

# Northbound Interface Adapter Customization Development Guide for HTTP-Based Protocol

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 Group	1
1.3	Typographic Conventions	1
1.4	Prerequisites	1
<b>2</b>	<b>Preparing for NBIA Development Environment</b>	<b>2</b>
2.1	IDE Installation	2
2.1.1	Prerequisites	2
2.1.2	Eclipse	2
2.2	Wizard	2
2.2.1	Installation of Wizard	2
<b>3</b>	<b>Tutorial Project Overview</b>	<b>3</b>
3.1	Adapted Northbound SOAP Interface	3
3.2	High-Level Use Cases	4
3.2.1	Creating a Subscription	6
3.2.2	Getting a Subscription	6
3.2.3	Setting a Subscription	6
3.2.4	Deleting a Subscription	6
<b>4</b>	<b>Northbound Interface Adapter Development</b>	<b>6</b>
4.1	Creating a Northbound Interface Wizard Project	7
4.2	Implementing the Northbound Interface	11
4.2.1	Project Structure	11
4.2.2	Northbound Interface Implementation	15
4.2.3	Web Services Implementation	16
4.2.4	Using Statistics Service	23
4.2.5	Packaging Northbound Adapter	26
<b>5</b>	<b>Deployment</b>	<b>27</b>
5.1	Administrating the Northbound Interface Adapter	27
5.1.1	Installing the Northbound Interface Adapter	27
5.1.2	Activate the Northbound Interface Adapter	28
5.1.3	Checking the Submodule Status	28
5.1.4	Uninstalling the Northbound Interface Adapter	28
5.1.5	Updating the Northbound Interface Adapter	28
5.2	Setting Properties for the Northbound Interface Adapter	29
5.2.1	Creating Properties for the Cluster Northbound Interface Adapter	30



5.2.2	Modifying Properties for the Cluster Northbound Interface Adapter	30
5.2.3	Restore the Default Value of the Property for the Cluster Northbound Interface Adapter	31
5.3	Configuring ESA Alarms	31
5.3.1	Adding the Alarm Definition	31
5.3.2	Configuring Alarm Resource ID	32
5.3.3	Raising an Alarm	33
5.4	Configuring Logs	33
5.5	Configuring Security	34
<b>6</b>	<b>Verifying the Northbound Adapter</b>	<b>34</b>
6.1	Creating a Subscription	34
6.2	Setting a Subscription	35
6.3	Getting a Subscription	36
6.4	Deleting a Subscription	37
6.5	Faults or Errors	38
	<b>Reference List</b>	<b>39</b>



# 1 Introduction

This document provides detailed instructions on how to develop, deploy, and verify customized Northbound Interface in the Northbound Adapter (NBIA) framework in Ericsson™ Dynamic Activation (EDA).

## 1.1 Purpose and Scope

This document is intended for the Customer Adaptation (CA) developers who develop the Northbound Interface with the HTTP-based protocol in NBIA, targeting provisioning integration with customer's Business Support System Northbound Interface.

## 1.2 Target Group

The target group for this document is as follows:

- System integrators
- Application designers

## 1.3 Typographic Conventions

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

## 1.4 Prerequisites

This document is written with the assumption that the users:

- Have good knowledge of Dynamic Activation, Java™ language, and Eclipse.
- Have read the following documents:
  - *Customization - Architectural Overview*, Reference [2]
  - *Northbound Interface Adapter Reference Manual*, Reference [3]



## 2 Preparing for NBIA Development Environment

### 2.1 IDE Installation

#### 2.1.1 Prerequisites

The NBIA module is written in Java. It is recommended to develop NBIA module with Eclipse Integrated Development Environment (IDE) and build them in Maven.

For how to prepare JDK and Maven, refer to *Customer Adaptation Development Guide for Resource Activation*, Reference [4].

#### 2.1.2 Eclipse

The NBIA design base project is general Maven project. The instruction is based on Eclipse, which is the recommended IDE. Kepler is the supported version of Eclipse.

For how to prepare Eclipse environment, refer to *Customer Adaptation Development Guide for Resource Activation*, Reference [4].

### 2.2 Wizard

Wizard is an Eclipse plug-in for generation of NBIA design base project.

#### 2.2.1 Installation of Wizard

Before installing the tool, make sure that Eclipse is installed (all releases from 4.3 are supported).

To install Wizard, perform the following actions:

1. Close Eclipse if it is running.
2. Copy the following files to the `plugins` directory of Eclipse installation folder:
  - `com.ericsson.ca.nbi-<version>.jar`
  - `com.ericsson.ca.nbi.feature-<version>.jar`



**Note:** Those files can be found in the NBIA folder of the `ca-plugins-<version>.tar.gz` package.

### 3. Start Eclipse.

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

## 3 Tutorial Project Overview

This tutorial project is based on a customized northbound SOAP protocol using Home Subscriber Server (HSS) Evolved Packet System (EPS) data model as an example.

### 3.1 Adapted Northbound SOAP Interface

In this tutorial, a SOAP-based Northbound Interface is defined. The interface includes the `Create`, `Set`, `Get`, and `Delete` operations.

For the `Create` and `Set` operations, there are two elements `Key` and `Payload`. `Key` and `Payload` elements can contain any XML segments. In this tutorial, HSS EPS is used as a concrete example.

