

DUP Customer Adaptation Migration Guide

Ericsson Dynamic Activation 1

USER GUIDE

Copyright

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

Disclaimer

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

Trademark List

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



Contents

1	Introduction	1
1.1	Purpose and Scope	1
1.2	Target Groups	2
1.3	Typographic Conventions	2
1.4	Prerequisites	2
2	DUP Migration Overview	3
2.1	DUP Migration Workflow	3
3	Preparing for CA Development Environment	5
4	CA DUP Migration	7
4.1	Converting DUP to JDV	7
4.1.1	MVNE DUP Conversion	7
4.1.2	SV DUP Conversion	9
4.2	Fixing Grammar Issues and Other Issues	10
4.2.1	Difference between DUP and Java	10
4.2.2	Runtime Error	14
4.2.3	Expand JDV to Support More Functions	15
4.2.4	API and Primitive Type Support	19
4.3	Error Mapping	19
4.4	Configuration Data Support	20
4.4.1	Configuration Data Support in Service JDV	20
4.4.2	Configuration Data Support in Resource JDV	21
4.5	CAI3G Interface Definition	22
4.6	Access Control Support	22
4.7	Special Routing Attribute (Optional)	22
4.8	Loose Error Handling Support (Optional)	23
4.9	CAI3G DC Support	23
4.9.1	Handling CAI3G DC External Fault	24
4.10	Building and Deploying a CA JDV	25
4.10.1	Building a CA JDV	25
4.10.2	Deploying a CA JDV	25
4.10.3	Undeploying a CA JDV	26
5	CA Development for Connector	27
5.1	Preparing for a Connector Project	27
5.1.1	Creating a Connector Project	27
5.1.2	Importing Files into a Connector Project	27



5.1.3	Creating a ConnectorFactory Class	28
5.2	Building and Deploying a CA Connector	28
5.2.1	Building a CA Connector	28
5.2.2	Deploying a CA Connector	29
5.2.3	Undeploying a CA Connector	30
6	Configuring Dynamic Activation for CA Provisioning	31
7	Testing and Provisioning over CAI3G	33
8	Existing JDVs and Connectors Supporting Multiple Versions	35
8.1	CA DUP Migration	35
8.1.1	Preparing for a JDV Project	35
8.1.2	Building and Deploying a CA JDV	36
8.2	CA Development for Connector	36
8.2.1	Preparing for a Connector Project	36
8.2.2	Building and Deploying a CA Connector	36
	Reference List	37



1 Introduction

This document describes the processes of developing Customer Adaptation (CA) in Ericsson™ Dynamic Activation (EDA), which is converted from Data Unit Processing (DUP) and Java Link in Classic Multi Activation. This document is intended for customers to convert DUP and Java Link to provisioning Business Logic (BL) with Java Data Views (JDVs) or outbound Connectors.

The following figure shows the overall structure of DUP migration:

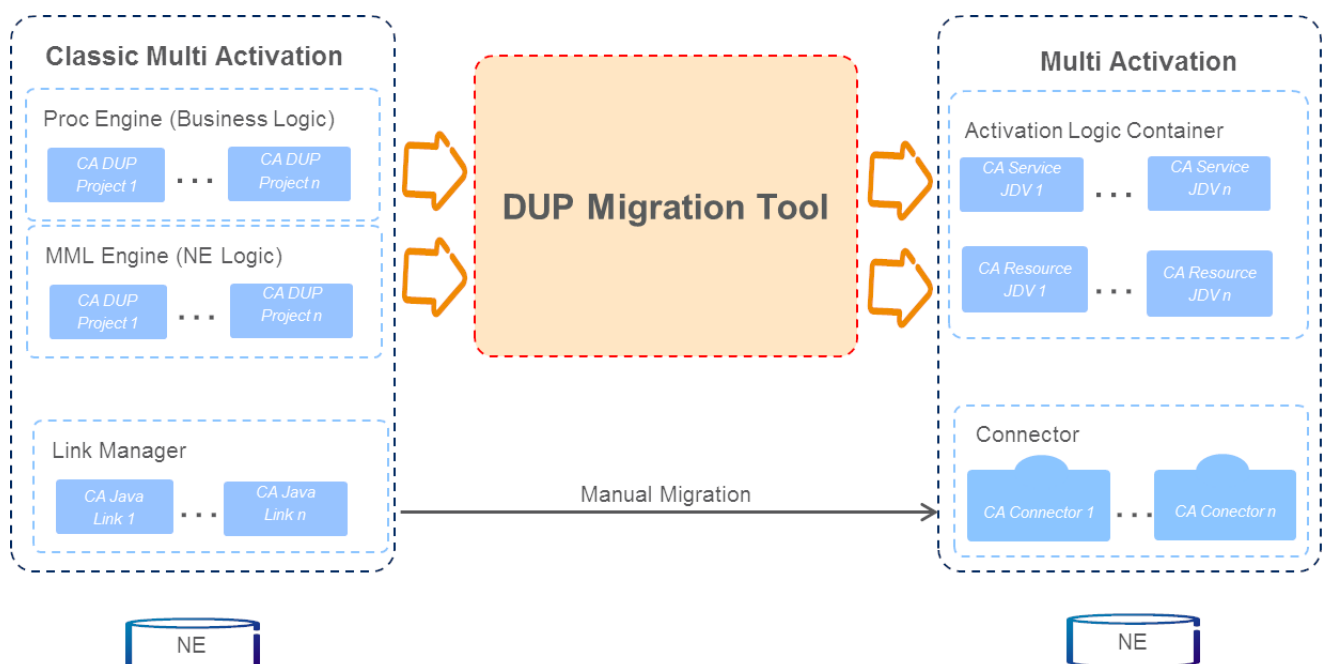


Figure 1 DUP Migration Overall Structure

1.1 Purpose and Scope

This document is intended to:

- Convert a Multi Vendor Network Element (MVNE) DUP CA to a JDV CA.
- Convert a DUP customer adaptation Subscriber View (SV) project to an SV JDV.
- Adapt Java Link to outbound Connectors.



