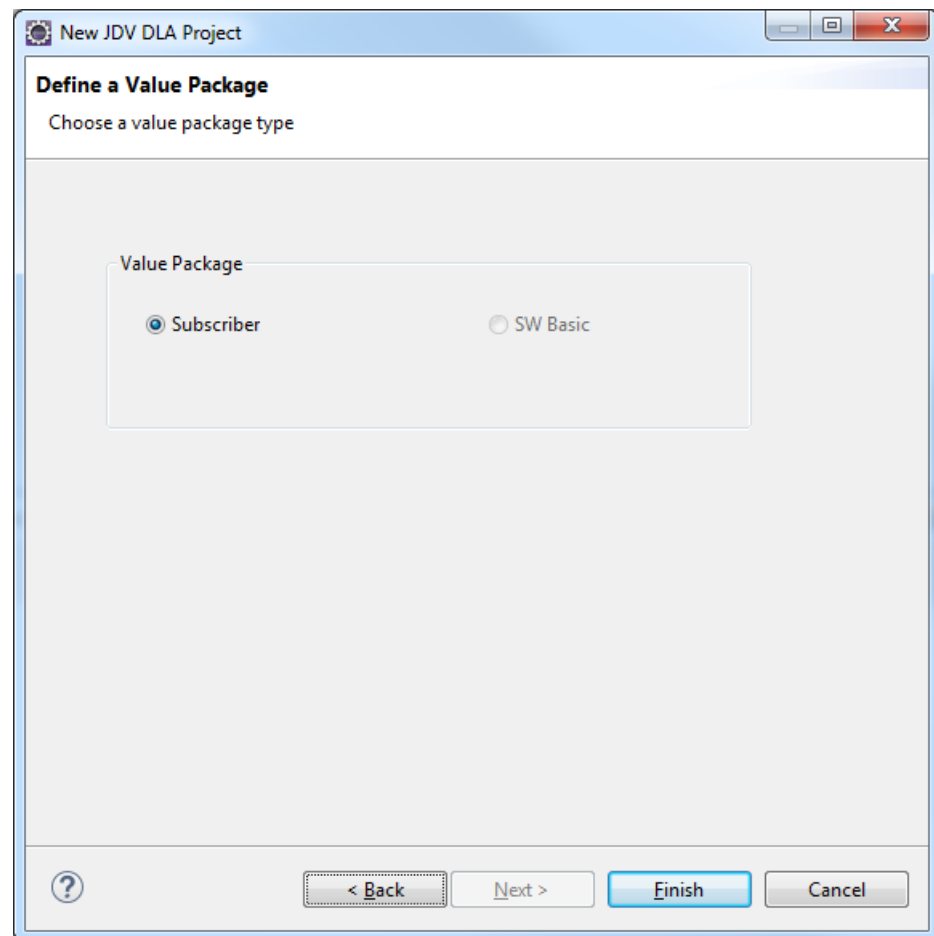
Name	Type	Default value
rootdn	String	dc=operator,dc=com

Buttons: Add..., Remove, < Back, Next >, Finish, Cancel

Parameter	Description
Configuration parameters	Add list of parameters that should be possible to configure from the GUI to use in the JDV. <sup>(1)</sup>

(1) Do not use Java reserved words, follow Java variable naming rules.

7. Select what value package to use and click **Finish**.



Parameter	Description
Value Package	Select which value package to use for this JDV. <code>SW Basic</code> is only used for Subscriber view JDVs. For more details about what value package to use, see Section 10.1.2 on page 100.

Now two new projects are generated, one with the configuration and one BL JDV.

### 9.1.3 Interface Specification

The following limitations exist in the data model definition for CUDB layered data provisioning:

- The CAI northbound interface must be set to `http://schemas.ericsson.com/ma/cai/1.0/`



The CAI3G northbound interface namespace is recommended containing `http://schemas.ericsson.com/ma/CA/` as prefix. For example, namespace for Custom1 is `http://schemas.ericsson.com/ma/CA/Custom1`.

- The CA of CUDB layered data provisioning only supports one identifier per application. The identifiers are predefined as IMSI, MSISDN, and IMPI.
- The CA of CUDB layered data provisioning supports up to four CA applications at the same time.

The created projects have a base structure that can be extended with more attributes. For examples on how to extend, see the `Custom1` base project.

The procedure of defining a JDV data model for CUDB layered data provisioning is as follows:

- Customize the northbound interface data model:
  - Find the northbound schema and WSDL files in the `src/main/resources/META-INF/webservice_provisioning_cai3g1.2` directory of the CA JDV project `JDV-Custom1-Resource-Provisioning`.
  - For CAI interfaces, `javadataview-descriptor` must:
    - Have triggers with MO in capital letters.
    - Use the namespace `http://schemas.ericsson.com/ma/cai/1.0/`

Example of valid CAI trigger of `javadataview-descriptor.xml`:  
`<trigger value="ns>CreateEIRSUB"/> where`  
`xmlns:ns="http://schemas.ericsson.com/ma/cai/1.0/"`

- For CAI3G 1.2 interfaces, define the namespace, operations, and other related parameters of northbound requests in the schema and WSDL files according to *Generic CAI3G Interface 1.2*, Reference [3] .
- Customize the southbound interface data model:

The southbound interface data model is defined in a CUDB configuration file. It can be found in the root of the created Configuration project with the name `CUDBConfig_<MO name>.xml`. The CUDB configuration file mainly defines the CUDB attribute mapping relationship between the northbound interface and the CUDB. For detailed mapping description, see Section 13 on page 117. For examples on how to implement CUDB configuration, see the `JDV-Custom1-Configuration` design base project that provides a CUDB configuration file for the Custom1 project.





#### 9.1.4 Define the JDV Access Control Model

The access control model includes the configuration of JDV MO, attributes, and operations that fulfill the access control criteria in `managedobjects-descriptor.xml` file. Dynamic Activation applies user authority on both MO and attribute level with these criteria and rules defined in GUI, see Section 11.6 on page 112.

The procedure of defining a JDV access control model is as follows:

1. Find the MO descriptor file in the `src/main/resources/META-INF/` directory of the CA JDV project.
2. Define a JDV group with MO, attributes, operations, and other related parameters to be shown on the access control GUI.

#### 9.1.5 Handling the JDV Business Operations

When the JDV factory object receives an XML request, the JDV factory instance creates a new JDV instance. This JDV instance contains the method `execute`, which is called at this moment and is the entry point to the actual BL for the request.

The `execute` method is `public Response execute(Request<Document> request) throws JavaDataViewException;`

The handling steps of an incoming request are as follows:

##### 9.1.5.1 Handling CAI Data Validation

The incoming request of the JDV is an `org.w3c.dom.Document Request` object. There is currently no support in Wizard or any code base examples to use as template or guideline on CAI.

For example, Dynamic Activation transforms the following northbound CAI request:

```
CREATE:CUSTOMSUB:TRANSID,123456:
IMSI,26400000005010:KI,1234:ADKEY,509:
A38,2:A4,7:RID,12:CUSTOMATTRIBUTE,2;
```

to the following payload document on the request object available in the `execute` method of the CA:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<Create CUSTOMSUB xmlns="http://schemas.ericsson.com/ma/cai/1.0/">
  <IMSI>26400000005010</IMSI>
  <KI>1234</KI>
  <ADKEY>509</ADKEY>
  <A38>2</A38>
  <A4>7</A4>
  <RID>12</RID>
```



```
<CUSTOMATTRIBUTE>2</CUSTOMATTRIBUTE>
</Create CUSTOMSUB>
```

### 9.1.5.2 Handling CAI3G Data Validation

The incoming request of the JDV is an `org.w3c.dom.Document` Request object. The default class to validate the JDV inbound requests is `com.ericsson.jdv.common.validator.SchemaValidatorUtil`, located in the LIB-JDV-Public library.

The CAI3G data validation includes two steps:

1. Initiate a `SchemaValidatorUtil` instance.
2. Call the `SchemaValidatorUtil` instance to execute the validation.

The `SchemaValidatorUtil` instance loads the following two schema files for initiation:

- `cai3g1.2_provisioning.xsd`, common schema file shared by all JDVs
- `<CA-JDV-Project-Name>/src/main/resources/META-INF/webservice_provisioning_cai3g1.2/schemas`, JDV specified schema files

The `SchemaValidatorUtil` instance is initiated in the JDV factory object and called in the JDV object. For details, see the `execute` method of `<moname>JavaDataView.java` and the `newInstance` method of `<moname>JavaDataViewFactory.java`.

To enable multiple includes or imports for an xsd file, initiate the resource resolver as follows:

1. Initiate a `ResourceResolver` instance.
2. Set the multiple includes to true.

### 9.1.6 Transforming Request and Response Data

The default class that transforms the `Document` object to the internal format is `com.ericsson.jdv.common.transformer.XSLTTransformerUtil`, located in the LIB-JDV-Public library.

Transforming the XML data includes two steps:

1. Initiate a `XSLTTransformerUtil` instance.
2. Call the `XSLTTransformerUtil` instance to execute the transforming.

The `XSLTTransformerUtil` instance loads a standard XSL file for initiation. An XSL file, such as `JDV-Custom1-Provisioning/src/main/java/com/ericsson/jdv/custom1/handler/request-transformation.xsl`, is located in the JDV-Custom1-Provisioning project.



The `XSLTTransformerUtil` instance is initiated in the JDV factory object or JDV object and called for in the JDV object. For details, see the following example source codes in the JDV-Custom1-Provisioning project:

- The `newInstance` method of the JDV-Custom1-Provisioning/src/main/java/com/ericsson/jdv/custom1/Custom1JavaDataViewFactory.java file
- The `execute` method of the JDV-Custom1-Provisioning/src/main/java/com/ericsson/jdv/custom1/Custom1JavaDataView.java file

### 9.1.7

## BL Implementation for CUDB Layered NE Provisioning

For details about how the JDV class routes the MO operation, see the `execute` method of `JavaDataView` java file in the generated project.

For detailed API description and examples, see the design base project source code and java doc.

Figure 14 shows the CUDB layered data model.

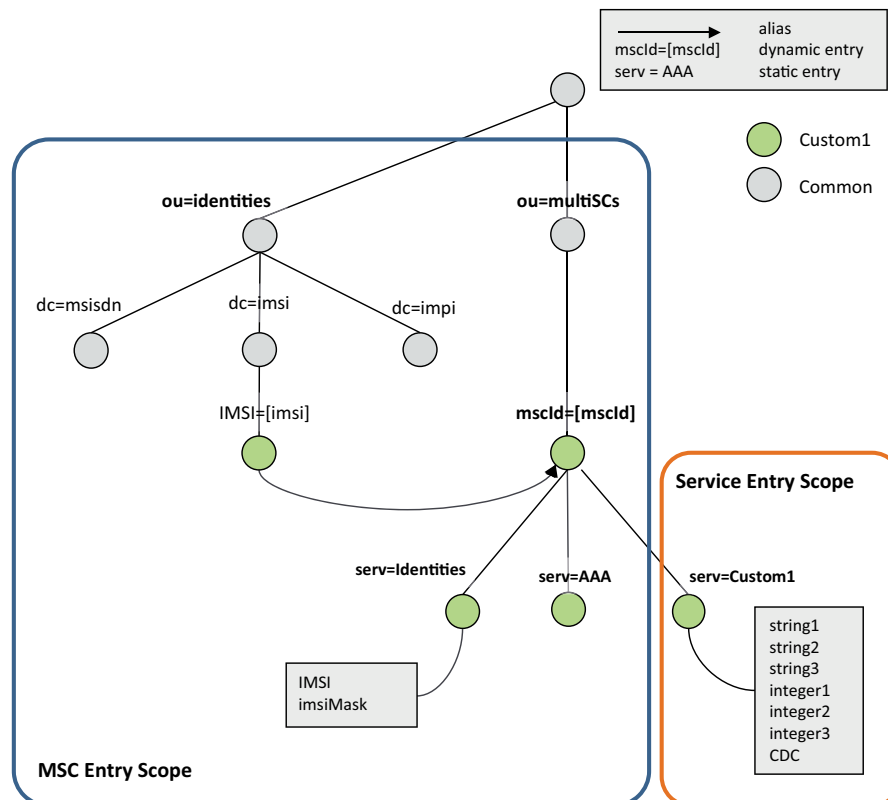


Figure 14 CUDB Layered Data Model

The `com.ericsson.jdv.cudb.datamodel.CUDBServiceHelper` class, located in the LIB-JDV-Public library, provides a formal way to create, read,



update, or delete the data in CUDB. All provisioning BL must be connected to CUDB through the `CUDBServiceHelper` class. The `CUDBServiceHelper` class has helper functions for the provisioning to the main parts of the CUDB data model.

The Multi Service Consumer (MultiSC) Entry Scope is common between services in the CUDB. This is handled inside the API. The Service Entry Scope is for the CA Service only, and is controlled inside the CA JDV.

For details about how the provisioning BL class handles the MO operation, see the `JDV-Custom1-Provisioning/src/main/java/com/ericsson/jdv/custom1/handler/CreateBusinessLogic.java` provisioning example file in the `JDV-Custom1-Provisioning` project.

### 9.1.8 Checking Whether CUDB Is in the Backup Process

The `JavaDataViewConfiguration.getProvisioningBlock` API, which is located in the `API-PAS` library, can check whether the CUDB is blocked for backup. If that is the case, no traffic is allowed towards the CUDB.

For details about how to use the `getProvisioningBlock` API, see the `verifyCUDB` method in the `JavaDataView.java` file in the generated project.

### 9.1.9 CUDB Data Inconsistency Alarm

The only alarm that is applicable to be sent from a JDV is the `PartiallyExecuteAlarm` that is caused by data inconsistency in CUDB.

The `com.ericsson.jdv.common.utilities.AlarmUtil` API can be used to send alarms and events when there is data inconsistency. This API provides the `sendPartiallyExecuteAlarms(EventType eventType, String handlerName, List<ResourceKey> ids, Exception e)` interface to send alarms and events to Ericsson SNMP Agent (ESA).

For details about how to use the `sendPartiallyExecuteAlarms` API, see the example JDV class files `CreateBusinessLogic.java` and `DeleteBusinessLogic.java`.

### 9.1.10 CUDB Data Inconsistency During the Create and Delete Operation

The `CUDBServiceHelper` class provides the following methods for handling data inconsistency when an error occurs during the Create operation:

- `getInconsistencyInCUDB` - This method checks whether the Create operations are partially executed in CUDB.
- `isCudbDataHasBeenChanged` - This method checks whether the Delete operations are partially executed in CUDB.



- `isRollback` - This method checks whether the current operation is a rollback operation or not.

The workflow of keeping data consistency when an error occurs during the Create operation is as follows:

- When creating the BL, use the `getInconsistencyInCUDb` method to check whether data is changed but not all data is committed. If the result is true, create a Delete request to invoke the Delete operation as shown in the following instance:  