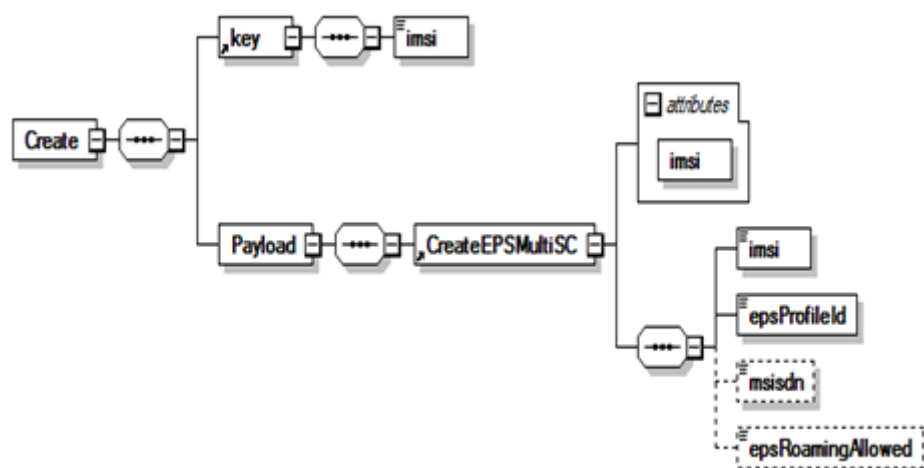


Figure 1 Create Subscription

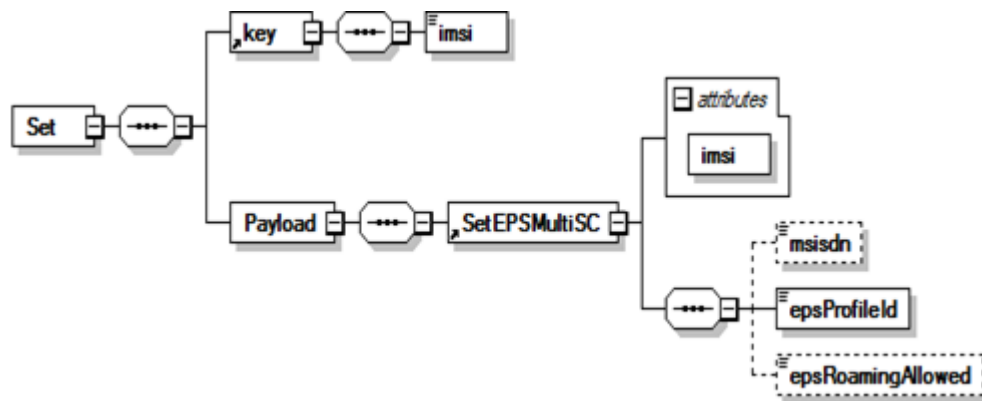


Figure 2 Set Subscription

For the Get and Delete operations, there is only one element Key.

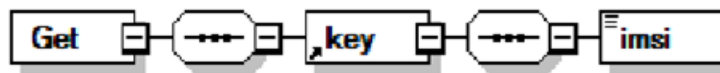


Figure 3 Get Subscription

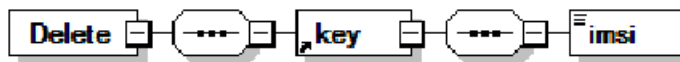


Figure 4 Delete Subscription

The NBIA must focus on the conversion of protocols only instead of any specific provisioning logic. HSS EPS is used here for easy understanding.

## 3.2 High-Level Use Cases

The following figure shows the workflow of processing high-level use cases.

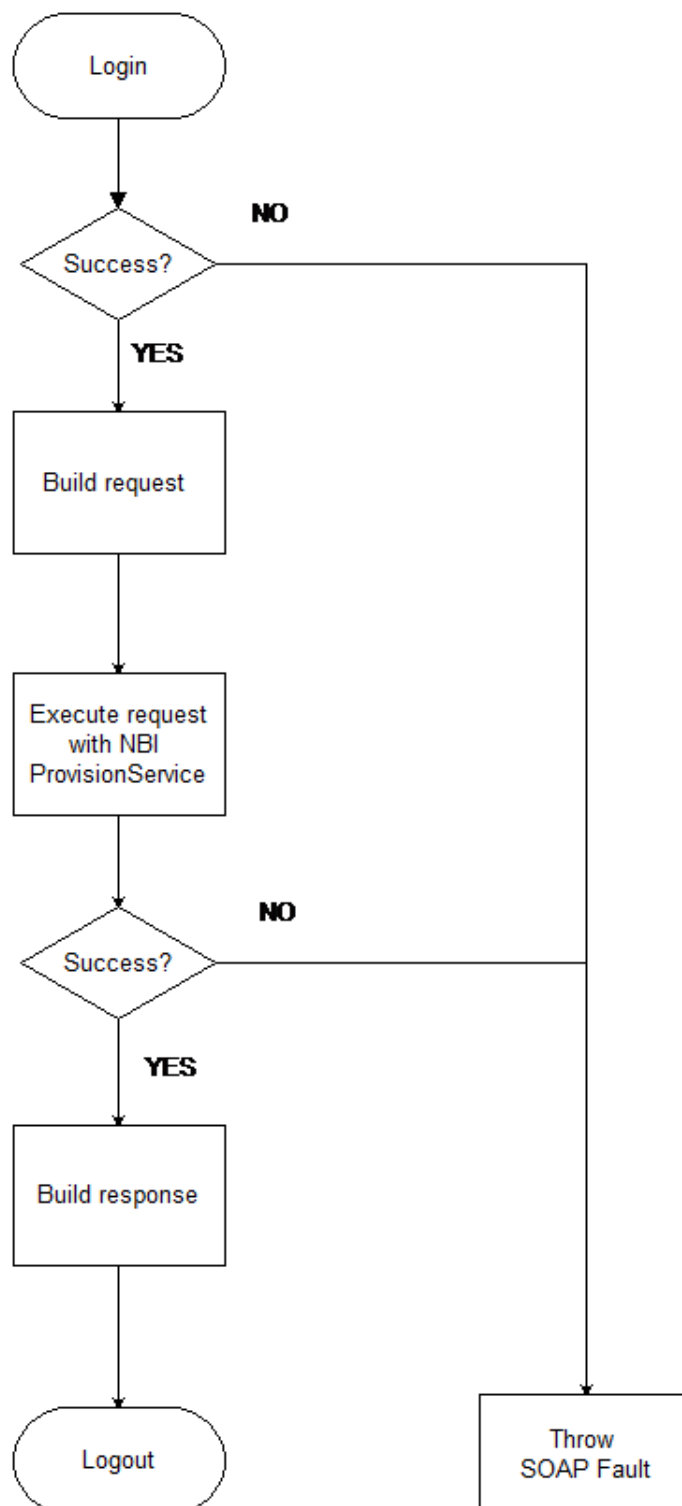


Figure 5 High-Level Use Cases



### 3.2.1 Creating a Subscription

This use case creates a subscription in operator's network. A Northbound Interface request with Mobile Subscriber ISDN Number (MSISDN), International Mobile Subscriber Identity (IMSI), and other profile IDs is sent to Dynamic Activation to trigger the provisioning towards HSS EPS.