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 JDV and Connector development
- Knowledge of DUP BL and Java Link in Classic Multi Activation
- Knowledge of Linux™
- Knowledge of the Java language, Eclipse, and Maven



2 DUP Migration Overview

The purpose of the DUP migration is to convert DUP code into JAVA code with some manual configuration.

2.1 DUP Migration Workflow

A typical DUP migration workflow is shown in Figure 2.

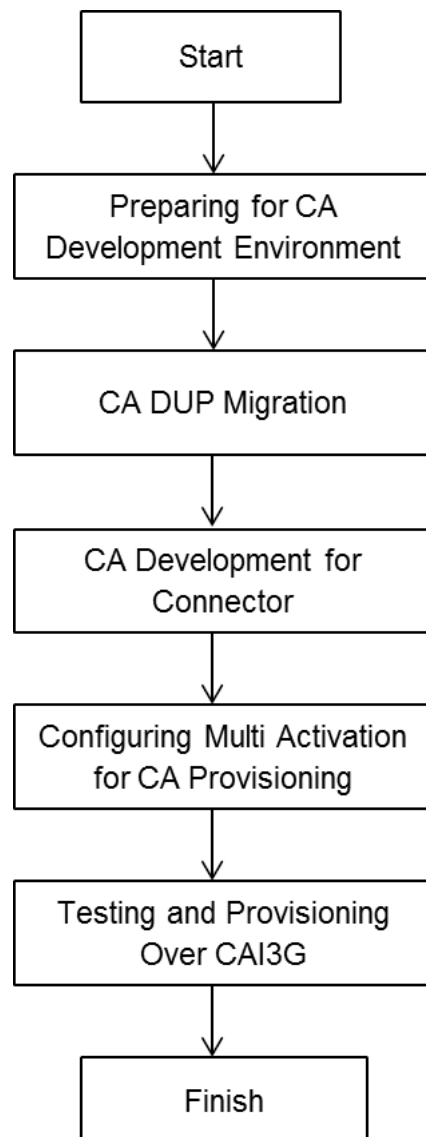


Figure 2 DUP Migration Workflow Chart





3 Preparing for CA Development Environment

For detailed instructions on preparing CA development environment, refer to section *Preparing for CA Development Environment* in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

To prepare the CA development environment:

1. Create a directory `C:\CA_Development`.
2. Transfer `ca-dup-convert-tool-<version>.tar.gz` to `C:\CA_Development` and extract the migration package. All the extracted files are stored in `C:\CA_Development\dup-migration-1.0.0-tool`.





4 CA DUP Migration

This section describes how to convert DUP code to JDV with a Migration Tool. The Migration Tool can detect the type of DUP project and generate a corresponding well-structured JDV project. Some issues need manual adaptation after conversion. It only supports source code rather than compiled package. The Migration Tool has the following limitations:

- The Migration Tool cannot find all DUP grammar issues in the original DUP project. These issues must be fixed either before executing the migration or manually modifying the JDV codes after migration execution, see Section 4.2 on page 10.
- The Migration tool only supports CAI3G interface. CAI and CAI3G projects are both migrated into CAI3G projects. CAI3G schema validation is not supported yet.
- The Migration Tool ignores the following DUP codes:
 - CAI mapping
 - Initial
 - Destroy
 - Routing handler
- DUP SV migration supports the following SV logics:
 - SV logics call MVNE (CA DUP codes) logics.
 - SV logics call standard DUP codes which have the same interface implementation in JDV logics. Such as, HLR/AUC/MNP, EPS, MTAS, and IPWorks ENUM.

4.1 Converting DUP to JDV

4.1.1 MVNE DUP Conversion

To convert an MVNE DUP project to an MVNE JDV project:

Note: All following commands are executed in **Windows**.

1. Copy the MVNE DUP project `<CA-Dup-Project-Name>` to local disk `C:\CA_Development`.
2. Go to the directory that stores DUP Migration Tool:



```
> cd C:\CA_Development\dup-migration-1.0.0-tool\bin
```

3. Run the following command to convert Processing Logic DUP code to Service JDV and convert NE Logic DUP code to Resource JDV.

```
> dup_migration_tool.bat input output [-ns base_namespace]
```

For example:

Run the following command to convert Processing Logic DUP code to Service JDV:

```
> dup_migration_tool.bat C:\CA_Development\CA-Dup-Project-Name\CA-DUP-ProcEngine C:\CA_Development\CA-Java-Project-Name\CA-Service-JDV
```

Run the following command to convert NE Logic DUP code to Resource JDV:

```
> dup_migration_tool.bat C:\CA_Development\CA-Dup-Project-Name\CA-DUP-MMLEngine C:\CA_Development\CA-Java-Project-Name\CA-Resource-JDV
```

For detailed usage of the preceding command, run the following command:

```
> dup_migration_tool.bat --help
```

4. Import the Service JDV project into Eclipse:
 - a Open Eclipse, and click **File > Import**.
 - b In the **Import** dialog, select **Maven > Existing Maven Projects** and click **Next**.
 - c Click **Browse** and select the directory of Service JDV project. Click **Select all**, and click **Next**.
 - d Click **Finish**.
5. Import the Resource JDV project into Eclipse:
 - a Open Eclipse, and click **File > Import**.
 - b In the **Import** dialog, select **Maven > Existing Maven Projects** and click **Next**.
 - c Click **Browse** and select the directory of Resource JDV project. Click **Select all**, and click **Next**.
 - d Click **Finish**.



4.1.2 SV DUP Conversion

To convert an SV DUP project to an SV JDV project:

Note: All following commands are executed in **Windows**.