```
Response response = executionManager.execute(deleteRequest);
```
- Check whether the rollback is successful according to the result of `response.getError`.
- Check if the Delete request is invoked by the rollback and the rollback operation is executed (using the `isCudbDataHasBeenChanged` and `isRollback` methods). Raise an alarm and throw an exception to the client user.

For details about how to handle data inconsistency during the Create operation, see the `CreateBusinessLogic.java` and `DeleteBusinessLogic.javaprovisioning` example files in the generated project.

### 9.1.11 Implementation of Customized Error Code

If there is a need for a customized error code, some Java code needs to be included in the Business Logic.

A `JavaDataViewException` needs to be thrown:

```
throw new
JavaDataViewException(JavaDataViewException.EXTERNAL_ERROR,
<Error message>,
    <Customized AdditionalClassifiedError with customized error code>, ex);
```

## 9.2 CA Development for Monolithic Data Provisioning

### 9.2.1 Preparations

Prepare for the creation of the new CA JDV by analyzing the application to be provisioned and its data model. CA developers need to decide the application type, the application data model details, possible data model differences between the northbound and southbound interfaces, and so on.

## 9.2.2 Project Creation

Use the wizard in Eclipse to create a new monolithic JDV.

1. Start Eclipse and select **File > New > Other** from the menu bar.
2. Select the type of wizard to use: **EMA Customer Adaptation > Monolithic JDV Wizard** and click **Next**.
3. Specify the information for the JDV project to create and click **Next**.

Parameter	Description
Project name	The name of the project to be created for Monolithic JDV.
Package name	The Java package structure.
Use default location	If checked, the system default project name and location is used.
Location	The directory where the new project is stored.



Parameter	Description
Version	Version of build to be used when packaging CA.
Network	Monolithic

4. Specify the information for the JDV data model and click **Next**.

**New JDV Mono Project**

**Define a JDV Data Model**

Enter a MO name

MO Name: Custom3

Namespace: http://schemas.ericsson.com/ma/CA/Custom3

MOType: Custom3@http://schemas.ericsson.com/ma/CA/Custom3

Operations:

☒ Create ☒ Get ☒ Set ☒ Delete

Routing:

NE Type: CLASSIC\_CUSTOM3

< Back Next > Finish Cancel

Parameter	Description
MO Name	The specific MO name. Must start with an alphabetical character. Must be in capital letters if using CAI northbound interface.
Namespace	The namespace to use for the MO. <ul style="list-style-type: none"> <li>For CAI, must begin with <code>http://schemas.ericsson.com/ma/cai/1.0/</code></li> <li>For CAI3G, recommend beginning with <code>http://schemas.ericsson.com/ma/CA/</code></li> </ul>
MO Type	The MO Name and Namespace together form MO Type.

Parameter	Description
Operations	Select which operations to create.
Routing	Specify the name of the NE type to use.

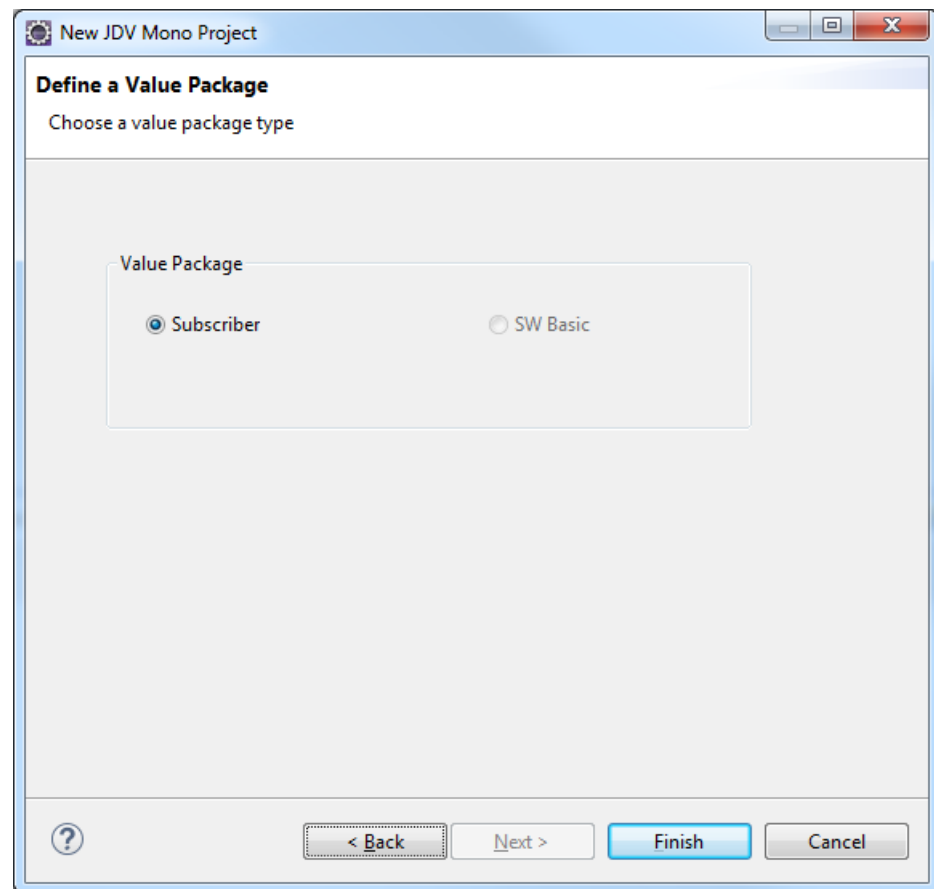
- Specify configuration parameters for the JDV and click **Next**.

Parameter	Description
Configuration parameters	Add list of parameters that can be configured from the GUI to use in the JDV. <sup>(1)</sup>

(1) Do not use Java reserved words, follow Java variable naming rules.

- Select what value package to use for this JDV, default value is **subscriber**, and click **Finish**.





Parameter	Description
Value Package	Select which value package to use for this JDV. <code>SW Basic</code> is only used for Subscriber view JDVs. For more details about what value package to use, see Section 10.1.2 on page 100.

Now two new projects are generated, one service JDV and one resource JDV.

### 9.2.3 Interface Specification

The procedure of defining a JDV data model for monolithic data provisioning is as follows:

- Customize the northbound interface data model:
  - Find the northbound schema and WSDL files in the `src/main/resources/META-INF/webservice_provisioning_cai3g1.2` directory of the CA JDV project `JDV-Custom3-Resource-Provisioning`.

- For CAI interfaces, `javadataview-descriptor` must:
  - Have triggers with MO in capital letters.
  - Use the namespace `http://schemas.ericsson.com/ma/cai/1.0/`

Example of valid CAI trigger of `javadataview-descriptor.xml`:

```
<trigger value="ns:CreateEIRSUB"/> where
xmlns:ns="http://schemas.ericsson.com/ma/cai/1.0/"
```

- For CAI3G 1.2 interfaces, define the namespace, operations, and other related parameters of northbound requests in the schema and WSDL files according to *Generic CAI3G Interface 1.2*, Reference [3] .
- State the names and constants of the application:
  - Name of the NE type to use in the Dynamic Activation GUI, Routing configuration dialog. Refer to `NETYPE` variable in `Constants`.
  - Name of the value package to use in license control. Refer to `valuePackage` in `javadataview-descriptor`.

For more details about what value package to use, see Section 10.1.2 on page 100.

## 9.2.4

### Define the JDV Access Control Model

The access control model includes the configuration of JDV MO, attributes, and operations that fulfill the access control criteria in `managedobjects-descriptor.xml` file. Dynamic Activation applies user authority on both MO and attribute level with these criteria and rules defined in GUI, see Section 11.6 on page 112.

The procedure of defining a JDV access control model is as follows:

1. Find the MO descriptor file in the `src/main/resources/META-INF/` directory of the CA JDV project.
2. Define a JDV group with MO, attributes, operations, and other related parameters to be shown on the access control GUI.

## 9.2.5

### Define JDV Resource Key Attribute (Optional)

The MO identities can be used for routing the requests to the target NE. If an MO identity is defined as "JDV Resource Key attribute" in the MO descriptor file, this attribute can be selected in the **Attribute Name** field in the **Routing** tab in GUI.

If no JDV resource key attribute has been defined, `imsi` and `msisdn` are listed in the **Attribute Name** field.

The procedure of defining a JDV Resource Key attribute is as follows:



**Note:** Ensure that the NE type of the CA JDV project is specified before defining the JDV Resource Key attribute.

1. Find the MO descriptor file `managedobjects-descriptor.xml` in the `src/main/resources/META-INF/` directory of the CA JDV project.
2. Set the resource key attribute by editing the value of class to `com.ericsson.dve.common.managedobject.ResourceKeyAttribute`.

```
<bean id="userId" class="com.ericsson.dve.common.managedobject.ResourceKeyAttribute">
  <property name="name"><value>custom3:userId</value></property>
  <property name="moGroupNames"><list><value>CUSTOM3</value></list></property>
  <property name="dataType"><value>NUMERIC</value></property>
</bean>
```

**Example 1** *An Example of Defining a JDV Resource Key Attribute in MO Descriptor File*

This JDV Resource Key attribute shows on GUI as below:

The screenshot shows the 'Ericsson Multi Activation Operation & Management' interface. The 'Routing' tab is selected under 'Network Elements'. The configuration is for 'CLASSIC\_CUSTOM3'. In the 'Routing Methods' section, a list contains 'NumberRangeRouting', 'cust3\_st,userId:100:200', and 'cust3\_st,userId:20:30'. Below this, the 'Network Element (Group)' is 'cust3\_st'. The 'Attribute Name' dropdown is circled in red and shows 'userId'. The 'Start range' and 'Stop range' fields are empty. An 'Add' button is to the right. At the bottom are 'Apply' and 'Cancel' buttons.

Figure 15 Example of JDV Resource Key Attribute Shows on GUI

## 9.2.6

### Define JDV Loose Error Handling (Optional)

The JDV Loose Error Handling includes loosable errors and pre-defined loose error rules which are used to support the Configurable Loose Error Handling in GUI.



### 9.2.6.1 Configure Errors

Do the following to add errors that can be configured in the **Loose Error Handling** tab in GUI.

1. Find the Loose Error Handling descriptor file `looseerror-descriptor.xml` in the `src/main/resources/META-INF/` directory of the CA Resource JDV project.
2. Add errors with name, error code and error description to be shown on the **Loose Error Handling** tab in GUI.

```
<error name="UserAlreadyExistError">
  <errorCode>1024</errorCode>
  <description>User Already Exists.</description>
</error>
```

*Example 2 An Example of Adding an Error in Loose Error Handling Descriptor File*

### 9.2.6.2 Configure Pre-defined Loose Error Rules

Do the following to configure pre-define loose error rules that can be configured in the **Loose Error Handling** tab in GUI.

1. Find the Loose Error Handling descriptor file `looseerror-descriptor.xml` in the `src/main/resources/META-INF/` directory of the CA Resource JDV project.
2. Configure the pre-defined loose error rules with name, description, inbound triggers, outbound triggers and error codes to be shown on the **Loose Error Handling** tab in GUI.