### 3.2.2 Getting a Subscription

This use case gets an HSS EPS subscription in operator's network. A Northbound Interface request with MSISDN, IMSI, and other profile IDs are sent to Dynamic Activation to trigger the provisioning towards HSS EPS.

### 3.2.3 Setting a Subscription

This use case sets a subscription in operator's network. A Northbound Interface request with MSISDN, IMSI, and other profile IDs are sent to Dynamic Activation to trigger the provisioning towards HSS EPS.

### 3.2.4 Deleting a Subscription

This use case deletes a subscription in operator's network. A Northbound Interface request with MSISDN, IMSI, and other profile IDs are sent to Dynamic Activation to trigger the provisioning towards HSS EPS.

## 4 Northbound Interface Adapter Development

Based on the Northbound Interface wizard, create new Northbound Interface for the provisioning towards HSS EPS according to the steps as follows:

1. Use adapted Simple Object Access Protocol (SOAP) Web Services Description Language (WSDL) as an example and support `Create`, `Get`, `Set`, and `Delete` (CRUD) operations with specific parameters and responses for HSS EPS.
2. Northbound Interface wizard automatically generates Java skeleton code according to adapted SOAP WSDL.
3. Implement Northbound Interface to handle adapted HSS EPS provisioning requests.

The following figure shows the function design of this adapter:

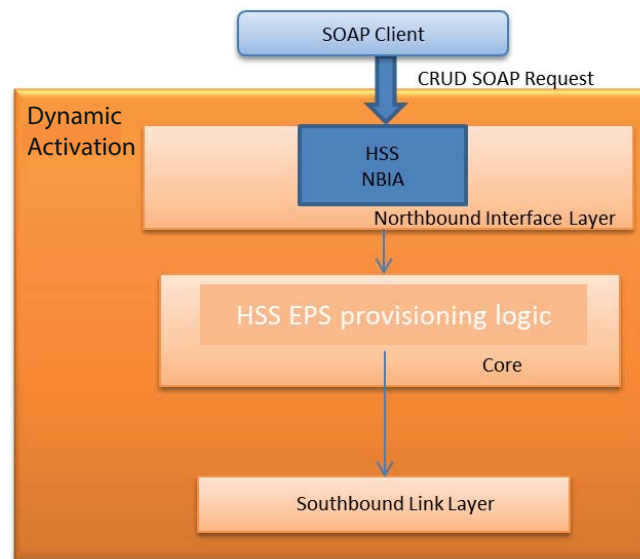
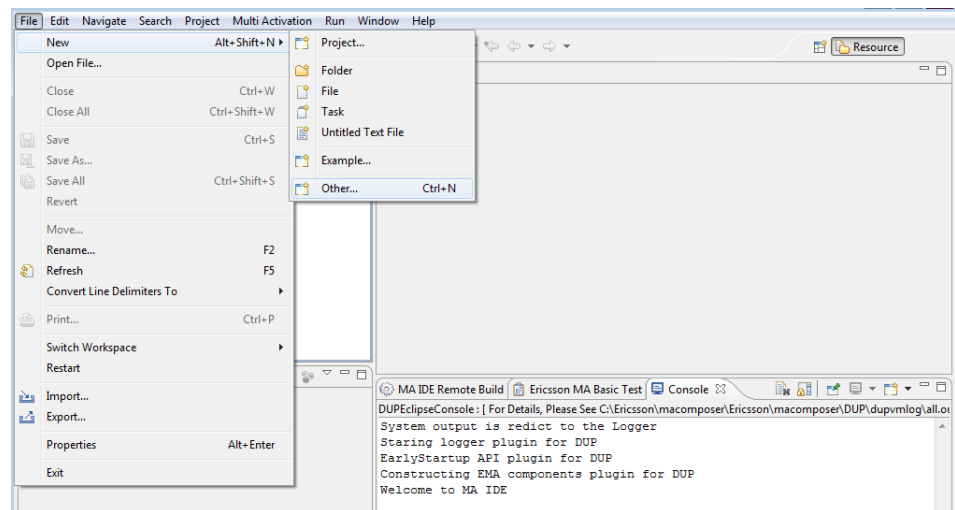


Figure 6 Northbound Interface Adapter Development Workflow

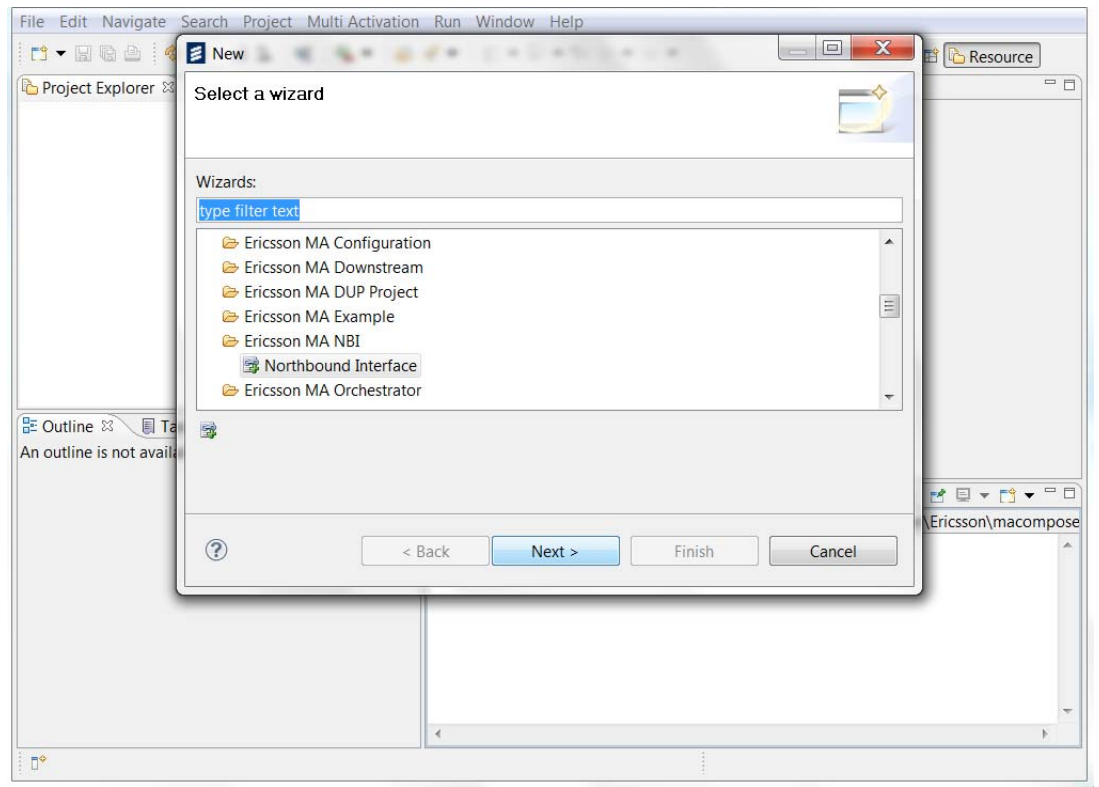
## 4.1 Creating a Northbound Interface Wizard Project