1. Copy the SV DUP project *<CA-Dup-Project-Name>* to local disk *C:\CA_Development*.
2. Go to the directory that stores DUP Migration Tool:


```
> cd C:\CA_Development\dup-migration-1.0.0-tool\bin
```
3. Convert an SV DUP project to an SV JDV project.
 - For SV DUP depends on both underlay DUP MO definition and logics, underlay DUP BL component (MVNE JDV) must be converted first, see Section 4.1.1 on page 7 for detailed steps of MVNE DUP Conversion. Then run the following command to convert the SV DUP project to the SV JDV project.

```
> dup_migration_tool.bat input output [-ns
"base_namespace"] [-depend "dependent JDV"]
```

For example:

```
> dup_migration_tool.bat C:\CA_Development\SV-
Dup-Project-Name\Dup-SCEpsApnSV C:\CA_Develop
ment\SV-Java-Project-Name\SCEpsApn-JDV -depend
C:\CA_Development\HSSSESMPL-JDV
```

- For SV DUP depends on the standard Dynamic Activation DUP logics which has the same interface implementation in JDV logics (such as HLR/AUC, EPS, MTAS, and IPWorks), the SV depended standard Dynamic Activation MO ASN.1 must be copied to the ASN.1 folder of the SV before the migration. Then run the following command to convert the SV DUP project if there is no MVNE DUP dependency in SV.

```
> dup_migration_tool.bat input output [-ns
"base_namespace"]
```

For example:

```
> dup_migration_tool.bat C:\CA_Development\SV-Dup-Pr
oject-Name\Dup-SCEpsApnSV C:\CA_Development\SV-Java-
Project-Name\SCEpsApn-JDV
```

For detailed usage of the preceding command, run the following command:

```
> dup_migration_tool.bat --help
```

4. Import the SV JDV project into Eclipse.



- a Open Eclipse, and click **File > Import**.
- b In the **Import** dialog, select **Maven > Existing Maven Projects** and click **Next**.
- c Click **Browse**, and select the directory of SV JDV project. Click **Select all**, and click **Next**.
- d Click **Finish**.
- e Fix grammar issues and other issues in SV JDV, see Section 4.2 on page 10.

4.2 Fixing Grammar Issues and Other Issues

After conversion, some grammar issues need to be resolved. The following sections provide some known grammar issues.

4.2.1 Difference between DUP and Java

4.2.1.1 Mandatory Type Conversion

DUP supports automatic type conversion while Java does not. So when DUP project is converted to Java project, manually convert the type.

For example:

The DUP code example is as follows:

```
declare newMo HlrBL.Subscription;  
declare newHlrMo HlrBL.Subscription;  
newHlrMo ::= Engine::process("Get", newMo);
```

The converted Java code example is as follows:

```
Subscription newMo = new Subscription();  
Subscription newHlrMo = new Subscription();  
newHlrMo = DU.cloneDU(ServiceLayer.process("Get", newMo));
```

Manually replace the line `newHlrMo = DU.cloneDU(ServiceLayer.process("Get", newMo));` with the following line:

```
newHlrMo = (Subscription)DU.cloneDU(ServiceLayer.proces  
s("Get", newMo));
```



4.2.1.2 Method Conversion

In API, some DUP methods are converted to multi Java methods. In some cases, this conversion can cause Java compiling errors.

For details on the API function change, refer to *DUP Migration API Reference Manual*, Reference [4].

For example, the following DUP method `String::readUntilToken` is converted to two Java methods:

- `DUString.readUntilToken`
- `DUString.readUntilTokenSecondPart`

The DUP code example is as follows:

```
declare tplIdAddSubSIM  INTEGER;
declare configurationValue  IA5String;
tplIdAddSubSIM  ::= String::str2int (String::readUntilToken
    (configurationValue, "#"));
```

The converted Java code example is as follows:

```
tplIdAddSubSIM =DUString.str2int(DUString.readUntilToken(c
onfigurationValue, "#");configurationValue = DUString.readU
ntilTokenSecondPart(configurationValue, "#"));
```

Manually replace the preceding line with the following lines:

```
tplIdAddSubSIM =DUString.str2int(DUString.readUntilToken
(configurationValue, "#));

configurationValue = DUString.readUntilTokenSecondPart(con
figurationValue, "#);
```

4.2.1.3 Class Import

DUP supports calling the method in other `ipc` file without head declaration. But in Java, this is an error to be solved.

For example:

In DUP, the code examples are as follows:

In file `HlrBL--handleConfigData.ipc`

```
VOID setCallForwardParam()
{
    __print("setCallForwardParam");
```



```
}
```

In file HlrBL--handleCallForward.ipc

```
VOID handleCallForward()  
{  
    setCallForwardParam();  
}
```

Converted Java code examples are as follows:

In file HandleConfigData.java

```
public class HandleConfigData {  
    public static void setCallForwardParam() {  
        System.out.println("setCallForwardParam");  
    }  
}
```

In file HandleCallForward.java

```
public class HandleCallForward {  
    public static void handleCallForward() throws  
        FDSStandardException {  
        setCallForwardParam();  
    }  
}
```

Manually add import statement as follows:

```
Import static com.ericsson.pg.ca.hlrbl.HandleConfigData.*;  
public class HandleCallForward {  
    public static void handleCallForward() throws  
        FDSStandardException {  
        setCallForwardParam();  
    }  
}
```

4.2.1.4 Try-Catch Method

DUP can use a try-catch method to some sentences without exception but Java cannot.