```
<looseErrorRule>
  <name>rule to loose user already exists error</name>

  <description>To loose error : 1024 user already exists. And some other errors.</description>

  <inboundTriggers>
    <trigger name="custom3:CreateCustom3User" />
    <trigger name="custom3:SetCustom3User" />
  </inboundTriggers>

  <outboundTriggers>
    <trigger name="custom3:CreateCustom3User_RESOURCE" />
  </outboundTriggers>

  <errorCodes>
    <error errorCode="1024"></error>
  </errorCodes>
</looseErrorRule>
```

*Example 3 An Example of Configuring a Pre-defined Loose Error Rule in Loose Error Handling Descriptor File*



## 9.2.7 Handling the JDV Business Operations

When the JDV factory object receives an XML request, the JDV factory instance creates a JDV instance containing the method `execute`. This method is the entry point to the actual BL for the request.

The `execute` method is `public Response execute(Request<Document> request)` throws `JavaDataViewException`;

### 9.2.7.1 Handling CAI Data Validation

See Section 9.1.5.1 on page 51.

### 9.2.7.2 Handling CAI3G Data Validation

The incoming request of the JDV is an `org.w3c.dom.Document` Request object. The default class to validate the JDV inbound requests is `com.ericsson.jdv.common.validator.SchemaValidatorUtil`, located in the LIB-JDV-Public library.

The CAI3G data validation includes two steps:

1. Initiate a `SchemaValidatorUtil` instance.
2. Call the `SchemaValidatorUtil` instance to execute the validation.

The `SchemaValidatorUtil` instance loads the following two schema files for initiation:

- `cai3g1.2_provisioning.xsd`, common schema file shared by all JDVs
- `<CA-JDV-Project-Name>/src/main/resources/META-INF/webservice_provisioning_cai3g1.2/schemas`, JDV specified schema files

The `SchemaValidatorUtil` instance is initiated in the JDV factory object and called in the JDV object. For details, see the `execute` method of `<moname>JavaDataView.java` and the `newInstance` method of `<moname>JavaDataViewFactory.java`.

To enable multiple includes or imports for an XSD file, initiate the resource resolver as follows:

1. Initiate a `ResourceResolver` instance.
2. Set the multiple includes to true.

## 9.2.8 Transforming Request and Response Data

The default class that transforms the `Document` object to the internal format is `com.ericsson.jdv.common.transformer.XSLTTransformerUtil`, located in the LIB-JDV-Public library.



Transforming the XML data includes two steps:

1. Initiate a `XSLTTransformerUtil` instance.
2. Call the `XSLTTransformerUtil` instance to execute the transforming.

The `XSLTTransformerUtil` instance loads a standard XSL file for initiation. An XSL file, such as `JDV-Custom3-Provisioning/src/main/java/com/ericsson/jdv/custom3/handler/request-transformation.xsl`, is located in the `JDV-Custom3-Provisioning` project.

The `XSLTTransformerUtil` instance is initiated in the JDV factory object or JDV object and called for in the JDV object. For details, see the following example source codes in the `JDV-Custom3-Provisioning` project:

- The `newInstance` method of the `JDV-Custom3-Provisioning/src/main/java/com/ericsson/jdv/custom3/Custom3JavaDataViewFactory.java` file
- The `execute` method of the `JDV-Custom3-Provisioning/src/main/java/com/ericsson/jdv/custom3/Custom3JavaDataView.java` file

## 9.2.9 BL Implementation for Monolithic NE Provisioning

For details about how the JDV class routes the MO operation, see the `execute` method in the example `JDV-Custom3-Resource-Provisioning/src/main/java/com/ericsson/jdv/custom3/Custom3JavaDataView.java` JDV file in the `JDV-Custom3-Resource-Provisioning` project.

The `JDV-Custom3-Resource-Provisioning` project, located in the `LIB-JDV-Public` library, provides a formal way to CRUD the data in the monolithic NEs. All provisioning BL must be connected to the NEs through the `CustomJCAHelper` class.

For details about how the provisioning BL class handles the MO operation, see the `JDV-Custom3-Resource-Provisioning/src/main/java/com/ericsson/jdv/custom3/handler/CreateBusinessLogic.java` provisioning example file in the `JDV-Custom3-Resource-Provisioning` project.

For detailed API description and examples, see the design base project source code and java doc.

## 9.2.10 Implementation of Customized Error Code

If there is a need for a customized error code, some Java code needs to be included in the Business Logic.

A `JavaDataViewException` needs to be thrown:



```
throw new  
JavaDataViewException(JavaDataViewException.EXTERNAL_ERROR,  
<Error message>,  
    <Customized AdditionalClassifiedError with customized error code>, ex);
```

## 9.3 CA Development for Java Connector Architecture

### 9.3.1 Preparations

Prepare for the creation of the new CA JCA by analyzing the JCA connection by collecting and clarifying requirements. CA developers need to analyze NE protocols and message formats. CA developers design and customize the JCA based on this information.

### 9.3.2 Project Creation

Use the wizard in Eclipse to create a JCA.

1. Start Eclipse and select **File > New > Other** from the menu bar.
2. Select the type of wizard to use: **EDA Customer Adaptation > JCA Wizard** and click **Next**.
3. Specify the information for the JCA project to create and click **Next**.



**New JCA Project**

**Create a JCA Project**

Enter a project name

Project name: JCA-Custom3

Package name: com.ericsson

☒ Use default location

Location: C:\EMA\CA Browse...

Version: 1.0.0

< Back Next > Finish Cancel

Parameter	Description
Project name	The name of the project to be created for JCA.
Package name	The Java package structure.
Use default location	If checked, the system default project name and location are used.
Location	The directory where the new project is stored.
Version	Version of build to be used when packaging CA.

4. Specify the information for link protocol and click **Finish**.





**New JCA Project**

**Define a Link Protocol**

Enter a link name

Display Name: CustomLink

Description: This is a CustomLink connection

Vendor Name: Ericsson AB

Connection properties:

Name	Type
	String
host	String
port	Integer
secure	boolean

☒ Http based template

[? Help](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)

Parameter	Description
Display Name	The JCA connector name that is displayed in Dynamic Activation GUI.
Description	The description of the JCA connector.
Vendor Name	The name of the JCA connector vendor.
Connection properties	The hostname and the port for the connection. <sup>(1)</sup>
HTTP based template	If checked, HTTP based template is used.

(1) Do not use Java reserved words, follow Java variable naming rules.



### 9.3.3 Configuring the Resource Adapter Description File

Each JCA connector must have a resource adapter description file to define all the key information and key interfaces. The default description filename is `ra.xml`, which is located in the `src/main/rar` directory. The following table contains mandatory parameters defined in this file.

*Table 8 Parameters in ra.xml*

Name	Description
<code>display-name</code>	The JCA connector name that is displayed in the Dynamic Activation GUI
<code>vendor-name</code>	The name of the JCA connector vendor
<code>description</code>	The description of the JCA connector
<code>eis-type</code>	The Enterprise Information Systems (EIS) type of this JCA connector. The EIS type name of a CA JCA must start with <code>CA-</code> to support logging in to the Dynamic Activation processing log file.
<code>resourceadapter-version</code>	Resource adapter version; the default value is 1.0
<code>managedconnectionfactory-class</code>	The <code>ManagedConnectionFactory</code> class that is used in the JCA connector. This class must be implemented in the CA JCA connector.
<code>config-property</code>	The JCA connector property that is used to initialize a new connector instance.  More than one <code>config-property</code> is available in a CA JCA connector.
<code>config-property.description</code>	The property display information, which is shown when creating NE in the Dynamic Activation GUI.
<code>config-property.config-property-name</code>	The property name, which is used as a java bean name.
<code>config-property.config-property-type</code>	The property type, of which the value is a basic java data type class.
<code>connectionfactory-interface</code>	The default JCA <code>ConnectionFactory</code> interface. There is no need to change the default value <code>javax.resource.cci.ConnectionFactory</code>
<code>connectionfactory-impl-class</code>	The default implementation class of the <code>ConnectionFactory</code> class in the <code>LIB-JDV-Public</code> library. There is no need to change the default value <code>com.ericsson.jca.common.cci.ConnectionFactoryImpl</code> .
<code>connection-interface</code>	The default JCA <code>Connection</code> interface. There is no need to change the default value <code>javax.resource.cci.Connection</code> .
<code>connection-impl-class</code>	The default implementation class of the <code>Connection</code> class in the <code>LIB-JDV-Public</code> library. There is no need to change the default value <code>com.ericsson.jca.common.cci.ConnectionImpl</code> .

### 9.3.4 Implementing ManagedConnectionFactory

To implement the `ManagedConnectionFactory` interface:



- Define a subclass of `com.ericsson.jca.common.spi.AbstractManagedConnectionFactoryImpl`, which is located in the `LIB-JDV-Public` library.
- Provide setter methods for the `config-property` variable that is defined in `ra.xml`. The method names must meet the naming convention of java-bean setter methods. For example, if `Host` is a name of a `config-property` of type `java.lang.String`, the name of the setter method is `setHost`. The argument of the method is `Host` of type `java.lang.String`, and the method is located in the subclass of `AbstractManagedConnectionFactoryImpl`.

For further information regarding the naming convention to use, see `ra.xml` file and `ManagedConnectionFactoryImpl` in the `Custom3` design base project.

- The subclass of `AbstractManagedConnectionFactoryImpl` implements the following abstract methods of `AbstractManagedConnectionFactoryImpl`:
  - `getConnectionRequestInfo`  
Get the connection request information.
  - `getInvalidConnections`  
This method returns a set of invalid `ManagedConnection` objects selected from a specified set of `ManagedConnection` objects. This method is used for handling the heartbeat functionality.
  - `createManagedConnection`  
Create a managed connection with provided connection requests.

### 9.3.5 Implementing ManagedConnection

To implement the `ManagedConnectionFactory` interface:

- Define a subclass of `com.ericsson.jca.common.spi.AbstractManagedConnectionImpl`, which is located in the `LIB-JDV-Public` library.
- The subclass of `AbstractManagedConnectionImpl` implements the abstract method `executeMappedRecord` of `AbstractManagedConnectionImpl`.

The `executeMappedRecord` method receives the `procLogId` and the input variable `MappedRecord` that contains the commands sent by a JDV. This method must implement the logic to connect real NE resource and send a corresponding message to those resources. When a response is sent back from the real NE resource, the response must be stored into the output variable `MappedRecord` that returns the response message to the JDV.



- During the command handling process, processing log points must be implemented. Create the `ProcLogEntry` object to record command and response. After completing the command-handling process and before a response is returned, CA developers must complete the `ProcLogEntry` according to the real result:
  - If the real NE resource handles the command successfully, call the `getProcLogHelper().completeSuccessfulProcLog` method.
  - If the real NE resource fails to handle the command, call the `getProcLogHelper().completeFailedProcLog` method.
- To trace the southbound request logging by context, `ManagedConnection` implementation needs to get the context value and record the context into the `proclog` instance.

For Example:

```
String instance = "context=" + getValue(MappedRecord, "Context", String.class);
getProcLogHelper().completeSuccessfulProcLog(procLogEntry,
operation.toString(), JCA_HTTP, getRequestString(httpRequest),
getResponseString(httpResponse), connectionRequestInfo.getDataRepositoryName());
```

### 9.3.6 Implementing StatisticsReporter for Dashboard

To get an instance of `StatisticsReporter` to a JCA:

- In the `ManagedConnectionFactory` class, implement the `StatisticsHandler` interface, and the method of the `setStatisticsHelper(StatisticsHelper statisticsHelper)` interface.
- The `StatisticsHelper` interface contains one method named `createStatisticsReporter(String dataRepositoryName)`. Use this method to pass along an instance of `StatisticsReporter` to the `ManagedConnectionImpl` instance.

For Example:

```
ManagedConnectionImpl(customlinkConnectionInfo, procLogHelper,
statisticsHelper.createStatisticsReporter(customlinkConnectionInfo.getDataRepositoryName()));
```

**Note:** The name, when creating the reporter, is the name that will show what NE the statistics is reported from. Make sure to get the right name for the repository.

The `StatisticsReporter` instance which is used for reporting statistics, is now created in the `ManagedConnectionImpl` class.

To get an instance of `StatisticsReporter` to a Connector:



- In the `create(ConnectorContext connectorContext)` method use the `ConnectorContext` object to get an instance of `StatisticsHelper`
- Use the `createStatisticsReporter(instanceName)` method to create an instance of `StatisticsReporter` and pass this instance to `ConnectorImpl`.

**Note:** To get the repository name, implement the annotation `@InstanceName`.

Functionality of the `StatisticsReporter` and how to take measurements.

- The `StatisticsReporter` has two methods:

```
reportSuccessful(int responseCode, Duration
responseTime);
```

and

```
reportFailed(int responseCode, Duration responseTime);
```

- To get response time in the right format use the helper class `StatisticsTimer`, and its two methods `start()`, and `stop()`;

Call the `start()` method when wanting to start taking a measurement, and the `stop()` method when wanting to stop the same measurement. The time is then returned as a duration.

- The `stop()` method is preferably called as an input to the report methods.

For Example:

```
reportFailed(errorCode, statisticsTimer.stop());
statisticsReporter.reportSuccessful(responseCode, statisticsTimer.stop());
```

## 9.4 CA Development for Cluster Strategy

This section contains information about Cluster Strategy development. The CA Cluster Strategy can be developed in two ways:

- Implementing the CA Cluster Strategy by wizard, see Section 9.4.1 on page 71.
- Customizing off-the-shelf Cluster Strategy, see Section 9.4.3 on page 86.

## 9.4.1 Implementing CA Cluster Strategy by Wizard

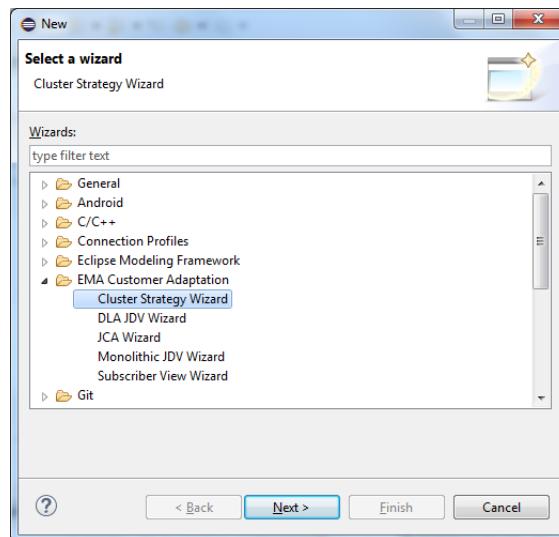
### 9.4.1.1 Preparation

Prepare for the creation of the new CA cluster strategy by analyzing the application to be provisioned and its data model.

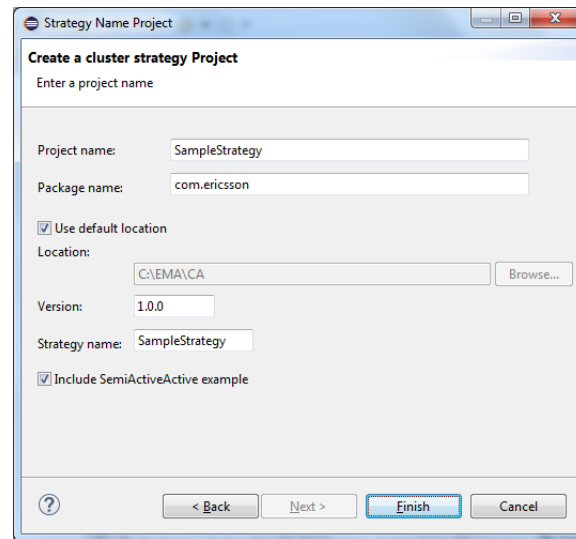
### 9.4.1.2 Creating Project

Use the wizard in Eclipse to create a cluster strategy project.

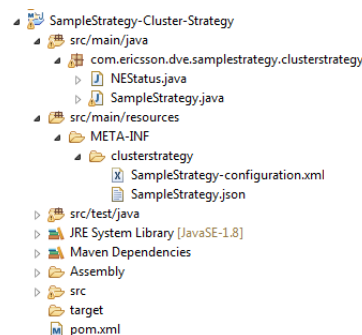
1. Start Eclipse and select **Menu > File > New > Others....**



2. Select the type of wizard to use: **EMA Customer Adaptation > Cluster Strategy Wizard** and click **Next**. Then enter the project name, version and Strategy name and click **Finish**.



3. The project structure generated by Cluster Strategy wizard is as the following:



The wizard contains:

- `SampleStrategy-configuration.xml` - The configuration file of the cluster strategy properties. CA developers only modifies this files when the customized attributes have to be defined. For detailed information, see Section 9.4.1.3 on page 74.
- `SampleStrategy.json` - The fields and their properties to be displayed on Cluster Strategy GUI. For detailed information, see Section 9.4.1.4 on page 75.
- `SampleStrategy.java` - The main JDV class files of the Cluster Strategy. For detailed information, see Section 9.4.1.5 on page 83.
- `NEStatus.java` - The auxiliary class that contains the runtime NE connection state. NE connection state is used by some of the Cluster Strategy implementation. No change is needed.

**Note:** It is recommended to use the off-the-shelf cluster strategy as a reference to have a better understanding of the cluster strategy project.

The following figure shows how the implementation to be displayed on GUI.

#### SampleStrategy-configuration.xml