To create a Northbound Interface wizard project:

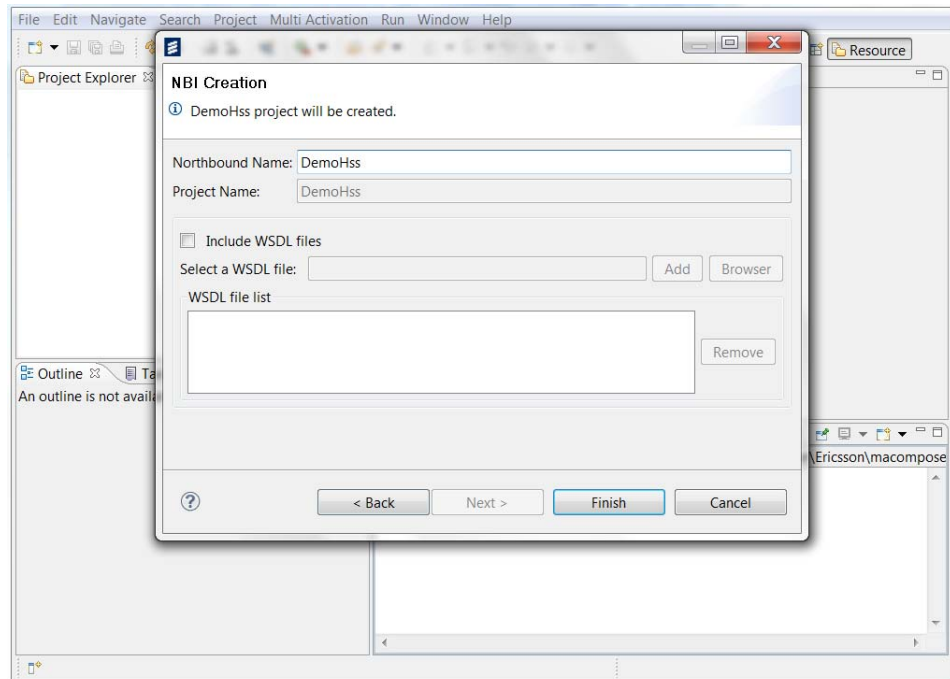
1. Start **Dynamic Activation IDE** and select **File > New > Other**.



2. In the **New** dialog, select **Northbound Interface** in the **Ericsson DA NBI** folder, and click **Next**.



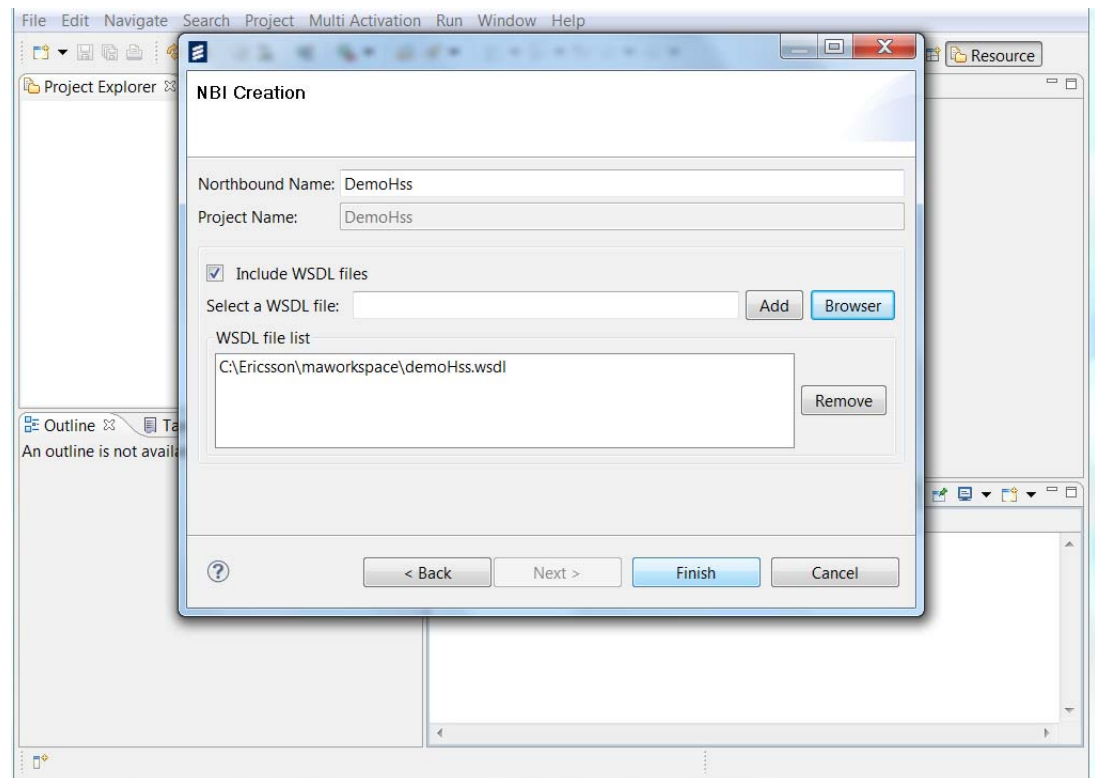
3. Specify a name in the **Northbound Name** field.