For example:



The DUP code example is as follows:

```
Declare oDBPB IA5String;
Declare oDBPB1 BOOLEAN;
try{
    __print ("Fetch ODBPB1 existing values from HLR");
    if (String::matchRe ("NOBOC ", oDBPB))
    {
        oDBPB1 ::= FALSE;
    }
} catch (ex FDS.FDSStandardException){
    __print ("exception here...");
}
```

The converted Java code example is as follows:

```
String oDBPB = " " ;
boolean oDBPB1 = false;
try{
    System.out.println("Fetch ODBPB1 existing values from HLR");
    if(DUString.matchRe("NOBOC ",oDBPB)){
        oDBPB1 = false;
    }
} catch(FDSStandardException ex){
    System.out.println("exception here...");
}
```

Update the preceding example as follows, because there is no exception in try block.

```
String oDBPB = " " ;
boolean oDBPB1 = false;

System.out.println("Fetch ODBPB1 existing values from HLR");
if(DUString.matchRe("NOBOC ",oDBPB)){
    oDBPB1 = false;
}
```

4.2.1.5 Assignment between Different MOs

After the migration, the complicated type parameter in DUP ANS.1 is converted into a JavaBean. If the complicated type parameter does **Assign** operation or other operations between two different MOs in DUP code, modify the converted JDV logics as follows:

1. Convert the JavaBean to the Document object.
2. Convert the Document object to the new JavaBean.



For example, a parameter `HSS.PrivateUserIdState` has been defined in MO IMS and HSS:

```
declare privateUserIdStateData      HSS.PrivateUserIdState;  
declare new_privateUserIdStateData  IMS.PrivateUserIdState;
```

And it has the **Assign** operation in DUP code as follows:

```
new_privateUserIdStateData := privateUserIdStateData;
```

In the converted JDV, the JDV logic procedure must be:

```
try {  
    Document doc = JAXBUtils.bean2Xml(PackagePrivate.class,  
        privateUserIdStateData);  
    com.ericsson.pg.ca.imssv.IMS.PrivateUserIdState  
    new_privateUserIdStateData = JAXBUtils.xml2Bean(  
        com.ericsson.pg.ca.imssv.IMS.PrivateUserIdState.class, doc);  
    } catch (JAXBException | TransformerException | XMLStreamException e) {  
        //  
    }  
}
```

4.2.2 Runtime Error

Not all errors can be found in compiling steps, but some errors could cause wrong behavior during runtime.

- DUP supports IN/OUT parameters while Java does not support them.

For example, in DUP code:

```
VOID change(str I5AString)  
{  
    str+="change";  
}
```

After calling the previous function `change`, the original parameter `str` is changed to `str+"change"`. But Java does not support the IN/OUT parameter `str`. This issue cannot be found during compiling step. To support the same method in Java, manually update the code in Java as follows:

```
String change(String str){  
    Return str+"change"  
}
```

- Enum inputs for `isDefined()` function causes endless loop issue due to the limitation of the function itself.

For example, in Java code:



```
@XmlType(name = "ODBIC")
@XmlEnum
public enum ODBIC {
    @XmlEnumValue("0")
    nOBIC("0"),
    @XmlEnumValue("1")
    bAIC("1"),
    @XmlEnumValue("2")
    bICROAM("2");
}
if (isDefined(theMo.getODBIC())) {...}
```

As the value of ODBIC is Enum, the condition runs into an endless loop. To resolve the issue, remove the `isDefined()` function from the condition as follows:

```
if (theMo.getODBIC() != null) {...}
```

4.2.3 Expand JDV to Support More Functions

4.2.3.1 Use Standard Link Connector in Resource JDV

Resource layer JDV logics are able to use the standard link connector through Link Proxy implementation. Two kinds of standard link connectors proxy are provided for LDAP and CAI3G connectors.

In converted Resource Layer JDV projects, the default `NELogicHandler` gets the response from the connector as follows:

```
NELogicHandler logicHandler = new NELogicHandler();
```

To support New link proxy, manually modify the code according to the connectors to be used:

- If this Resource Layer JDV project is going to use CAI3G connectors, replace the statement in `MmlDispatcher.java` with the new statement as follows:

```
NELogicHandler logicHandler = new NELogicHandlerCustom
erExtension (new CAi3gJCA());
```

And add the import statement as follows to the updated Java file:

```
import com.ericsson.jdv.fw.resource.NELogicHandlerCus
tomerExtension;
```

```
import com.ericsson.jdv.fw.outbound.proxy.CAi3gJCA;
```



- If this Resource Layer JDV project is going to use LDAP connectors, replace the statement in `MmlDispatcher.java` with the new statement as follows:

```
NELogicHandler logicHandler = new NELogicHandlerCustomExtension (new LdapJCA ());
```

And add the import statement as follows to the updated Java file:

```
import com.ericsson.jdv.fw.resource.NELogicHandlerCustomExtension;
```

```
import com.ericsson.jdv.fw.outbound.proxy.LdapJCA;
```

And the `rootDN()` information can be implemented by the following two ways.

- Hard code `rootDn()`.
 - Add a configurable parameter `rootDN` on GUI. For detailed steps, see Section 4.4.2 on page 21.
- Other connectors are not supported by DUP Migration in current version.

4.2.3.2 Support `xsi:nil=true`

4.2.3.2.1 Support `xsi:nil=true` for a Simple Type

Do the following steps to support `xsi:nil=true` for a simple type (such as String and Integer) in JavaBean:

Note: `epsIndividualDefaultContextId` in `EPSPMultiSC` is used as an example in the following steps.

1. Modify the definition of `epsIndividualDefaultContextId` in `EPSPMultiSC.java`:

Modify the following code:

```
protected String epsIndividualDefaultContextId;
```

To

```
@XmlElementRef(name = "epsIndividualDefaultContextId",
    type = JAXBElement.class, required = false)
protected JAXBElement<String> epsIndividualDefaultContextId;
```

2. Add annotations to class `EPSPMultiSC`:



```
@XmlRootElement(name = "EpsMultiSC")
@XmlRegistry
public class EpsMultiSC
    implements Serializable
{
    ...
}
```

3. Add XmlElementDecl of epsIndividualDefaultContextId in EPSMultiSC.java:

```
private final static QName _epsIndividualDefaultContextId_QNAME = new QName(
    "", "epsIndividualDefaultContextId");
/**
 * Create an instance of {@link JAXBElement} {@code <{@link String }{@code >}}
 *
 */
@XmlElementDecl(namespace = "", name = "epsIndividualDefaultContextId",
    scope = EpsMultiSC.class)
public JAXBElement<String> createEpsMultiSCEpsIndividualDefaultContextId(String value) {
    return new JAXBElement<String>(_epsIndividualDefaultContextId_QNAME,
        String.class, EpsMultiSC.class, value);
}
```