```

....
<cluster-strategy-name>SampleStrategy</cluster-strategy-name>
....
<config-property>
  <config-property-name>DataRepositories</config-property-name>
  .....
  </config-property-description>
</config-property>
<config-property>
  <config-property-name>Attribute1</config-property-name>
  ...
</config-property>
<config-property>
  <config-property-name>Attribute2</config-property-name>
  ...
</config-property>
</cluster-strategy-configuration>
</cluster-strategy>

```

#### SampleStrategy.json

```

{
  "types": "SampleStrategy",
  ....
  "membershipConfigurations": {
    ...
  },
  "genericConfigurations": [
    ...
  ]
}

```

Create Cluster Strategy Instance

Name

Strategy Type

Network Elements

Network Element	Provisioning State	role
<input type="text" value="Please select"/>	<input type="text" value="Please select"/>	<input type="text" value="Please select"/>

Name	Provisioning State	role
Attribute2		
Attribute1		

Figure 16 Relationship between Code and GUI

### 9.4.1.3 Implementing Cluster Strategy Properties

Configure the SampleStrategy-configuration.xml file if there is any additional attributes required. Otherwise, no change is needed.

The following is an example of adding attributes Attribute1 and Attribute2 in SampleStrategy-configuration.xml.





```
<?xml version="1.0" encoding="UTF-8"?>
  <cluster-strategy xmlns="http://com.ericsson/clusterstrategies/1.0">
    <cluster-strategy-name>SampleStrategy</cluster-strategy-name>
    <cluster-strategy-implementation>com.ericsson.dve.samplestrategy.clusterstrategy.SampleS
    <cluster-strategy-configuration>
      <config-property>
        <config-property-name>DataRepositories</config-property-name>
        <config-property-type>java.util.ArrayList</config-property-type>
        <config-property-description>Strategy specific membership configuration.</config
      </config-property>
      <config-property>
        <config-property-name>Attribute1</config-property-name>
        <config-property-type>java.lang.String</config-property-type>
        <config-property-description>attribute1 parameter</config-property-description>
      </config-property>
      <config-property>
        <config-property-name>Attribute2</config-property-name>
        <config-property-type>java.lang.boolean</config-property-type>
        <config-property-description>attribute2 parameter</config-property-description>
      </config-property>
    </cluster-strategy-configuration>
  </cluster-strategy>
```

#### Example 4 Adding Attributes in SampleStrategy-configuration.xml

The following table gives the description of Config-property.

**Table 9 Configuration Properties**

Configuration Property	Description
config-property-name	The name of the attribute. This name must follows the java variable naming policy, except the first letter must be capitalized.
config-property-type	The data type of the attribute. Supported type: String and Boolean
config-property-description	The description of the attribute.

#### 9.4.1.4

#### Implementing Cluster Strategy GUI

SampleStrategy.json defines the elements to be displayed on the Cluster Strategy GUI. Update the file if developers need to modify the existing parameters or add new additional parameters. This configuration is in .json format.

The following is an example of configuring a sample strategy in SampleStrategy.json.



```
{
  "types": "SampleStrategy",
  "locales": {
    "en-us": {
      "networkElements": "Network Element",
      "provisioningStates": "Provisioning State",
      "Inactive": "Inactive",
      "Active": "Active",
      "role": "Role",
      "Attribute2": "Attribute2",
      "Attribute1": "Attribute1"
    }
  },
  "membershipConfigurations": {
    "items": [
      {
        "name": "networkElements",
        "type": "selectbox",
        "itemUrl": "/clusterstrategy-backend/cs/cai3g/ne/list"
      },
      {
        "name": "provisioningStates",
        "type": "selectbox",
        "items": ["Active", "Inactive"]
      },
      {
        "name": "role",
        "type": "selectbox",
        "items": ["primary", "secondary", "third"]
      }
    ],
    "policies": []
  },
  "genericConfigurations": [
    {
      "name": "Attribute2",
      "type": "switcher"
    },
    {
      "name": "Attribute1",
      "type": "input"
    }
  ]
}
```

#### Example 5 SampleStrategy.json

The following table gives a detailed description of each parameters.

Table 10 Parameter Description

Section	Parameter	Description
types	-	The name of this Cluster Strategy. No change is needed.
locales	-	<p>The elements to be displayed in the <b>Create Cluster Strategy Instance</b> window.</p> <p>If a additional attribute need to be added, then the field name of this additional attribute must be added in this section.</p> <p>For example, additional attribute - "Attribute2": "Attribute2"</p>



Section	Parameter	Description
memberShipConfiguration	-	The membership of the NEs in the cluster. The section is display in <b>Network Elements</b> field.
	networkElements	The NE list that available for the cluster strategy. No change is needed.
	provisioningState	The candidate values of the provisioning status. Update this field according to the cluster strategy requirement. provisioningState cannot be deleted.
	role	The candidate values of role. Update this field according to cluster strategy requirement. This parameter can be deleted if no value needs to be defined.
genericConfiguration	-	The additional attributes of the Cluster Strategy. The section is displayed in <b>Properties</b> field.
	name	The attribute name which is same as the one defined in SampleStrategy-configuration.xml
	type	The type of the additional attribute. The value type of this field needs to be aligned with the property type defined in SampleStrategy-configuration.xml Supported types are:
	input	The value type of this field is string. The syntax is: <pre>{   "name": "Attribute1",   "type": "input",   "isRequired": "true",   "validators": [{     "operation": "match",     "value": "^[0-9]*\$",     "errorMessage": "Attribute1NotMatchNumber"   }] }</pre> The parameter name and type must be provided. The parameter isRequired or validators are optional. When isRequired is defined as ture, user must gives value for this field when configuring the cluster in GUI.
	selectbox	The value type of this field is string. The syntax of selectbox is: <pre>{   "name": "Attribute3",   "type": "selectbox",   "items": ["Master", "Slave"] }</pre> The parameter name, type and item must be provided. User must gives value for parameters in type selectbox when configuring the cluster in GUI.
	switcher	The value type of this field is boolean. The syntax of switcher is: <pre>{   "name": "Attribute2",   "type": "switcher" }</pre> The parameter name and type must be provided.



#### 9.4.1.4.1 Advanced Configuration - Validator

CA developers can define one or more validators of NE items and attributes in `genericConfigurations` to restrict the input value. Properties `operation`, `value` and `errorMessage` are required for a completely validator. The validator checks the input with defined value by using the provided operation.

*Table 11 Parameter*

Properties	Type	Description
operation	string	The type of build-in validator operation. It could be match, gt, lt, eq, gte, lte, and so on.
value	string or number	The type of the value is determined by the type of operation. <ul style="list-style-type: none"><li>• If operation is set to gt, lt, gte or lte, value should be a number.</li><li>• If operation is set to match, value should a regex string.</li><li>• If operation is set to eq, value should be a string.</li><li>• If operation is set to isTrue, no need to define value.</li></ul>
errorMessage	string	The index of the error message. The detailed information is defined according to the local settings. It shows when the validation is not passed.

Then following is an example of configuring a validator for `Attribute1`.

```
{
  "name": "Attribute1",
  "type": "input",
  "validators": [{
    "operation": "match",
    "value": "^[0-9]*$",
    "errorMessage": "Attribute1NotMatch"
  }]
}
```

#### *Example 6 Validator of Attribute1*

The above validator checks whether the input value matches regex or not. If the input value does not match the number value `^[0-9]*$`, the error message `Attribute1NotMatch` will be returned on GUI.

#### 9.4.1.4.2 Advanced Configuration - Policy

CA developers can define the polices in `membershipConfigurations` to restrict the user operation of cluster membership attributes in GUI. When user changes the cluster configurations in GUI and this change matches the policy condition, then the policy checker will be invoked. If specific condition is not defined, then the checker will be invoked in all changes.



Properties conditions and checkers are required for each policy.

**Table 12 Parameters**

Properties	Sub-properties	Sub-properties type	Description
conditions	attributeName	string	The condition check is applied on which item of the cluster membership attribute.
	operation	string	The type of build-in validator operation. It could be match, gt, lt, eq, gte, lte, and so on.
	value	string or number	<p>The type of the value is determined by the type of operation.</p> <ul style="list-style-type: none"> <li>• If operation is set to gt, lt, gte or lte, value should be a number.</li> <li>• If operation is set to match, value should be a regex string.</li> <li>• If operation is set to eq, value should be a string.</li> <li>• If operation is set to isTrue, no need to define value.</li> </ul>
checkers <sup>(1)</sup>	checkerType	string	The type of the policy checker. The supported types are QUEUE_EMPTY, UNIQUE_VALUE, ROW_CONSTRAINT, or MULTI_ROW_CONSTRAINT.
	args	object	It is an optional parameter which is determined by the type of checker

(1) The detailed description of checkers is described in below.

The supported checker type are:

- QUEUE\_EMPTY

It checks whether the requests of the selected NE is already queued in the processing queue or not. This checker is only invoked when editing the existing cluster strategy instance. This checker is not applicable for creating a new cluster instance on GUI. No specific args is needed for QUEUE\_EMPTY.

The following is an example of configuring the QUEUE\_EMPTY :

```
{ "checkerType": "QUEUE_EMPTY" }
```

- ROW\_CONSTRAINT



It checks whether the input value for this NE matches the cluster strategy rules. It is used with the conditions. The specific condition as args and errorMessage need to be defined for ROW\_CONSTRAINT.

Then following is an example of configuring the ROW\_CONSTRAINT.

```
"policies": [{
  "conditions": [{
    "attributeName": "primaryNe",
    "operation": "eq",
    "value": "Yes"
  }],
  "checkers": [{
    "checkerType": "ROW_CONSTRAINT",
    "args": {
      "conditions": [{
        "attributeName": "provisioningStates",
        "operation": "eq",
        "value": "On"
      }]
    }
  ]
}]
}]
```

The above checker is used in the condition when modifying an NE that role is primaryNE and the Provisioning State is On. If user modified the Provisioning State to other value, such as off, an error message will be returned as below.

Network Elements

---

Network Element  
HLR01 ▼

Provisioning State  
Off ▼

Primary Network Element  
Yes ▼

❗ Provisioning State should be On when it is primary network element

↓ Add

- UNIQUE\_VALUE

It checks whether the input value of a specific attribute is unique or not. The attribute name must be given in the UNIQUE\_VALUE.

Then following is an example of configuring the ROW\_CONSTRAINT.



```
{
  "policies": [{
    "conditions": [{
      "attributeName": "primaryNe",
      "operation": "eq",
      "value": "Yes"
    }],
    "checkers": [{
      "checkerType": "UNIQUE_VALUE",
      "args": {
        "attributeNames": ["primaryNe"]
      }
    },
    {
      "checkerType": "QUEUE_EMPTY"
    }
  ]
}]
}
```

The above checker checks that only one NE can be set as primary role in a cluster. The checker is invoked when customer creates or sets an NE as primary role. The checker validates all the NEs, if any NE has already been set to primaryNE, an error message will be returned as below.

primaryClusterSample

Strategy Type  
PrimaryBackup

Network Elements

Network Element  
HLR02

Provisioning State  
On

Primary Network Element  
Yes

❗ This value of Primary Network Element should be unique

Add

<input type="checkbox"/>	Name	Provisioning State	Primary Network E...
<input type="checkbox"/>	HLR01	On	Yes

Properties

Auto Off-Store Disabled

Create Cancel

- MULTI\_ROW\_CONSTRAINT

It checks whether the configuration for multiple NEs meet the defined constraints or not.

The supported constraint type is AT\_LEAST\_ONE. It checks whether the defined condition is configured for an NE at least once.

The following is an example of configuring the MULTI\_ROW\_CONSTRAINT.