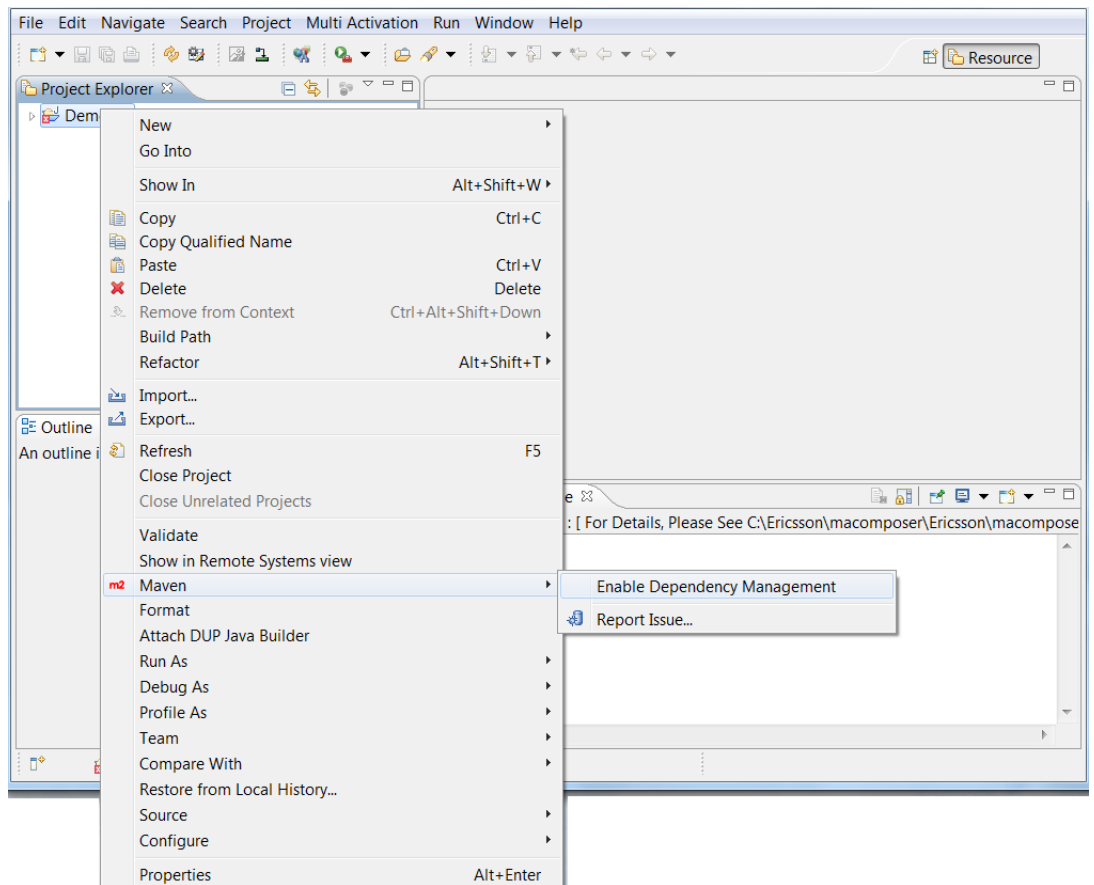


4. If WSDL files are needed, select **Include WSDL files** and specify the paths of WSDL files in the **Select a WSDL file** field. Click **Finish** to create the Northbound Interface project.

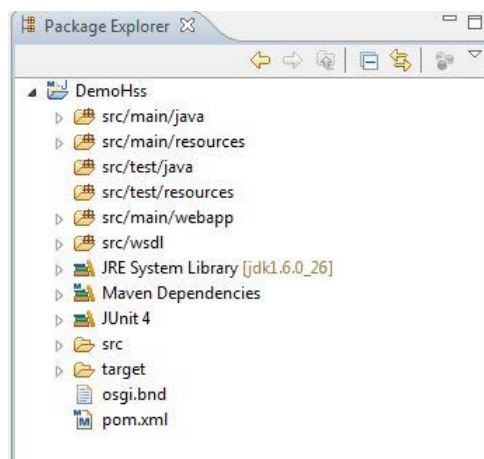
If the WSDL file is not needed, select it and click **Remove**.



5. In the **Project Explorer**, right-click the created Northbound Interface project and select **Maven > Enable Dependency Management**.



Now the Northbound Interface project is ready.





## 4.2 Implementing the Northbound Interface

### 4.2.1 Project Structure

The demo project `mpe-business-nbi-demoHSS` is an example project in this section. After the installation of Dynamic Activation IDE, find the demo project `mpe-business-nbi-demoHSS` in `C:\Users\<user>\.m2\repository\com\ericsson\mpe\mpe-business-nbi-demoHSS\<Version>\mpe-business-nbi-demoHSS-<Version>.jar`. Extract `mpe-business-nbi-demoHSS-<Version>.jar` to `business-nbi-demoHSS` in your workspace folder. The following figure shows the main project directory and configuration files:

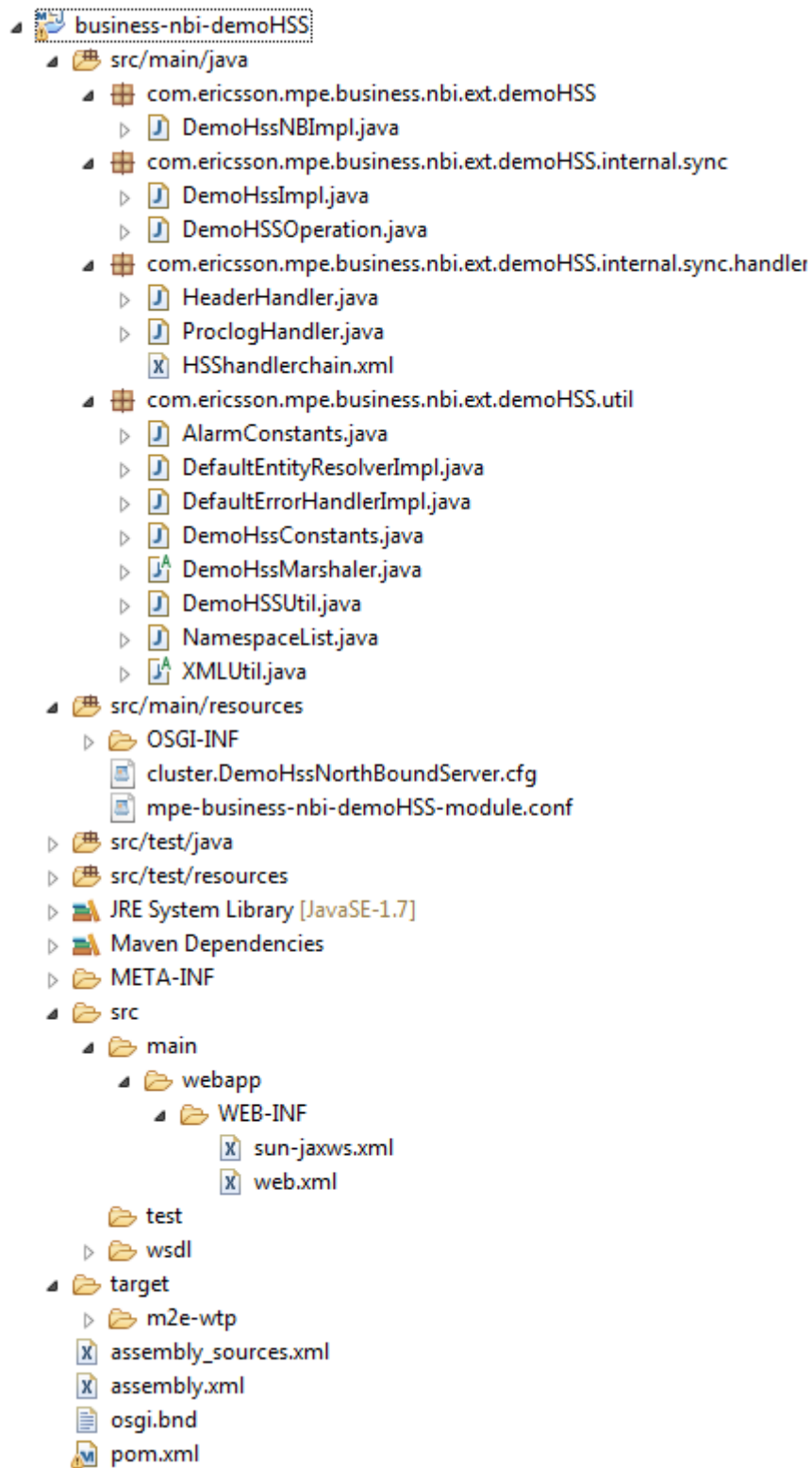


Figure 7 Project Structure



#### 4.2.1.1 Source Code Packages

The source code of this demo includes the following packages:

- `com.ericsson.mpe.business.nbi.ext.demoHSS`: The `DemoHssNBImpl` class implements the HSS EPS Northbound Interface.
- `com.ericsson.mpe.business.nbi.ext.demoHSS.internal.sync`:
  - The `DemoHssImpl` class implements the HSS EPS provisioning interface. It is the entrance of processing SOAP requests.
  - The `DemoHSSOperation` class implements detailed processing logic of SOAP requests.
- `com.ericsson.mpe.business.nbi.ext.demoHSS.internal.sync.handler`:
  - `HeaderHandler` implements `SOAPHandler` to process the header message of SOAP requests and responses.
  - `ProclogHandler` implements `SOAPHandler` to restore the proclog of the header message of SOAP requests and responses.
- `com.ericsson.mpe.business.nbi.ext.demoHSS.util`:
  - `DefaultEntityResolverImpl` and `XMLUtil` include methods for handling the XML object.
  - `DemoHSSUtil` includes public static methods to handle universal logic.
  - `DemoHssConstants` defines static global variables.
  - `DemoHssMarshaler` transforms an object to an XML document.
  - `NamespaceList` includes some methods of creating the XML namespace.
  - `DefaultErrorHandlerImpl` is used by the XML parser to report any errors that occur during parsing.
  - `AlarmConstants` defines alarm code and message.

#### 4.2.1.2 OSGi Bundle Configuration

The file `serviceComponent_DemoHssNorthBoundServer.xml` defines bundle attributes, including bound name, implementation class, and interface. It is automatically generated by the Northbound Interface wizard.



### 4.2.1.3 Java Web Service Client Code

The Java web service client code is automatically created by `wsimport`. For detailed information about generating the code, see Section 4.2.3.1 on page 16.

### 4.2.1.4 Web Server Configuration Files

`web.xml`, `osgi.bnd`, and `sun-jaxws.xml` are the configuration files of Jetty HTTP server. The default accessing URL is `http://127.0.0.1:8080/demoHSS/demoHSS`.

**Note:** The first `demoHSS` in the above URL is denoted as `<Web-ContextPath>`, and the second `demoHSS` is denoted as `<url-pattern>`, which are both used in the following configuration steps.

The following is an example of customizing the accessing URL:

1. Configure `<url-pattern>` in the file `web.xml` as follows.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <listener>
    <listener-class>
      com.sun.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>demoHSS</servlet-name>
    <servlet-class>
      com.sun.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>demoHSS</servlet-name>
    <url-pattern>/demoHSS</url-pattern>
  </servlet-mapping>
</web-app>
```

2. Configure `<Web-ContextPath>` in the file `osgi.bnd` as follows.



```

#-----
# Use this file to add customized Bnd instructions for the bundle
#-----
#Bundle-Activator: ${bundle.namespace}.internal.Activator
Service-Component: OSGI-INF/*.xml

Web-ContextPath: /demoHSS
Export-Package:
Embed-Dependency: *;scope=compile|runtime
Embed-Transitive: true
Embed-Directory: WEB-INF/lib
Bundle-ClassPath: .,WEB-INF/classes
Import-Package: !javax.annotation;javax.servlet;javax.servlet.http;*

```

3. Configure `<url-pattern>` in the file `sun-jaxws.xml` as follows.

```

<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime" version="2.0">
  <endpoint name="demoHSS" implementation="com.ericsson.mpe.business.nbi.ext.demoHSS.internal.sync." />
</endpoints>

```

After the preceding configuration, the accessing URL has been set to `http://127.0.0.1:8080/demoHSS/demoHSS`.

#### 4.2.1.5

#### WSDL File and Jax Web Service Binding Files

The `demoHss.wsdl` file defines the demo HSS EPS web service interface. It is tailored HSS EPS WSDL in this demo.

The `jaxws_binding.xml` file defines how to generate JAX-WS web service client code according to the WSDL.

#### 4.2.2

#### Northbound Interface Implementation

Class `DemoHssNBImpl` implements the Northbound Interface and is managed with dependency injection. Dynamic Activation provides some basic services by the `NBContext` class, including alarm, authorization, configuration, log, provisioning, and configuration.

For information about how to get all the preceding services by `NBContext`, refer to the `start` method of `DemoHssNBImpl`, as shown in the following figure.