4. Modify the Getter and Setter of epsIndividualDefaultContextId in EPSMultiSC.java:

Modify the Getter code to:

```
public JAXBElement<String> getEpsIndividualDefaultContextId() {
    if (epsIndividualDefaultContextId == null) {
        epsIndividualDefaultContextId = createEpsMultiSCEpsIndividualDefaultContextId("");
    }
    return epsIndividualDefaultContextId;
}
```

Modify the Setter code to:

```
public void setEpsIndividualDefaultContextId(JAXBElement<String> value) {
    this.epsIndividualDefaultContextId = value;
}
```

5. Set nil=true in Service JDV:

```
theMo.getEpsIndividualDefaultContextId().setValue(null);
```

Note: If setting the element `nillable` in sub-MO, its `XmlElementDecl` must be added to its root element class.

4.2.3.2.2 Support xsi:nil= true for a Complex Type

Adding the following Java code in Service JDV to support `xsi:nil=true` for a complex type (such as `List<String>`) in JavaBean:

Note: `epsIndividualContextId` in `EPSMultiSC` is used as an example in the following code.



```
theMo.getEpsIndividualContextId().add(null);
```

4.2.3.3 Call Standard JDVs in SV

Migration tool generates an API such as `ServiceLayer.process(String operation, Object mo)` to call the Service JDV logic. If the Service JDV is a standard JDV in Dynamic Activation, this API needs to be manually replaced by a new API `ServiceLayer.process(QName trigger, String operation, Object mo)`. This `QName` trigger is a standard JDV trigger.

Modify the original converted code:

```
hlrMoRef = (com.ericsson.pg.ca.gsmhmr.GsmHmr.HmrSubscription) DU.cloneDU(ServiceLayer.process("Get", hlrMo));
```

To:

```
QName qname = new QName("http://schemas.ericsson.com/ema/UserProvisioning/GsmHmr/", "GetSubscription");
```

```
hlrMoRef = (com.ericsson.pg.ca.gsmhmr.GsmHmr.HmrSubscription) DU.cloneDU(ServiceLayer.process(qname, "Get", hlrMo));
```

4.2.3.4 Customize Request or Response of Standard JDV in SV

Follow these steps to customize request or response of standard JDV in SV:

1. Create a class to implement the interface `RequestHandler` or `RspnseHandler` and modify the method handler (`Request<Document> request`) according to the requirement. The following example aims to reverse the order of elements `imsi` and `msisdn` to compliance with the XML schema.

```
public class SetMsisdnAndImsiOrderHandler implements RequestHandler {
    @Override
    public void handle(Request<Document> request) {
        if(request == null || request.getPayload() == null){
            return;
        }

        Document doc = request.getPayload();
        Element rootEle = doc.getDocumentElement();
        NodeList children = rootEle.getChildNodes();

        Element msisdn = XMLUtil.getFirstChildElementByLocalName(rootEle, "msisdn");
        Element imsi = XMLUtil.getFirstChildElementByLocalName(rootEle, "imsi");

        if(msisdn != null && imsi != null){
            rootEle.removeChild(msisdn);
            rootEle.insertBefore(msisdn, imsi);
        }
    }
}
```

2. Apply the new class created in Step 1 into DUP JDV:



```

public void main(String theOperation, Subscription theMo,
BaseObject theBaseObject, Filter theFilter, Scope theScope) throws FDSStandardException {
...
RequestHandler hlrRequestHandler = new SetMsisdnAndImsiOrderHandler();
QName qname = new QName("http://schemas.ericsson.com/ema/UserProvisioning/GsmHlr/", "GetSubscription");
hlrMoRef = (com.ericsson.pg.ca.gsmhmr.GsmHmr.HlrSubscription) DU.cloneDU(ServiceLayer.process(
    qname, "Get", hlrMo, hlrRequestHandler, null));
//process a get operation and return the result java bean
...
}

```

4.2.4 API and Primitive Type Support

Only partial DUP APIs are provided in new Dynamic Activation JDV platform, the description of supported APIs and primitive type can be found in *DUP Migration API Reference Manual*, Reference [4]. Other unsupported DUP APIs must be implemented in Java by CA project according to original DUP API description in Reference [5].

4.3 Error Mapping

This section shows how to map errors manually in Dynamic Activation to support the format used in Classic Multi Activation.

Although both Classic Multi Activation and Dynamic Activation support CAI3G 1.2 Northbound Interface, they have different error message formats, which can cause failure for Customer Administration System (CAS) in parsing error messages.

The following text in bold shows the difference in the formats. Classic Multi Activation uses the tag of **UserProvisioningFault** while Dynamic Activation uses **PGFault** by default.

```

<Fault>
  <faultcode>S:Server</faultcode>
  <faultstring>error,please check detail</faultstring>
  <detail>
    <Cai3gFault xmlns="http://schemas.ericsson.com/cai3g1.2/">
      <faultcode>3010</faultcode>
      <faultreason>
        <reasonText>Internal fatal error.</reasonText>
      </faultreason>
      <faultrole>MF</faultrole>
    </Cai3gFault>
    <details>
      <ema:UserProvisioningFault xmlns:ema="http://schemas.ericsson.com/ema/UserProvisioning/">
        <ema:respCode>1000</ema:respCode>
        <ema:respDescription>something wrong</ema:respDescription>
      </ema:UserProvisioningFault>
    </details>
  </detail>
</Fault>

```

Example 1 Classic Multi Activation Error Message Format



```
<Fault>
<faultcode>ns2:Server</faultcode>
<faultstring>This is a server fault</faultstring>
<detail>
  <Cai3gFault:Cai3gFault xmlns:Cai3gFault="http://schemas.ericsson.com/cai3g1.2/">
    <faultcode>4005</faultcode>
    <faultreason>
      <reasonText>Internal fatal error.</reasonText>
    </faultreason>
    <faultrole>MF</faultrole>
    <details>
      <PGFault:PGFault xmlns:PGFault="http://schemas.ericsson.com/pg/1.0">
        <errorcode>1000</errorcode>
        <errorMessage>something wrong</errorMessage>
        <errordetails>something wrong - [Processed by PG Node: node1]</errordetails>
      </PGFault:PGFault>
    </details>
  </Cai3gFault:Cai3gFault>
</detail>
</Fault>
```

Example 2 Dynamic Activation Error Message Format

Dynamic Activation does not support the customization of all content in details but use namespace to distinguish between Classic Multi Activation error format and Dynamic Activation error format.