```
{
  "policies":
  [{
    "conditions": [],
    "checkers": [{
      "checkerType": "MULTI_ROW_CONSTRAINT",
      "args": {
        "constraints": [{
          "conditions": [{
            "attributeName": "order",
            "operation": "eq",
            "value": "0"
          }],
          "type": "AT_LEAST_ONE"
        }]
      }
    }]
  }]
}
```

The above checker checks that at least one of the NE in a cluster strategy has been configured with `order` equals 0. Otherwise, an error will be returned as below.

#### Create Cluster Strategy Instance

Name

Strategy Type

Network Elements

Network Element

Provisioning State

Order

<input type="checkbox"/>	Name	Provisioning State	Order
<input type="checkbox"/>	AIR01	N/A	1

There must be Order value 0 as primary node in multiple air cluster.

#### 9.4.1.4.3 Advanced Configuration - Sub-attribute

CA developers can define the sub-attribute for any attribute in `genericConfigurations`. And the `formatter` can be used to format the sub-attributes inputs. The value of the parameters are combined to a string with the separator defined in `formatter` and send to Dynamic Activation.

The following is an example of configuring the Sub-attribute.





```
{
  "genericConfigurations": [{
    "name": "AutoOffStore",
    "type": "switcher",
    "formatter": {
      "type": "CONCAT",
      "args": {
        "separator": ";"
      }
    },
    "subAttributes": [{
      "name": "MinimumDuration",
      "type": "input"
    }]
  }]
}
```

In the above example, a sub-attributes **MinimumDuration** has been added for attribute **AutoOffStore**. If user enables the **AutoOffStore** and set **MinimumDuration** to 40, a string with value `true;40` is sent back to Dynamic Activation.

#### 9.4.1.4.4 Advanced Configuration - Pre-condition

CA developers can define the pre-condition of the sub-attributes in `genericConfigurations`. If the input matches the pre-condition, the field of the sub-attribute is enabled or shown on the GUI.

For example:

```
{
  "genericConfigurations": [{
    "name": "AutoOffStore",
    "type": "switcher",
    "subAttributes": [{
      "name": "MinimumDuration",
      "type": "input",
      "preCondition": {
        "name": "AutoOffStore",
        "operation": "isTrue",
        "action": "show"
      }
    }]
  }]
}
```

In the above example, when the **AutoOffStore** is enabled, the sub-attribute **MinimumDuration** is shown on GUI.

### 9.4.1.5 Implementing Cluster Strategy Workflow

#### 9.4.1.5.1 Cluster Strategy Functions

CA developers need to modify the `SampleStrategy.java` to implement the Cluster Strategy workflow. In the skeleton file, the following functions need to be modified.

**Table 13** Functions

Function	Description
<code>public String getName()</code>	Define the name of the cluster strategy. It implements the interface of <code>ClusterStrategy</code> .
<code>public void setNetworkElements(List&lt;String&gt; networkElements)</code>	Initialize the private variable <code>networkElementsStatus</code> , to carry the runtime NE status. It implements the interface of <code>ClusterStrategy</code> .
<code>public void networkElementStatusChanged(String networkElement, NetworkElementStatus networkElementStatus)</code>	Handle the NE status change notification, and update private variable <code>networkElementsStatus</code> . It implements the interface of <code>ClusterStrategy</code> .
<code>public Response execute(Request&lt;?&gt; request, ExecutionManager executionManager)</code>	Main entry to run the cluster strategy workflow. It implements the interface of <code>ClusterStrategy</code> .
<code>public void setInstanceName(String instanceName)</code>	Set the cluster strategy instance name to private variable <code>instanceName</code> . It implements the interface of <code>ClusterStrategyPQAware</code> .
<code>public String getInstanceName()</code>	Return the cluster strategy instance name. It implements the interface of <code>ClusterStrategyPQAware</code> .
<code>public synchronized void setDataRepositories(List&lt;String&gt; dataRepositories)</code>	Set the private variable <code>NEList</code> which carries the NE membership information including Provisioning Status and Role. The content of each <code>dataRepositories</code> item is in <code>name,provisionstat,role</code> format. For example, <code>mvnel, active, primary</code> .
<code>public synchronized void setAttribute1(String strA)</code>	Set the private variable <code>Attribute1</code> .
<code>public synchronized void setAttribute2(boolean boolA)</code>	Set the private variable <code>Attribute2</code> .

**Note:** The naming of function `setXXXXX` must align with the property name configured in `SampleStrategy-configuration.xml`.

#### 9.4.1.5.2 Implementing Cluster Strategy with Processing Queue Function

To add the process queue function, the CA cluster strategy class needs to implement the `ClusterStrategyPQAware` interface, as each cluster needs to be aware of its own instance name in order to connect with the process queue.

For example:

```
public class SampleStrategy implements ClusterStrategy, ClusterStrategyPQAware {...
```

#### *Example 7 ClusterStrategyPQAware Interface*

### 9.4.2 Cluster Strategy APIs

To use the API in CA cluster strategy, below dependency needs be added in `pom.xml`.



```
<dependency>
  <groupId>com.ericsson</groupId>
  <artifactId>LIB-Cluster-Strategy</artifactId>
  <scope>provided</scope>
</dependency>
```

*Example 8 pom.xml*

### 9.4.2.1

## ProcQueueRequestProxy

*Table 14 Method Summary*

Modifier and Type	Method and Description
void	enqueue(QueueOperationContext id, Request request);  Send the request to the processing queue. The QueueOperationContext information (including cluster strategy name, strategy instance name and destination NE name) needs to be provided.
boolean	isRequestInQueue(QueueOperationContext id, Request request);  Check whether the same MOld in this request already exists in the processing queue or not.

### 9.4.2.2

## ClusterStrategyConfService

*Table 15 Method Summary*

Modifier and Type	Method and Description
void	setAttribute(String clusterStrategyType, String clusterStrategyInstanceName, String attributeName, Object value)  Set the attribute for specific strategy instance.
Object	getAttribute(String clusterStrategyType, String clusterStrategyInstanceName, String attributeName)  Get the attribute value from specific cluster strategy instance.

### 9.4.2.3

## ClusterStrategyAlarmService

*Table 16 Method Summary*

Modifier and Type	Method and Description
void	reportEvent(ClusterStrategyAlarmSource alarmSource);  Send a event to ESA.



#### 9.4.2.4 ClusterStrategyEventService

Table 17 Method Summary

Modifier and Type	Method and Description
void	<code>persist(ClusterStrategyEvent clusterStrategyEvent, int ttl)</code> This persist method stores the event into storage at a specified time. For example, If the provisioning state needs to be automatically changed when an event with provisioning failure sent out for a period time, the ClusterStrategyEventService persist method can be used.
boolean	<code>isEventExisted(EventID eventID)</code> Check whether the event associated with the specified event ID exists or not.
void	<code>persist(ClusterStrategyEvent clusterStrategyEvent)</code> Persist the cluster strategy event into the storage.
ClusterStrategyEvent	<code>retrieve(EventID eventID)</code> Retrieve the event associated with the event ID from storage.
void	<code>delete(EventID eventID)</code> Delete the cluster strategy event associated with the event ID.

#### 9.4.2.5 ClusterStrategyInfoService

Table 18 Method Summary

Modifier and Type	Method and Description
String	<code>getClusterStrategyType(String clusterStrategyName)</code> Get the cluster strategy type associated with the strategy name.
Map<String, String>	<code>getNEProvisionStatus(String clusterStrategyName)</code> Get all the provisioning status of NEs in the given cluster strategy.
List<String>	<code>getNEs(String clusterStrategyName)</code> Get all the NEs for the given strategy name.

### 9.4.3 Customizing Off-the-shelf Cluster Strategy

It is highly recommended to use the wizard for generating projects. However, the design based projects are still available.

CA developers can start to initialize the customized strategy project by importing existing design based project. The projects are under CA-SourceCode folder in CA development package. The following strategy are available:

- ActiveActiveSample
- ActiveActiveBestEffortSample
- MultipleAirSample



- `ActiveActiveQueueSample`
- `ActivePassiveSample`
- `PrimaryBackupSample`

After importing the project, it is suggested to rename the project to distinguish with the existing cluster strategy.

Follow the steps below to rename the project:

1. Rename the directory name of the project.
2. Replace the `artifactId` and `name` in the `pom.xml` file with a unique project id and name. In the following example, the values of `artifactId` and `name` are set to `SampleStrategy`:

```
<artifactId>SampleStrategy-Cluster-Strategies</artifactId>
<name>SampleStrategy-Cluster-Strategies</name>
```

3. Follow steps in Step 1 to Step 4 in Section 8.5.1 on page 40 to import the project into Eclipse again.
4. Rename the relevant java package and class name in Eclipse.
5. Change the value of `cluster-strategy-name` tag in the `<Strategy Name>.xml` to a unique cluster strategy name.
6. Rename of the `<Strategy Name>.xml` file with the same file name of `cluster-strategy-name` value that set in Step 5.
7. Change the value of `types` tag in `<Strategy Name>.json` to a new cluster strategy name.
8. Rename `<Strategy Name>.json`.

## 9.5 CA Development for Subscriber View

### 9.5.1 Preparations

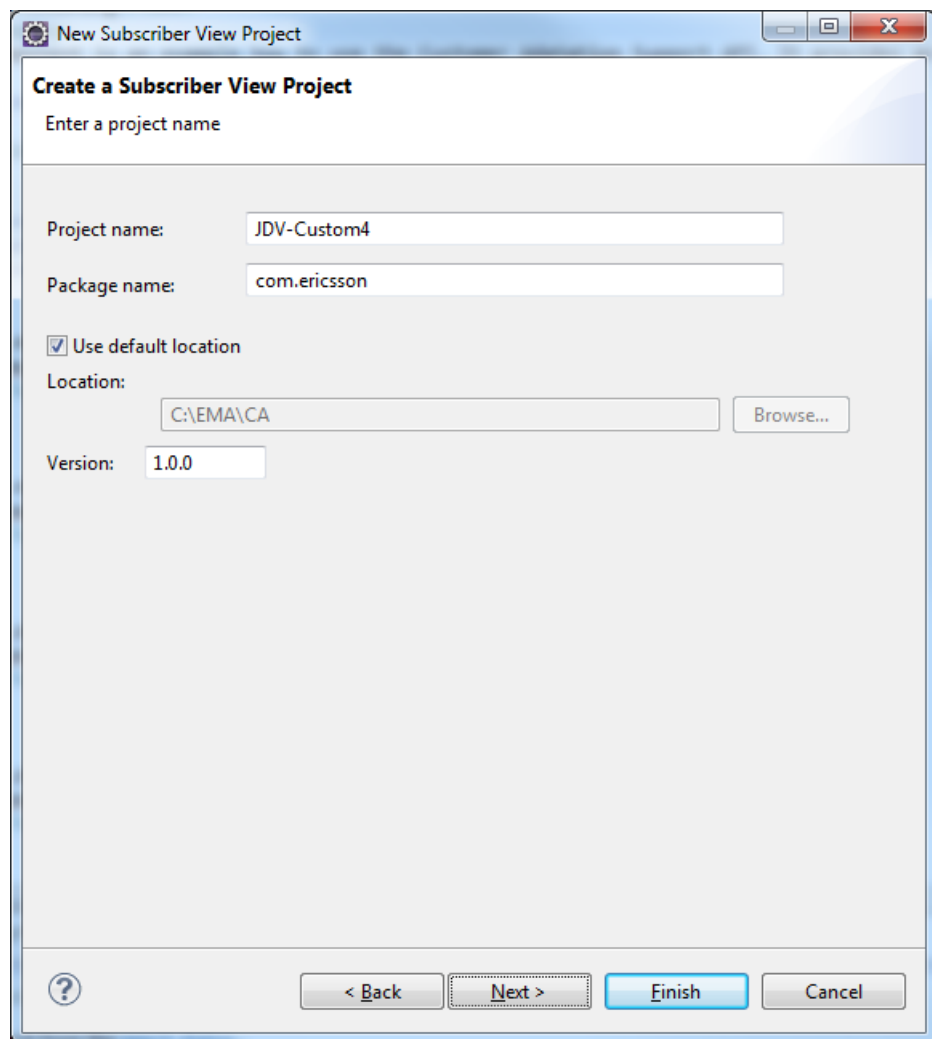
Prepare for the creation of the new CA JDV by analyzing the application to be provisioned and its data model. CA developers need to decide the application type, the application data model details, possible data model differences between the northbound and southbound interfaces, and so on.

### 9.5.2 Project Creation

Use the wizard in Eclipse to create a new Subscriber View JDV.

**Note:** Subscriber Views can also be implemented by using the Designer Studio feature. For details, see *User Guide for Designer Studio*, Reference [6].

1. Start Eclipse and select **File > New > Other** from the menu bar.
2. Select the type of wizard to use: **EMA Customer Adaptation > Subscriber View Wizard** and click **Next**.
3. Specify the information for the subscriber view project to create and click **Next**.



Parameter	Description
Project name	The name of the project to be created for SV.
Package name	The Java package structure.



Parameter	Description
Use default location	If checked, the system default project name and location are used.
Location	The directory where the new project is stored.
Version	Version of build to be used when packaging CA.

4. Specify the information for the subscriber view data model and click **Next**.

Parameter	Description
MO Name	The specific MO name. Must start with an alphabetical character. Must be in capital letters if using CAI northbound interface.



Parameter	Description
Namespace	<p>The namespace to use for the MO.</p> <ul style="list-style-type: none"><li>• For CAI, must begin with <code>http://schemas.ericsson.com/ma/cai/1.0/</code></li><li>• For CAI3G, recommend beginning with <code>http://schemas.ericsson.com/ma/CA/</code></li></ul>
MO Type	<p>The MO Name and Namespace together form MO Type.</p>
MO ID	<p>Select which identity to use, see Section 7.4.2 on page 31.</p> <p>The CA of SV data provisioning only supports one identifier per application. The identifiers are predefined as IMSI, MSISDN, and IMPI.</p>
Operations	<p>Select which operations to create.</p>
Used HLR and EPS as reference template of Sub MOs	<p>If checked, HLR and EPS are used as reference template of Sub MOs. MO IDs will be set to IMSI and MSISDN when checked.</p>

5. Specify configuration parameters for the subscriber view and click **Next**.





**New Subscriber View Project**

**Add configuration parameters**

Configuration parameters:

Name	Type	Default Value
	String	

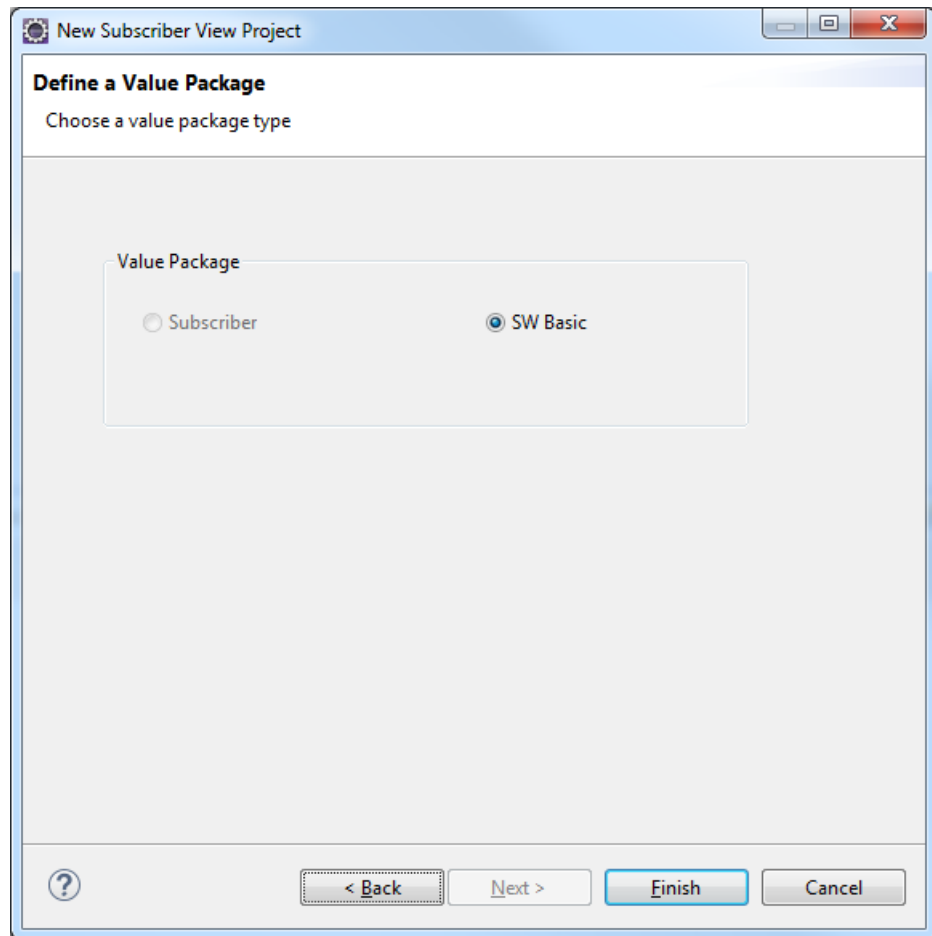
Buttons: Add... Remove

Navigation: ? < Back Next > Finish Cancel

Parameter	Description
Configuration parameters	Add list of parameters that can be configured from the GUI to use in the subscriber view. <sup>(1)</sup>

(1) Do not use Java reserved words, follow Java variable naming rules.

6. Select **sw Basic** as value package, and click **Finish**.



### 9.5.3 Interface Specification

To define a JDV data model for subscriber view, the northbound interface data model must be customized as follows:

- Find the northbound schema and WSDL files in the `src/main/resources/META-INF/webservice_provisioning_cai3g1.2` directory of the CA JDV project.
- For CAI interfaces, `javadataview-descriptor` must:
  - Have triggers with MO in capital letters.
  - Use the namespace `http://schemas.ericsson.com/ma/cai/1.0/`

Example of valid CAI trigger of `javadataview-descriptor.xml`:

```
<trigger value="ns:CreateEIRSUB"/> where
xmlns:ns="http://schemas.ericsson.com/ma/cai/1.0/"
```



- For CAI3G 1.2 interfaces, define the namespace, operations, and other related parameters of northbound requests in the schema and WSDL files according to *Generic CAI3G Interface 1.2*, Reference [3] .

#### 9.5.4 Define the JDV Access Control Model

The access control model includes the configuration of JDV MO, attributes, and operations that fulfill the access control criteria in `managedobjects-descriptor.xml` file. Dynamic Activation applies user authority on both MO and attribute level with these criteria and rules defined in GUI, see Section 11.6 on page 112.

The procedure of defining a JDV access control model is as follows:

1. Find the MO descriptor file in the `src/main/resources/META-INF/` directory of the CA JDV project.
2. Define a JDV group with MO, attributes, operations, and other related parameters to be shown on the access control GUI.

#### 9.5.5 Handling the JDV Business Operations

##### 9.5.5.1 Handling CAI Data Validation

See Section 9.1.5.1 on page 51.

##### 9.5.5.2 Handling CAI3G Data Validation

The incoming request of the JDV is an `org.w3c.dom.Document Request` object. The default class to validate the JDV inbound requests is `com.ericsson.jdv.common.validator.SchemaValidatorUtil`, located in the `LIB-JDV-Public` library.

The CAI3G data validation includes two steps:

1. Initiate a `SchemaValidatorUtil` instance.
2. Call the `SchemaValidatorUtil` instance to execute the validation.

The `SchemaValidatorUtil` instance loads the following two schema files for initiation:

- `cai3g1.2_provisioning.xsd`, common schema file shared by all JDVs
- `<CA-JDV-Project-Name>/src/main/resources/META-INF/webservice_provisioning_cai3g1.2/schemas`, JDV specified schema files

The `SchemaValidatorUtil` instance is initiated in the JDV factory object and called in the JDV object. For details, see the `execute` method of `<moname>JavaDataView.java` and the `newInstance` method of `<moname>JavaDataViewFactory.java`.

To enable multiple includes or imports for an XSD file, initiate the resource resolver as follows:

1. Initiate a `ResourceResolver` instance.
2. Set the multiple includes to true.

## 9.5.6 Transforming Request and Response Data

The default class that transforms the `Document` object to the internal format is `com.ericsson.jdv.common.transformer.XSLTTransformerUtil`, located in the `LIB-JDV-Public` library.

Transforming the XML data includes two steps:

1. Initiate a `XSLTTransformerUtil` instance.
2. Call the `XSLTTransformerUtil` instance to execute the transforming.

The `XSLTTransformerUtil` instance loads a standard XSL file for initiation. An XSL file, such as `JDV-Custom4-Provisioning/src/main/java/com/ericsson/jdv/custom4/handler/request-transformation.xsl`, is located in the `JDV-Custom4-Provisioning` project.

The `XSLTTransformerUtil` instance is initiated in the `JDV` factory object or `JDV` object and called for in the `JDV` object. For details, see the following example source codes in the `JDV-Custom4-Provisioning` project:

- The `newInstance` method of the `JDV-Custom4-Provisioning/src/main/java/com/ericsson/jdv/custom4/Custom1JavaDataViewFactory.java` file
- The `execute` method of the `JDV-Custom4-Provisioning/src/main/java/com/ericsson/jdv/custom4/Custom4JavaDataView.java` file

## 9.5.7 BL Implementation for Subscriber View

For details about how the `JDV` class routes the `MO` operation, see the `execute` method in the example `JDV-Custom4-Resource-Provisioning/src/main/java/com/ericsson/jdv/custom4/Custom4JavaDataView.java` `JDV` file in the `JDV-Custom4-Resource-Provisioning` project. This project also contains detail about how the provisioning `BL` class handles the `MO` operation.

For information how to generate an internal sub `JDV` request, see the class `com.ericsson.jdv.common.utilities.SubscriberViewRequestHelper`, located in the `LIB-JDV-Public` library. An `SV` can dispatch and execute the request by the `com.ericsson.pas.javadataview.ExecutionManager` class, located in the `API-PAS` library.



For information how to combine internal sub JDV response to final SV response returned to BSS, see the `com.ericsson.jdv.common.utilities.SubscriberViewResponseHelper` class, located in the LIB-JDV-Public library.

For detailed API description and examples, see the design base project source code and java doc.

### 9.5.8 SV Mapping and Ignoring Error Messages

When a request is sent from the SV to the internal sub JDV, errors can derive. The SV maps the error from the internal sub JDV according as specified in the `com.ericsson.jdv.common.ErrorMapper` class, located in the LIB-JDV-Public library. In this class, the `Faults_Mapping.properties` property is used. The mapped error is either sent back to BSS. If no match is found in `Faults_Mapping.properties`, the original error message is sent back to BSS.

Error mapping is disabled by default in SV. To enable error mapping, uncomment the `defaultErrorHandler` bean in `javadataview-descriptor.xml` file and define the error mapping in `Faults_Mapping.properties` file.

Some of the error messages derived from the internal sub JDV can be ignored, which means they are not be sent back to BSS by the SV. For example, a delete request is sent from the SV to two sub JDVs, but in one of the JDVs there is nothing to delete. This JDV then sends an error message to the SV, treating this error as a successful deletion and the error is ignored. For details about how to ignore error messages for a delete operation, see `DeleteBusinessLogic.java` provisioning example file in the JDV-Custom4-Provisioning project.

A list of ignorable errors is presented in `DeleteBusinessLogic` class

### 9.5.9 SV Data Inconsistency During Create Operation

An SV generates internal sub JDV requests using the `createRequest` method in the `SubscriberViewRequestHelper` class, and executes the requests one by one, using `ExecutionManager` class.

During Create operation towards the SV JDV, the requests towards sub JDVs can fail. If one or more sub JDV requests on this SV operation were successful before the failing request, the successful requests are rolled back.

The rollback is done by creating rollback request towards the SV Delete operation using `createRollbackRequest` in the `SubscriberViewRequestHelper` class and executing with the `ExecutionManager` class. If rollback is successful, the original SV Create operation returns that the operation failed but rollback was successful to BSS. If rollback also failed, the original SV Create operation returns that the operation failed and rollback failed to BSS.



The SET operation could be implemented in a similar way, but adds more complexity. The recommended way is to not have a rollback, but make sure that the same SET operation can be sent again without failure.

For details about how to handle data inconsistency during the Create operation, see the `CreateBusinessLogic.java` provisioning example files in the JDV-Custom4-Provisioning project.

## **9.5.10 Implementing CAI3G DC**

In order to dispatch a request to the remote CAI3G server through CAI3G Distributed Configuration (DC) in SV, the CAI3G DC routing needs to be configured on GUI. Make sure the `MOType` of the CAI3G DC routing is same with the `MOType` of the request generated by `createRequest` method in the `SubscriberViewRequestHelper` class. For detailed information, see Section 9.5.7 on page 94.

### **9.5.10.1 Handling CAI3G DC External Fault**

The external fault response is wrapped into a `JavaDataViewException`, which has a different format from call internal sub JDV. SV needs to parse the `AdditionalClassifiedError.userData` in `JavaDataViewException` to fetch the detailed fault message for a CAI3G DC external fault.

The following example shows a typical `JavaDataViewException` of the CAI3G DC external fault:



```

JavaDataViewException.ClassifiedError.errorCode = 1101

JavaDataViewException.ClassifiedError.errorDescription = External error.