```
@Override
public void start(NBContext context) throws NBIException {
    setAlarmService(context.getAlarmService());
    setAuthenticationService(context.getAuthenticationService());
    setMBeanService(context.getMBeanService());
    setOSGiService(context.getExtensionService());
    setProcLogService(context.getProcLogService());
    setProvisionService(context.getProvisionService());
    setStatisticsService(context.getStatisticsService());
}
```

Figure 8 Start Method of DemoHssNBImpl

### 4.2.3 Web Services Implementation

In this demo, adapted SOAP WSDL is used to generate web service client code and implement the provisioning interface to handle the provisioning of SOAP requests to HSS EPS.

#### 4.2.3.1 Generating JAX-WS Web Service Client Code

To generate JAX-WS web service client code:

1. Configure Maven plug-in in `pom.xml`.

Use `jaxws` Maven plug-in to generate client code and define necessary attributes according to your project, for example:

- `wsdlDirectory`: The directory for the WSDL files
- `wsdlFile`: Adapted WSDL, for example, `demoHss.wsdl`.
- `bindingDirectory`: The directory `jaxws_binding.xml`
- `bindingFiles`: Use `jaxws_binding.xml` in demo. Defines how to generate the JAX-WS web service client code according to the WSDL.

The following figure shows an example of `jaxws` Maven plug-in in the `demoHSS` project:



```

<plugin>
<groupId>org.jvnet.jax-ws-commons</groupId>
<artifactId>jaxws-maven-plugin</artifactId>
<executions>
  <execution>
    <id>generate-sources</id>
    <goals>
      <goal>wsimport</goal>
    </goals>
    <configuration>
      <wsdlDirectory>${basedir}/src/wsdl</wsdlDirectory>
      <wsdlFiles>
        <wsdlFile>demoHss.wsdl</wsdlFile>
      </wsdlFiles>
      <bindingDirectory>${basedir}/src/wsdl</bindingDirectory>
      <bindingFiles>
        <bindingFile>jaxws_binding.xml</bindingFile>
      </bindingFiles>
      <xadditionalHeaders>true</xadditionalHeaders>
      <xnoAddressingDataBinding>true</xnoAddressingDataBinding>
    </configuration>
  </execution>
</executions>
</plugin>

```

## 2. Configure the jaxws\_binding.xml file.

Define how to generate the JAX-WS web service client code according to the WSDL with the Java tool wsimport:

- wsdlLocation: The path for the WSDL file. default jaxws\_binding.xml and WSDL file are saved in same directory.
- package: The package name of the JAX-WS web service client code
- enableWrapperStyle: The default value is true. Use non-wrapped binding styles to handle SOAP requests and responses.

The following figure shows the jaxws\_binding.xml file of the demoHSS project:

```

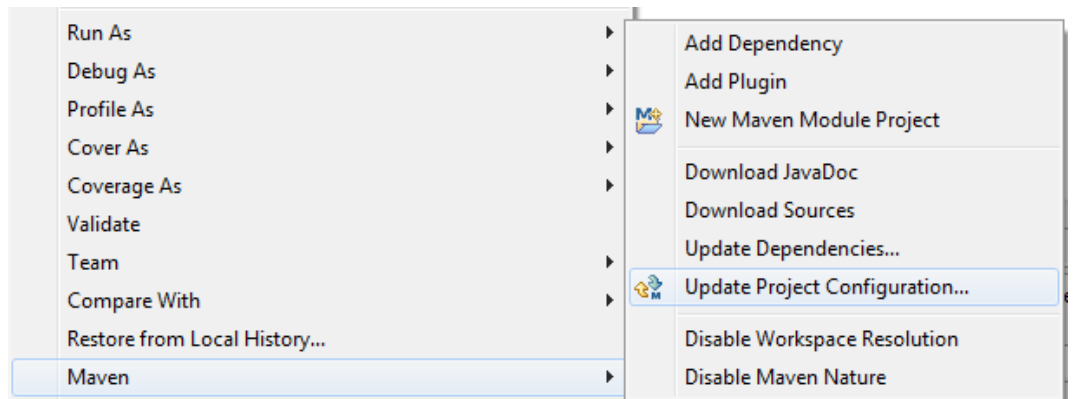
<bindings
  wsdlLocation="./demoHss.wsdl" xmlns="http://java.sun.com/xml/ns/jaxws"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb" xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <package name="com.ericsson.schemas.ma.hss.ext" />
  <enableWrapperStyle>false</enableWrapperStyle>
  <enableAsyncMapping>false</enableAsyncMapping>
  <bindings xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <jxb:schemaBindings>
      <jxb:package name="com.ericsson.schemas.ma.hss.ext" />
    </jxb:schemaBindings>
  </bindings>
</bindings>

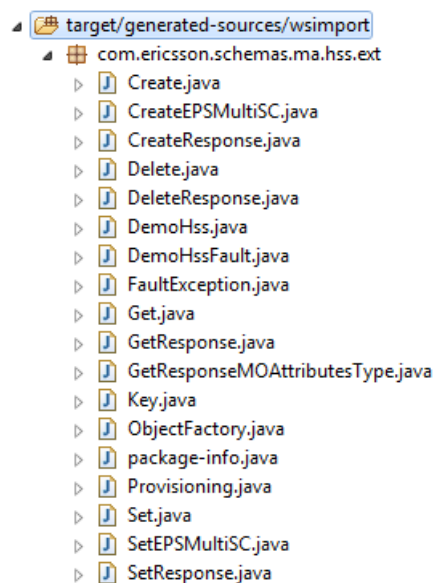
```

### 3. Generate web service client code.

After finishing editing the preceding configuration file, update Maven project in Dynamic Activation IDE by right-clicking the project and select **Maven > Update Project Configuration**, and wsimport is triggered to generate the web service client code.



Generate the source code by right-clicking the **project** and selecting **Run As > Maven generate-sources** in Dynamic Activation IDE. The following figure shows an example of Java web service client code.



### 4. Configure Jetty server.

Embedded Jetty of OSGi is used in this demo as HTTP server. The demo SOAP request and standard CAI3G request are both sent to the same Jetty server. Configure `web.xml` with correct servlet mapping, and define the `serverlet-name` and `url-pattern`.

The following figure shows an example `web.xml` file:



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                      http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <listener>
    <listener-class>
      com.sun.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>

  <servlet>
    <servlet-name>demoHSS</servlet-name>
    <servlet-class>
      com.sun.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>demoHSS</servlet-name>
    <url-pattern>/demoHSS</url-pattern>
  </servlet-mapping>

</web-app>
```

#### 4.2.3.2 Generating Web Service Provisioning Code

To generate web service provisioning code, follow the subsections.