Table 1 Namespace

Dynamic Activation namespace	Other than the namespace of Classic Multi Activation
Classic Multi Activation namespace	http://schemas.ericsson.com/ema/UserProvisioning/

According to Table 1, if Motype namespace starts with `http://schemas.ericsson.com/ema/UserProvisioning/`, for example, Subscription@http://schemas.ericsson.com/ema/UserProvisioning/DNS, the error format is Example 1. Otherwise the error format is Example 2.

4.4 Configuration Data Support

4.4.1 Configuration Data Support in Service JDV

The following procedure is an example to show how to support configuration data in Service JDV in Dynamic Activation:

1. Check the DUP project whether it supports configuration data.

If method `ConfigData::register()` is called in initial files in the DUP project, manually update JDV to support configuration data according to the following steps.

For example, `ConfigData::register("CRBT_PARAMS")` means CRBT_PARAMS is the parameter in Configuration Data.



2. For each parameter of configuration data, add the following functions to the java file `<NElogic>JavaDataViewFactory.java` in the service JDV project:

- `public String get<Parameter Name>()`
- `public void set<Parameter Name>(String value)`

The following is a template of the two functions:

```
@ConfigurationProperty(description = "Get <Parameter Name>")
public String get<Parameter Name>() {
    return configData.get("<Parameter Name>");
}

public void set<Parameter Name>(String value) {
    configData.put("<Parameter Name>", value);
}
```

The following is an example for the template:

```
@ConfigurationProperty(description = "Get CRBT_PARAMS")
public String getCRBT_PARAMS() {
    return configData.get("CRBT_PARAMS");
}

public void setCRBT_PARAMS(String value) {
    configData.put("CRBT_PARAMS", value);
}
```

4.4.2 Configuration Data Support in Resource JDV

The following procedure is an example on how to support configuration data in Resource JDV in Dynamic Activation:

1. For each parameter of configuration data, add the following functions to the Java file `<NElogic>JavaDataViewFactory.java` in the resource JDV project:

- `public String get<Parameter Name>()`
- `public void set<Parameter Name>(String value)`

The following is a template of the two functions:



```

@ConfigurationProperty(description = "Get <Parameter Name>")
public String get<Parameter Name>() {
    return configData.get("<Parameter Name>");
}
public void set<Parameter Name>(String value) {
    configData.put("<Parameter Name>", value);
}

```

The following is an example for the template:

```

@ConfigurationProperty(description = "Get RootDn")
public String getRootDn() {
    return configData.get("RootDn");
}
public void setRootDn(String value) {
    configData.put("RootDn", value);
}

```

2. Add the following APIs to use the configuration data in Resource JDV dynamically:

ConfigDataResourceLayer.get(String name, NamedValues out): to get the config data as namedValues which is separated by a comma
 ConfigDataResourceLayer.get(String name): to get the config data as a string

4.5 CAI3G Interface Definition

The Migration Tool supports to migrate the original DUP project to a JDV project that provides a CAI3G interface by default. The situation is divided into two situations after migration is finished:

The original DUP project is CAI3G-based

The original WSDL and schema can be reused directly.

The original DUP project is CAI-based

Create a WSDL and schema file for generated code according to document *Generic CAI3G Interface 1.2*, Reference [3]. There is a local schema file in the `jaxb` package in generated code and the file can be used as a reference:

4.6 Access Control Support

Follow the steps in section *Define the JDV Access Control Model in Customer Adaptation Development Guide for Resource Activation*, Reference [2].

4.7 Special Routing Attribute (Optional)

Dynamic Activation supports `msisdn` and `imsi` as JDV resource key attribute by default during routing configuration. To support other keys, add configuration in MO identities according to section *Define JDV Resource Key Attribute (Optional) in Customer Adaptation Development Guide for Resource Activation*, Reference [2].



Note: `Public static <T> void lookup(T theMo, List<ElementType> dests)` retrieves resource key attribute from the input JavaBean by adding annotation in `XmlAttribute`. That means only the key defined in JavaBean can be used as the resource key attribute. For example, in following definition only `imsi` can be used as the key.

```
public class Subscription
    implements Serializable
{
    @XmlAttribute(name = "imsi")
    protected String imsi;
    protected String msisdn;
    protected String newmsisdn;
    protected String servArea;
    protected String profile;
    ...
}
```

Example 3 *imsi as the Resource Key*

But in the following example, both `imsi` and `newmsisdn` can be used as the resource key.

```
public class Subscription
    implements Serializable
{
    @XmlAttribute(name = "imsi")
    protected String imsi;
    protected String msisdn;
    @XmlAttribute(name = "newmsisdn")
    protected String newmsisdn;
    protected String servArea;
    protected String profile;
    ...
}
```

Example 4 *imsi and newmsisdn as the Resource Key*

4.8 Loose Error Handling Support (Optional)

Loose error handling in Dynamic Activation is optional in JDV. Follow the steps in section Define JDV Loose Error Handling (Optional) in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

4.9 CAI3G DC Support

To dispatch a request to the remote CAI3G server through CAI3G Distributed Configuration (DC) connector, the CAI3G DC routing needs to be configured on GUI. Ensure that both the process method parameters (such as `qname`) in class `com.ericsson.jdv.dup.apis.ServiceLayer.class` and CAI3G DC routing needs to be defined regarding to the CAI3G request of remote CAI3G server.