JavaDataViewException.AdditionalClassifiedError.errorCode = 200000
JavaDataViewException.AdditionalClassifiedError.errorDescription = CAI3G DC External Exception
JavaDataViewException.AdditionalClassifiedError.userData = "
<S:Envelope xmlns:S='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:cai3g='http://schemas.ericsson.com/cai3g1.2/'>
  <S:Header>
    <cai3g:SessionId>22a79b11e9fa4ad7898481d0fd9014c9</cai3g:SessionId>
    <cai3g:SequenceId>1493671468</cai3g:SequenceId>
  </S:Header>
  <S:Body>
    <ns2:Fault xmlns:ns2='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:ns3='http://www.w3.org/2003/05/soap-envelope'>
      <faultcode>ns2:Client</faultcode>
      <faultstring>This is a client fault</faultstring>
      <detail>
        <Cai3gFault:Cai3gFault xmlns='http://schemas.ericsson.com/cai3g1.2/'
xmlns:Cai3gFault='http://schemas.ericsson.com/cai3g1.2/'>
          <faultcode>3013</faultcode>
          <faultreason>
            <reasonText>Invalid parameter.</reasonText>
          </faultreason>
          <faultrole>NEF</faultrole>
        </Cai3gFault:Cai3gFault>
        <UserProvisioningFault:UserProvisioningFault xmlns='http://schemas.ericsson.com/ema/UserProvisioningFault'
xmlns:UserProvisioningFault='http://schemas.ericsson.com/ema/UserProvisioning/'>
          <respCode>1006</respCode>
          <respDescription>Invalid parameter. The XML data is not valid. cvc-enumeration-valid: Value 'ONLINE'
is not facet-valid with respect to enumeration '[ALL, MOBILE, FIXED]'.
It must be a value from the enumeration. - [Processed by PG Node: EBS11-PL-4]
          </respDescription>
        </UserProvisioningFault:UserProvisioningFault>
      </detail>
    </ns2:Fault>
  </S:Body>
</S:Envelope>"

```

In the above example, `errorCode` of `AdditionalClassifiedError` is 200000 which is a special error code for CAI3G DC external fault. The `userData` of `AdditionalClassifiedError` is a string that contains the complete fault message from the remote CAI3G server.

### 9.5.11 Implementation of Customized Error Code

If there is a need for a customized error code, some Java code needs to be included in the Business Logic.

A `JavaDataViewException` needs to be thrown:

```

throw new
JavaDataViewException(JavaDataViewException.EXTERNAL_ERROR,
<Error message>,
  <Customized AdditionalClassifiedError with customized error code>, ex);

```







## 10 Building and Deployment

CA developers can use Maven to build a CA project. Install Maven according to Section 8.2 on page 37 and make sure to set the environment parameters correctly.

CA projects are deployed on payload nodes from a system controller node.

**Note:** The payload node and system controller node referred in this section are:

- PL node and SC node in Native deployment
- node and node-1 in Virtual and Cloud deployment

### 10.1 Building, Deploying, and Undeploying a CA JDV

Follow the steps below to build, deploy, and undeploy a CA JDV project:

1. Build a CA JDV project. See Section 10.1.1 on page 99.
2. Deploy a CA JDV project. See Section 10.1.2 on page 100.

**Note:** If a CA JDV project should be able to send data to a processing log, an additional step is required before deploying a CA JDV. See Section 11.8.1 on page 113 for information on how to add a persistent proclog entry for a CA project.

3. Undeploy a CA JDV project. See Section 10.1.3 on page 101.

**Note:** If a deployed CA JDV project should be able to send data to processing log, a additional step is required after undeploying a CA JDV. See Section 11.8.2 on page 113 for information on how to remove a persistent proclog entry for a CA project.

#### 10.1.1 Building JDV

Follow the steps below to build a CA JDV project:

1. Update the `pom.xml` file (Optional step).
  - For the CA JDV projects the `pom.xml` file is located in the CA Design Based Source code folder. Update the `artifactId`, and `version` in the `pom.xml` file.

**Note:** The JDV `jar` and `tar.gz` filenames inherit the `artifactId` name change. Do not change the parent `pom` version.



2. Run the following commands, in a command prompt. Go to the CA JDV project directory where the `pom.xml` file is located, and compile the CA JDV project with Maven.

**Note:** The `<CA-JDV-Project-Name>` in the following command is the JDV project name.

```
$ cd <CA-JDV-Project-Name>
```

```
$ mvn clean package
```

The packaged CA JDV `jar` file is generated in the `<CA-JDV-Project-Name>\target\` directory, and packed into a deployable `tar.gz` file in the same directory.

### 10.1.2 Deploying JDV

Follow the steps below to deploy a CA JDV project.

#### Prerequisites

The `javadataview-descriptor.xml` file must contain the tag `<valuePackage>` and the belonging value must match one of the licenses enabling the CA JDV feature.

The allowed values are:

- Subscriber
- SW Basic

**Note:** SW Basic is only used for SV.

Check that the license, matching the target value package, is valid:

- For Subscriber: FAT1023338/28 Multi Vendor Subscriber Services
- For SW Basic: FAT1023338/162 EMA Base Package

**Note:** For detailed license information, see **Installing Licenses** in *Software Installation for Native Deployment*, Reference [4].

1. Log on to the target Dynamic Activation server and transfer the CA JDV `tar.gz` files by FTP to the `/home/bootloader/CArepository` directory.
2. Change the owner and the group of the CA JDV `tar.gz` files.

```
# chown actadm:activation /home/bootloader/CArepository/<CA filename>
```

3. Do the following for each payload node where a CA JDV is to be installed:



## 1 Add the CA JDV to the payload node.

From a system controller node:

```
$ bootloader.py submodule add -n <CA filename> -t jdv
-p dve-application --host <hostname>
```

<hostname> is the hostname of the payload node to which the CA JDV is added.

## 2 Activate the payload node.

From a system controller node:

```
$ bootloader.py node activate --host <hostname/all>
```

<hostname> is the hostname of the payload node to which the CA JDV is activated.

**Note:** <all> is only used for new installations. It must not be used in any other cases (such as activation of a new CA JDV).

For example, run the following commands, from a system controller node, to install and activate JDV-Custom1-Provisioning on payload node 3:

Native deployment:

```
$ bootloader.py submodule add -n JDV-Custom1-Provisioni
ng-1.0.0.tar.gz -t jdv -p dve-application --host PL-3
```

```
$ bootloader.py node activate --host PL-3
```

Virtual or Cloud deployment:

```
$ bootloader.py submodule add -n JDV-Custom1-Provisioni
ng-1.0.0.tar.gz -t jdv -p dve-application --host node-3
```

```
$ bootloader.py node activate --host node-3
```

## 4. Check deployment result.

During the JDV deployment, Dynamic Activation parses JDV packages, accumulates all CA JDV triggers, and maps the corresponding CA license.

If the JDV is deployed successfully, the new JDV is displayed in the Dynamic Activation GUI in the **Activation Logic** tab. Check the GUI to see if the CA JDV is loaded successfully. If the new JDV cannot be found in the **Activation Logic** tab, check `consolidated.log` to find the cause of deployment failure. If the JDV fails because of license limitation, an alarm is raised as well. For detailed deployed JDV information, see Section 11.1 on page 111.



### 10.1.3 Undeploying JDV

Follow the steps below to undeploy a CA JDV project:

1. Do the following for each payload node where a CA JDV is to be removed:

- a. Remove the CA JDV from the payload node.

From a system controller node:

```
$ bootloader.py submodule delete -n <CA filename>
-p dve-application --host <hostname>
```

*<hostname>* is the hostname of the payload node from which the CA JDV is deleted.

- b. Activate the payload node.

From a system controller node:

```
$ bootloader.py node activate --host <hostname/all>
```

*<hostname>* is the hostname of the payload node.

**Note:** *<all>* is only used for new installations. It must not be used in any other cases (such as activation of a new CA JDV).

For example, run the following commands, from a system controller node, to undeploy JDV-Custom1-Provisioning on payload node 3:

Native deployment:

```
$ bootloader.py submodule delete -n JDV-Custom1-Provisi
oning-1.0.0.tar.gz -p dve-application --host PL-3
```

```
$ bootloader.py node activate --host PL-3
```

Virtual or Cloud deployment:

```
$ bootloader.py submodule delete -n JDV-Custom1-Provisi
oning-1.0.0.tar.gz -p dve-application --host node-3
```

```
$ bootloader.py node activate --host node-3
```

### 10.1.4 Updating JDV

Follow the step below to update a CA JDV project:

1. Log on to the target Dynamic Activation server and transfer the CA JDV `tar.gz` files, by use of FTP, to the `/home/bootloader/CArepository` directory.
2. Change the owner and the group of the CA JDV `tar.gz` files.



```
# chown actadm:activation /home/bootloader/CArepository/<CA filename>
```

3. Do the following for each payload node where a CA JDV is to be updated:

- 1 Update the CA JDV:

From a system controller node:

```
$ bootloader.py submodule update -n <CA filename> -t jdv -p dve-application --host <hostname>
```

*<hostname>* is the hostname of the payload node to which the CA JDV is updated.

- 2 Activate the payload node.

From a system controller node:

```
$ bootloader.py node activate --host <hostname>
```

*<hostname>* is the hostname of the payload node to which the CA JDV is activated.

For example, run the following commands, from a system controller node, to update and activate JDV-Custom1-Provisioning on payload node 3:

Native deployment:

```
$ bootloader.py submodule update -n JDV-Custom1-Provisioning-1.0.0.tar.gz -t jdv -p dve-application --host PL-3
```

```
$ bootloader.py node activate --host PL-3
```

Virtual or Cloud deployment:

```
$ bootloader.py submodule update -n JDV-Custom1-Provisioning-1.0.0.tar.gz -t jdv -p dve-application --host node-3
```

```
$ bootloader.py node activate --host node-3
```

## 10.2 Building, Deploying, and Undeploying a CA JCA

Follow the steps below to build, deploy, and undeploy a CA JCA project:

1. Build a CA JCA project. See Section 10.2.1 on page 104.
2. Deploy a CA JCA project. See Section 10.2.2 on page 104.



**Note:** If a CA JCA project should be able to send data to processing log, a additional step is required before deploying a CA JCA. See Section 11.8.1 on page 113 for information on how to add a persistent proclog entry for a CA project.

3. Undeploy a CA JCA project. See Section 10.2.3 on page 105.

**Note:** If a deployed CA JDV project should be able to send data to processing log, a additional step is required after undeploying a CA JDV. See Section 11.8.2 on page 113 for information on how to remove a persistent proclog entry for a CA project.

## 10.2.1 Building JCA

Follow the steps below to build a CA JCA project:

1. Update the `pom.xml` file (Optional step).
  - For the CA JCA projects the `pom.xml` file is located in the CA Design Based Source code folder. Update the `artifactId`, and `version` in the `pom.xml` file.

**Note:** The JCA `jar` and `tar.gz` filenames inherit the `artifactId` name change. Do not change the parent `pom` version.

2. Run the following commands, in a command prompt. Go to the CA JCA project directory where the `pom.xml` file is located, and compile the CA JCA project with Maven.

**Note:** The `<CA-JCA-Project-Name>` in the following command is the JCA project name.

```
$ cd <CA-JCA-Project-Name>
```

```
$ mvn clean package
```

The packaged CA JCA `jar` file is generated in the `<CA-JCA-Project-Name>\target\` directory, and packed into a deployable `tar.gz` file in the same directory.

## 10.2.2 Deploying JCA

Follow the steps below to deploy a CA JCA project:

1. Log on to the target Dynamic Activation server and transfer the CA JCA `tar.gz` files by FTP to the `/home/bootloader/CArepository` directory.
2. Change the owner and the group of the CA JCA `tar.gz` files.

```
# chown actadm:activation /home/bootloader/CArepository/<CA filename>
```



### 3. Do the following for each payload node where a CA JCA is to be installed:

#### 1 Add the CA JCA to the payload node.

From a system controller node:

```
$ bootloader.py submodule add -n <CA filename> -t
connector -p dve-application --host <hostname>
```

*<hostname>* is the hostname of the payload node to which the CA JCA is added.

#### 2 Activate the payload node.

From a system controller node:

```
$ bootloader.py node activate --host <hostname/all>
```

*<hostname>* is the hostname of the payload node to which the CA JCA is activated.

**Note:** *<all>* is only used for new installations. It must not be used in any other cases (such as activation of a new CA JCA).

For example, run the following commands, from a system controller node, to install and activate JCA-Custom3 on payload node 3:

Native deployment:

```
$ bootloader.py submodule add -n JCA-Custom3-1.0.0.tar.
gz -t connector -p dve-application --host PL-3
```

```
$ bootloader.py node activate --host PL-3
```

Virtual or Cloud deployment:

```
$ bootloader.py submodule add -n JCA-Custom3-1.0.0.tar.
gz -t connector -p dve-application --host node-3
```

```
$ bootloader.py node activate --host node-3
```

#### 4. Check deployment result.

If the JCA is deployed successfully, the display name of the JCA is available in the protocol candidate list of the **Add Network Element** wizard in the **Network Element** tab. For more information on how to use the GUI, see Section 11 on page 111. If the display name of the JCA cannot be found in the protocol candidate list, check `server.log` to find the cause of deployment failure.

## 10.2.3 Undeploying JCA

Follow the steps below to undeploy a CA JCA project:



1. Do the following for each payload node where a CA JCA is to be removed:

- 1 Remove the CA JCA from the payload node.

From a system controller node:

```
$ bootloader.py submodule delete -n <CA filename> -p
dve-application --host <hostname>
```

*<hostname>* is the hostname of the payload node from which the CA JCA is deleted.

- 2 Activate the payload node.

From a system controller node:

```
$ bootloader.py node activate --host <hostname/all>
```

*<hostname>* is the hostname of the payload node.

**Note:** *<all>* is only used for new installations. It must not be used in any other cases (such as activation of a new CA JCA).

For example, run the following commands, from a system controller node, to undeploy JCA-Custom3 on payload node 3:

Native deployment:

```
$ bootloader.py submodule delete -n JCA-Custom3-1.0.0.t
ar.gz -p dve-application --host PL-3
```

```
$ bootloader.py node activate --host PL-3
```

Virtual or Cloud deployment:

```
$ bootloader.py submodule delete -n JCA-Custom3-1.0.0.t
ar.gz -p dve-application --host node-3
```

```
$ bootloader.py node activate --host node-3
```

## 10.2.4 Updating JCA

Follow the step below to update a CA JCA project:

1. Log on to the target Dynamic Activation server and transfer the CA JCA `tar.gz` files, by use of FTP, to the `/home/bootloader/CArepository` directory.
2. Change the owner and the group of the CA JCA `tar.gz` files.