The following is an example of the `QName` in `com.ericsson.jdv.dup.apis.ServiceLayer.class`:



```
...
QName qname = new QName("http://schemas.ericsson.com/ema/UserProvisioning/GsmHlr/", "CreateSubscription");
ServiceLayer.process(qname, "Create", hlrMo, new SetMsisdnAndImsiOrderHandler(), null);
...
```

4.9.1 Handling CAI3G DC External Fault

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

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

```
FDSStandardException.errorCode = 20000
FDSStandardException.hint = "<S:Envelope xmlns:S='http://schemas.xmlsoap.org/soap/envelope/'
xmlns:cai3g='http://schemas.ericsson.com/cai3gl.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/cai3gl.2/'
xmlns:Cai3gFault='http://schemas.ericsson.com/cai3gl.2/'>
          <faultcode>3013</faultcode>
          <faultreason>
            <reasonText>Invalid parameter.</reasonText>
          </faultreason>
          <faultrole>NEF</faultrole>
        </Cai3gFault:Cai3gFault>
        <UserProvisioningFault:UserProvisioningFault xmlns='http://schemas.ericsson.com/ema/UserProvisioning/'
xmlns:UserProvisioningFault='http://schemas.ericsson.com/ema/UserProvisioning/'>
          <respCode>1006</respCode>
          <respDescription>Invalid parameter. The XML data is not valid. cvc-enumeration-valid: Value 'ONLY'
            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 previous example, `errorCode` of `FDSStandardException` is 200000 which is the error code for CAI3G DC external fault. The `hint` of `FDSStandardException` is a string that contains the complete fault message from the remote CAI3G server.



4.10 Building and Deploying a CA JDV

4.10.1 Building a CA JDV

For detailed steps, refer to section *Building JDV* in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

Two .gz packages are generated for one Service JDV project.

- Business Logic Package

For example:

```
JDV-ScmCucsubSV-Provisioning-JavaBean-1.0.277-SNAPSHOT.  
tar.gz
```

- JavaBean Package

For example:

```
JDV-ScmCucsubSV-Provisioning-Logic-1.0.277-SNAPSHOT.  
tar.gz
```

4.10.2 Deploying a CA JDV

For details about deploying business logic package, refer to section *Deploying JDV* in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

For details about deploying JavaBean package, refer to section *Add 3pp Jar Files to the System* in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

When an SV JDV project depends on another MVNE JDV project and this MVNE JDV project has been already deployed on Dynamic Activation before 16.0 CP2, to convert the SV project successfully, this MVNE JDV project needs to be redeployed according to the following steps:

1. Undeploy this MVNE JDV project from Dynamic Activation, see Section 4.10.3 on page 25.
2. Convert the MVNE DUP project to a new MVNE JDV project by using the migration tool released in 16.0 CP2 or higher version.
3. Copy the Java source code of the original MVNE JDV project to the new MVNE JDV project.
4. Build and Deploy the new MVNE JDV project to Dynamic Activation, see Section 4.10.2 on page 25.



4.10.3 Undeploying a CA JDV

For details about undeploying business logic package, refer to section *Undeploying JDV* in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].



5 CA Development for Connector

5.1 Preparing for a Connector Project

5.1.1 Creating a Connector Project

To create a Connector project:

1. Open Eclipse and select **File > New > Maven Project** from the menu bar.
2. Click **Create a simple project**, and click **Next**.
3. Fill in **Group Id** and **Artifact Id**.
4. Click **Finish**.

5.1.2 Importing Files into a Connector Project

To import files into a Connector project:

1. Copy all Java files from the original CA Connector project to the Connector project.
2. Copy the `assembly.xml` file and `pom.xml` file from `C:\CA_Development\dup-migration-1.0.0-tool\resource\template\connector-template` to the Connector project.
3. Edit the value of `<artifactId>` and `<name>` in the `pom.xml` file of the Connector project. The following is an example of the `pom.xml` file:

```
<artifactId>ca-java-link</artifactId>
<packaging>jar</packaging>
<name>ca-java-link</name>
<url>http://maven.apache.org</url>
<properties>
  <enforcer.skip>true</enforcer.skip>
</properties>
```

4. Update the Connector project in Eclipse. The imported Java files are visible. Update the Java files as follows, and fix code issues accordingly:
 - a Change `implements DownStreamConnection` to `extends AbstractConnector`.

For example, `public class TemplateResp extends AbstractConnector { }`
 - b Change `DownStreamException` to `Exception`.



For example, `public String sendMessage(String mmlString)` throws `Exception {}`

Note: If heartbeat functionality is needed, method connection can be implemented with detailed logic.

5.1.3 Creating a ConnectorFactory Class

To create a `ConnectorFactory` Class:

1. Create a Class file of Connector factory `<ca-connector-factory>.java` in the Connector project.
2. Implement the `<ca-connector-factory>.java` according to `C:\CA_Development\dup-migration-1.0.0-tool\resource\template\connector-template\src\main\java\com\ericsson\connector_template\TemplateRespFactory.java`. Override the `Create` method and return the new Connector.
3. Update `@ConnectorDescription` in `<ca-connector-factory>.java` according to the following format:

```
@ConnectorDescription(displayName = "<connector name>",
connectorType = "<connector type>", version = "<version>",
connectorInterface = AbstractConnector.class)
```

The following is an example:

```
@ConnectorDescription(displayName = "test http", connectorType = "http", version = "1.0", connectorInterface = AbstractConnector.class)
```

4. Update the value of `<Connector-Factory-Class>` in the `pom.xml` file of the CA Connector project. The following is an example:

```
<Connector-Factory-Class>com.ericsson.javalink.handler.CAJavaLinkFactory</Connector-Factory-Class>
```

5.2 Building and Deploying a CA Connector

5.2.1 Building a CA Connector

In a command prompt window, run the following commands to go to the CA Connector project directory where the `pom.xml` file is located, and compile the CA Connector project with Maven:

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

```
$ mvn clean package
```




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

5.2.2 Deploying a CA Connector

To deploy a CA Connector project:

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

2. Change the owner and the group of the CA Connector `tar.gz` files.

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

3. Do the following steps on each Processing Layer (PL) node where a CA Connector is to be installed:

- a. Add the CA Connector to the PL node by running the following command on a System Controller (SC) node:

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

Note: `<hostname>` is the hostname of the PL node to which the CA Connector is added.

- b. Activate the PL node by running the following command on an SC node:

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

Note: `<hostname>` is the hostname of the PL node to which the CA Connector is activated.

For example, run the following commands on an SC node to install and activate `JavaLink-1.0.0.tar.gz` on PL-3:

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

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

4. Check the deployment result.

If the link is deployed successfully, the display name of the link is available in the protocol candidate list on the **Add Network Element wizard in the Network Element** tab.



For more information on how to use the GUI, refer to section *Configuring Dynamic Activation for CA Provisioning in Customer Adaptation Development Guide for Resource Activation, Reference [2]*.

If the display name of the link cannot be found in the protocol candidate list, check `server.log` to find the cause of deployment failure.

5.2.3 Undeploying a CA Connector

To undeploy a CA Connector project, do the following steps for each PL node where a CA Connector is to be removed:

1. Run the following command on an SC node to remove the CA Connector from the PL node:

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

In the preceding command, `<hostname>` is the hostname of the PL node from which the CA Connector is deleted.

2. Run the following command on an SC node to activate the PL node:

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

In the preceding command, `<hostname>` is the hostname of the PL node.

Note: The parameter `all` is only used for new installations. It cannot be used in any other cases (such as activation of a new CA Connector).

For example, run the following commands, from an SC node, to undeploy `JCA-Custom3` on `PL-3`:

```
$ bootloader.py submodule delete -n JCA-Custom3-1  
.0.0.tar.gz -p dve-application --host PL-3
```

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



6 Configuring Dynamic Activation for CA Provisioning

For details, refer to section *Configuring Dynamic Activation for CA Provisioning* in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

In **Configuration Data**, if one parameter has several subparameters, use comma (",") between subparameters. The following figure shows an example:

The screenshot shows a web-based configuration interface. At the top, there are several tabs: 'Activation Logic', 'Network Elements', 'System', 'Access Control', 'Walkthroughs', and 'Log Management'. The 'Activation Logic' tab is selected. Below this, there is a sub-tab 'Options'. The main content area is titled 'Activation Logic - Change - JDV-Hua-Provisioning.jar'. Under this title, there is a section labeled 'Configuration Data'. Within this section, there is a label 'CRBT_PARAMS:' followed by a text input field. The input field contains the text 'OperatorAcc=bbb,OperatorPwd=ccc,Ope'. At the bottom of the configuration area, there are two buttons: 'Apply' and 'Cancel'.

Figure 3 Data Configuration

When finished configuration in the GUI, log on to the SC-node 1 through SSH, and perform configurations described in the following subsections of *Customer Adaptation Development Guide for Resource Activation*, Reference [2]:

- Configuring Logging
- Manage Custom Processing Log Settings





7 Testing and Provisioning over CAI3G

Refer to section `Testing and Provisioning over CAI3G` in *Customer Adaptation Development Guide for Resource Activation*, Reference [2].

Note: Some defects cannot be detected after going through all the operations in the preceding sections. Therefore, customers must perform a whole function test to make sure the BL work as expected.





8 Existing JDVs and Connectors Supporting Multiple Versions

To enable multiple versions of NEs in a DUP CA project that is converted with DUP migration tool before 16.0 CP1, follow the instructions in the following sections.

8.1 CA DUP Migration

8.1.1 Preparing for a JDV Project

In the Resource JDV converted from the DUP CA project, follow these steps:

1. Update the file `<Directory of Resource JDV>/src/main/resources/META-INF/javadataview-descriptor.xml`.

Add a line of the new NE version supported by the JDV `<version><NE version></version>` below the following lines:

```
<!-- Name of value package this CA will be bound to -->
<valuePackage>Subscriber</valuePackage>
```

2. Update `pom.xml` in the Resource JDV.

Update the JDV names by adding the new NE version in the following lines:

```
<artifactId><JDV name></artifactId>

<name><JDV name></name>
```

For example, the original JDV name is as follows:

```
<artifactId>JDV-ExampleNL-Provisioning</artifactId>

<name>JDV-ExampleNL-Provisioning</name>
```

Add the new NE version as follows:

```
<artifactId>JDV-ExampleNL-Provisioning-v1.0</artifact
Id>

<name>JDV-ExampleNL-Provisioning-v1.0</name>
```



8.1.2 Building and Deploying a CA JDV

For details, refer to Section 4.10 on page 24.

8.2 CA Development for Connector

8.2.1 Preparing for a Connector Project

Add the following information to `<ca-connector-factory>.java` in the Connector project.

```
@ConfigurationParameter(description = "Version", mandatory  
= false)
```

```
private String version = null;
```

8.2.2 Building and Deploying a CA Connector

For details, refer to Section 5.2 on page 28.



Reference List

Ericsson Documents

- [1] *Library Overview*, 18/1553-CSH 109 628 Uen
- [2] *Customer Adaptation Development Guide for Resource Activation*, 5/1553-CSH 109 628 Uen
- [3] *Generic CAI3G Interface 1.2*, 2/15519-FAY3020003 Uen
- [4] *DUP Migration API Reference Manual*, 2/2134-CSH 109 628 Uen
- [5] *Multi Activation IDE Reference Manual*, 1/2134-CSH 109 434 Uen

External Links

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