```
# chown actadm:activation /home/bootloader/CArepository/<CA filename>
```





### 3. Do the following for each payload node where a CA JCA is to be updated:

#### 1 Update the CA CJA:

From a system controller node:

```
$ bootloader.py submodule update -n <CA filename> -t
connector -p dve-application --host <hostname>
```

*<hostname>* is the hostname of the payload node to which the CA JCA is updated.

#### 2 Activate the payload node.

From a system controller node:

```
$ bootloader.py node activate --host <hostname>
```

*<hostname>* is the hostname of the payload node to which the CA JCA is activated.

For example, run the following commands, from a system controller node, to update and activate JCA-Custom3 on payload node 3:

Native deployment:

```
$ bootloader.py submodule update -n JCA-Custom3-1.0.0.ta
r.gz -t connector -p dve-application --host PL-3
```

```
$ bootloader.py node activate --host PL-3
```

Virtual or Cloud deployment:

```
$ bootloader.py submodule update -n JCA-Custom3-1.0.0.ta
r.gz -t connector -p dve-application --host node-3
```

```
$ bootloader.py node activate --host node-3
```

## 10.3 Building, Deploying, and Undeploying a CA Cluster Strategy

This section contains information about how to build, deploy, and undeploy a Cluster Strategy.

### 10.3.1 Packaging and Deployment

CA developers can use Maven to build a CA project. Install Maven according to Section 8.2 on page 37 and make sure to set the environment parameters correctly.



### 10.3.1.1 Building Cluster Strategy

Use `mvn clean package` to build the source code. The generated code ends up in `<Stratege Name>-Cluster-Strategy/target/<Strategy Name>-Cluster-Strategies-1.0.0.tar.gz` file.

### 10.3.1.2 Deploying Cluster Strategy

Follow the steps below to deploy a cluster strategy:

1. Log on to the target Dynamic Activation server and transfer the cluster strategy `tar.gz` files by FTP to the `/home/bootloader/CArepository` directory.

2. Change the owner and the group of the cluster strategy `tar.gz` files.

```
# chown actadm:activation /home/bootloader/CArepository/<CA filename>
```

3. Do the following for each PL node where a cluster strategy is to be installed:

- a Add the cluster strategy to the PL node.

From an SC node:

```
$ sudo -u actadm bootloader.py submodule add -n
<CA filename> -t clusterstrategy -p dve-application
--host <hostname>
```

`<hostname>` is the hostname of the PL node to which the cluster strategy is added.

- b Activate the PL node.

From an SC node:

```
$ sudo -u actadm bootloader.py node activate --host
<hostname/all>
```

`<hostname>` is the hostname of the PL node to which the cluster strategy is activated.

**Note:** `<all>` is only used for new installations. It must not be used in any other cases (such as activation of a new cluster strategy).

For example, run the following commands, from an SC node, to install and activate `ActiveActiveSample-Cluster-Strategy` on `PL-3`:

```
$ sudo -u actadm bootloader.py submodule add -n
ActiveActiveSample-Cluster-Strategy.tar.gz -t
clusterstrategy -p dve-application --host PL-3
```



```
$ sudo -u actadm bootloader.py node activate --host PL-3
```

### 10.3.1.3 Undeploying Cluster Strategy

Follow the steps below to undeploy an Cluster Strategy:

1. Do the following for each PL node where a Cluster Strategy is to be removed:

- a Remove the Cluster Strategy from the PL node.

From an SC node:

```
$ sudo -u actadm bootloader.py submodule delete -n
<CA filename> -p dve-application --host <hostname>
```

*<hostname>* is the hostname of the PL node from which the Cluster Strategy is deleted.

- b Activate the PL node.

From an SC node:

```
$ sudo -u actadm bootloader.py node activate --host
<hostname/all>
```

*<hostname>* is the hostname of the PL node.

**Note:** *<all>* is only used for new installations. It must not be used in any other cases (such as activation of a new Cluster Strategy).

For example, run the following commands, from an SC node, to undeploy ActiveActiveSample-Cluster-Strategy on PL-3:

```
$ sudo -u actadm bootloader.py submodule delete -n
ActiveActiveSample-Cluster-Strategy-1.0.0.tar.gz -p
dve-application --host PL-3
```

```
$ sudo -u actadm bootloader.py node activate --host PL-3
```

## 10.4 Add 3pp Jar Files to the System

Follow the steps below to add 3pp jar files as library extensions.

**Note:** The jar files in `lib/lib-ext` are loaded in a single class loader, hence only one version of a library should be present within the `lib/lib-ext` folders.

1. Create a `tar.gz` file, which wraps the jars:



```
$ sudo tar -cvf <nameOfTarFile>-<version>.tar.gz  
<First3ppFileName>.jar <Next3ppFileName>.jar
```

**Note:** The tar.gz file needs to follow the naming conversion;  
    <name>-<version>.tar.gz

All needed 3pp jar files can be packed into the same tar.gz file.

2. Log on to the target Dynamic Activation server, and transfer the <nameOfTarFile>-<version>.tar.gz file by use of FTP, to the /home/bootloader/repository directory.
3. Change the owner and the group of the <nameOfTarFile>-<version>.tar.gz file.

```
# chown actadm:activation /home/bootloader/repository/<n  
ameOfTarFile>-<version>.tar.gz
```

4. Perform the following for each payload node, where the 3pp jar files are to be added:

- 1 Add the 3pp jar files to the payload node.

From a system controller node:

```
$ bootloader.py submodule add -n <nameOfTarFile>-<ve  
rsion>.tar.gz -t lib-ext -p dve-application --host  
<hostname>
```

<hostname> is the hostname of the payload node to which the 3pp jar files are added.

- 2 Activate the payload node:

```
$ bootloader.py node activate --host <hostname>
```

<hostname> is the hostname of the payload node to be activated.



# 11 Configuring Dynamic Activation for CA Provisioning

After deploying all CA JDVs, JCAs and Cluster Strategy successfully, configure Dynamic Activation before the CA provisioning. The following configuration subsections guide developers to initialize the NE configuration, apply routing method, and grant the authority of a provisioning user regarding JDV triggers and MO attributes.

For details about the Dynamic Activation configuration for a CA, see *User Guide for Resource Activation*, Reference [5].

## 11.1 Checking JDV Information

A valid CA JDV is listed in the **Activation Logic** tab in the Dynamic Activation GUI. Find a CA JDV in the JDV list, click the **View Details** button at the beginning of the JDV row. The selected JDV properties are displayed. Double check the property information. If the `ConfigurationProperty` annotation is used on custom property setters or getters in the JDV factory, these properties are visible when clicking **View Details**. To change them, click **Change**.

## 11.2 Configuring NE

For CUDB layered data and monolithic data provisioning CA, create an NE configuration in the **Network Elements** tab.

## 11.3 Configuring Cluster Strategy

NEs can be attached to a cluster. CA developers can configure different cluster strategies (off-the-shelf cluster strategies or new CA cluster strategy)).

### 11.3.1 Configuration and Testing

#### 11.3.1.1 Check Deployment Result.

If the CA cluster strategy deployed successfully, the new strategy is displayed in the Dynamic Activation **GUI > Launchpad > Cluster Strategy > Create Cluster Strategy Dialog**.



### 11.3.1.2 Test CA Cluster Strategy

To test the cluster strategy logic, the NE and routing information need to be configured. For detailed information, see *User Guide for Resource Activation*, Reference [5].

## 11.4 Configuring Routing

In Dynamic Activation, **Routing** is used to find the target NE by following different routing methods with request resource key data. Follow the steps below to create routing:

1. Select the **Network Element Type**. All available NE types are registered by JDV.
2. Select available NEs or NE Groups as routing target candidates.
3. Select a method for the routing. Dynamic Activation supports four routing methods:
  - **Regular Expression**
  - **Number Range**
  - **Number Series**
  - **Unconditional**

Each routing method requires different parameters. Provide those mandatory parameters to complete the routing creation.

## 11.5 Configuring Loose Error Handling (Optional)

For monolithic data provisioning CA, configure the loose error handling in the **Loose Error Handling** tab.

## 11.6 Granting Authority to Provisioning User

Before sending a provisioning request to Dynamic Activation, grant admin user with the authority for provisioning user accounts. Admin user can create a user only for provisioning.

Put in the name, password, and other mandatory information in the first step of creating a user. Only select the **Read-only** authorities in the **Configuration Management** tab. In the **Provisioning Authorities** tab, select the required domain and NE for the provisioning user.



## 11.7 Configuring Logging

The file `/usr/local/pgngn/dve-application-<version>/config/log4j.xml` controls what log messages that are sent to the `consolidated.log`. To be able to see log messages originating from CA code, a category with the correct Java package name and the wanted priority need to exist in this file. For more information about log4j, see Reference [8].

## 11.8 Manage Custom Processing Log Settings

It is possible to set up persistent settings for Customer Adaptations that need to be able to send data to Dynamic Activation Application processing logs.

### 11.8.1 Deploy New Processing Log Settings

This section describes the procedure of deploying custom processing log settings for Dynamic Activation Application CA sub-modules.

1. To deploy a new proclog setting for a module, execute the following commands:

```
$ bootloader.py logging deploy-proclog -n <module name>
-c <category>
```

**Note:** `<category>` refers to the log category name that the CA sub-module is using.

### 11.8.2 Undeploy Processing Log Settings

This section describes the procedure of undeploying custom processing log settings for Dynamic Activation Application CA sub-modules.

---



---

### Caution!

Undeploying processing log entries on running/active modules may cause loss of processing log data, and/or traffic loss.

---



---

1. To undeploy a previously deployed custom proclog setting, run the following command:

```
$ bootloader.py logging undeploy-proclog --name/-n
<module name> --category/-c <proclog category>
```

**Note:** `<category>` refers to the log category name that the CA sub-module is using.



### 11.8.3 List Custom Processing Log Settings

This section describes the procedure of listing active processing log settings on a Dynamic Activation Application system.

1. To list the active processing log settings, execute the following command:

```
$ bootloader.py logging list -t proclog
```





## 12 Testing and Provisioning over CAI3G

This section provides some common tools that can be used for testing a customer adaptation.

### 12.1 Testing Tools for Sending Requests – SoapUI

SoapUI is a cross-platform functional testing solution. With an easy-to-use graphical interface and enterprise-class features, SoapUI allows to easily and rapidly create and execute automated functional, regression, compliance, and load tests. CA developers can use this tool to send CAI3G requests to Dynamic Activation and check the response from Dynamic Activation.

Download and install SoapUI from <http://www.soapui.org>.

### 12.2 LDAP Simulator – OpenLDAP

OpenLDAP is an open source implementation of the Lightweight Directory Access Protocol (LDAP). This tool can be used to simulate the CUDB.

Download and install OpenLDAP from <http://www.openldap.org/>.

### 12.3 Application Data Browser – Apache Directory Studio

Apache™ Directory Studio is a complete directory tooling platform intended to be used with any LDAP server. CA developers can use this tool to browse CUDB LDAP and verify provisioning result.

Download and install Apache Directory Studio from <http://directory.apache.org/studio/>.

### 12.4 HTTP Test Server

HTTP Test Server is a tool that can be used as an HTTP server with JCA-Custom3 project. The HTTP Test Server can be found in `ca-plugins-<version>.tar.gz`, located in the `CXP9029435-<version>.tar.gz` package. For information on how to use this tool, refer to the `readme` file in the `tar.gz` package.





## 13 Appendix A – CUDB Data Model in CUDB Configuration Files

The CUDB configuration files contain the data model in XML format of different applications in CUDB. The following sections contain description of different elements.

### 13.1 XML Tags in CUDB Configuration Files

This section contains information about XML tags in CUDB configuration files.

#### 13.1.1 LDAP Attribute Entry

This attribute represents an LDAP entry and there must be at least one entry in a CUDB configuration file.

*Table 19 Tags of LDAP Attribute Entry*

Name	Description
name	The LDAP domain name
jndi	The name of the CUDB configuration file. This element is unique, and it is also defined in the JDV.

#### 13.1.2 LDAP Attribute staticattr

Static attributes are added when the entry is created and the attributes are ignored in modifying requests. This attribute is used to define object classes.

*Table 20 Tags of LDAP Attribute staticattr*

Name	Description
ldap	The ldap attribute name in CUDB
value	The value of the ldap attribute. Multiple values are supported.

#### 13.1.3 Element attr

*Table 21 Tags of LDAP Attribute attr*

Name	Description
pod	The name of the attr attribute in an inbound request.
ldap	The ldap attribute name in CUDB



Name	Description
ldap-type	<p>The ldap attribute type in CUDB</p> <p>Supported types:</p> <ul style="list-style-type: none"><li>• Integer</li><li>• NumericString</li><li>• OctetString</li><li>• DirectoryString</li><li>• BitString</li><li>• IA5String</li></ul>
mandatory	<p>The mandatory attribute is set to true if the attr attribute is mandatory.</p> <p>The LDAP attributes are optional by default if the mandatory attribute is not present.</p>
ndc	<p>Number of Collisions to Detect. If the ndc attribute is set to 0, the cdc value is not increased.</p>
cdc-max	<p>The maximum value of the cdc attribute. When cdc reaches the cdc-max value, the value of cdc is reset to 0.</p>



## 14 Appendix B – Version History

The following table specifies the version history of the CA API artifact. The current version can be found in the Maven repository in `ca-repository-<version>.tar.gz`, located in the `CXP9029435-<version>.tar.gz` package. The version is specified in `pom.xml` of the example projects.

*Table 22 Versions of CA API Artifact*

Version	Description
1.0.0	<p>Baseline revision. Included from Dynamic Activation version 7.0 CP3.</p> <p>From Dynamic Activation version 7.1 CP1, the following applies:</p> <ul style="list-style-type: none"><li>• Support for Subscriber view JDVs is included</li><li>• Older CAs have to be updated to handle value package licenses. See Section 10.1.2 on page 100 for different value package licenses.</li></ul>





## Reference List

### Ericsson Documents

- [1] *Library Overview*, 18/1553-CSH 109 628 Uen
- [2] *Function Specification Resource Activation*, 3/155 17-CSH 109 628 Uen
- [3] *Generic CAI3G Interface 1.2*, 2/155 19-FAY 302 0003 Uen
- [4] *Software Installation for Native Deployment*, 1/1531-CSH 109 628 Uen
- [5] *User Guide for Resource Activation*, 1/1553-CSH 109 628 Uen
- [6] *User Guide for Designer Studio*, 10/1553-CSH 109 628 Uen
- [7] *Northbound Interface Adapter Customization Development Guide for HTTP-Based Protocol*, 7/1553-CSH 109 628 Uen

### External Links

- [8] *Apache Log4j 2*, <http://logging.apache.org/log4j/2.x/>