##### 4.2.3.2.1 Creating a Class to Handle the SOAP Request Header

Create class `HeaderHandler` and class `ProclogHandler` in package `com.ericsson.mpe.business.nbi.ext.demoHSS.internal.sync.handler`. These classes implement the interface `SOAPHandler<SOAPMessageContext>`. The SOAP message handler provides a mechanism for intercepting the SOAP message in both the request and response of the web service.

In the `HeaderHandler` class, it is assumed that username and password are transported by the security header of an SOAP request. Each SOAP request triggers a login and logout operation. The `handleWSSecurity` method gets username and password from the security header of the SOAP request and saves them in the `HttpSession` object.

The `handleMessage` method handles the request and response.

In the `ProclogHandler` class, it saves the SOAP message of requests in `HttpSession` and the SOAP message of responses in the processing log.

Create `HSShandlerchain.xml` to define which class is used to handle the header of SOAP messages. In this demo project, the `HeaderHandler` class



is used to handle the header of SOAP messages, and the `ProclogHandler` class is used to handle the proclog of SOAP messages.

```
<?xml version="1.0" encoding="UTF-8"?>
<jws:handler-chains xmlns:jws="http://java.sun.com/xml/ns/javaee">
  <jws:handler-chain name="DemoHssHandlerChain">
    <handler xmlns="http://java.sun.com/xml/ns/javaee">
      <handler-name>HeaderHandler</handler-name>
      <handler-class>com.ericsson.mpe.business.nbi.ext.demoHSS.internal.sync.handler.HeaderHandler</handler-class>
    </handler>
    <jws:handler xmlns="http://java.sun.com/xml/ns/javaee">
      <jws:handler-name>ProclogHandler</jws:handler-name>
      <jws:handler-class>com.ericsson.mpe.business.nbi.ext.demoHSS.internal.sync.handler.ProclogHandler</jws:handler-class>
    </jws:handler>
  </jws:handler-chain>
</jws:handler-chains>
```

Figure 9 *HSShandlerchain.xml File*

#### 4.2.3.2.2 Creating a Class to Implement the HSS EPS Provisioning Interface

Create the `DemoHssImpl` class to implement the `com.ericsson.schemas.ma.hss.ext.Provisioning` interface and add necessary annotations:

- `@XmlSeeAlso`: Instructs JAXB to bind other classes when binding the `DemoHssImpl` class.
- `@WebService`: Marks a Java class when implementing a web service. It includes the following attributes:

<code>name</code>	The class name for implementing a web service
<code>targetNamespace</code>	WSDL namespace
<code>serviceName</code>	Web service name, which must be as same as the endpoint name of <code>sun-jaxws.xml</code>
<code>endpointInterface</code>	The name of WSDL provisioning interface

- `@HandlerChain`: Indicates the configuration file of handler chains of the web service.



```

@XmlSeeAlso({ com.ericsson.schemas.ma.hss.ext.ObjectFactory.class })
@WebService(name = "DemoHssImpl", targetNamespace = "http://schemas.ericsson.com/ma/HSS/ext/",
    serviceName = "demoHSS", endpointInterface = "com.ericsson.schemas.ma.hss.ext.Provisioning")
@HandlerChain(file = "handler/HSSHandlerchain.xml")
public class DemoHssImpl implements Provisioning {

    private final static Logger logger = LoggerFactory.getLogger(DemoHssImpl.class);
    @Resource
    private WebServiceContext webServiceContext;

    @Override
    public CreateResponse create(Create create) throws FaultException {
        logger.debug("Create request received.");

        DemoHSSOperation hssOperation = initHSSOperation();
        return hssOperation.getCreateResponse(create);
    }
}

```

Implement four methods to handle SOAP Create, Delete, Get, and Set operations. Each method receives a corresponding request object that contains all WSDL request parameters. Each method returns a corresponding response object that contains all WSDL response parameters. Each method throws out `FaultException`.

The following is an implementation example of the Create method:

```

@Override
public CreateResponse create(Create create) throws FaultException {
    logger.debug("Create request received.");

    DemoHSSOperation hssOperation = initHSSOperation();
    return hssOperation.getCreateResponse(create);
}

```

This example receives the Create object as a parameter and returns the CreateResponse object. The Create and CreateResponse objects are generated according to the demoHss.wsdl file. For details on the implementation, refer to the java code of Create and CreateResponse methods in the `com.ericsson.schemas.ma.hss.ext` package.

#### 4.2.3.2.3 Processing Logic

The processing logic is implemented in the `HSSOperation` class. It implements all necessary request processing logic, including login, logout, SOAP request mapping with internal data mode and recording the ProcLog function.

The main processing logic includes:

- Login: Gets username and password from the `HttpSession`. Get an authentication service with the `DemoHssNBImpl.getAuthenticationService` method and execute the login method.

The Login method returns the `sessionID` and saves it for feature use. Create a ProcLog entry with the `ProcLogEntry.createProcLogEntry`



method, and save successful or failed end point of ProcLog with the `procLogEntry.succeed` or `procLogEntry.fail` method.

- **Alarm:** The Northbound Interface can get an alarm service from `NBContext`. Dynamic Activation provides an alarm code range 7701–7899 and specific `resourceId` .1.1.1.1.6.5 for northbound adapter. Customers can define their own alarm message according to business requirement. For details on alarm definition, refer to class `AlarmConstants`.

Raise an alarm if login fails in this demo. For example:

```
catch (NBException e) {
    procLogEntry.fail(String.valueOf(e.getErrorCode()));
    if (isLogEnabled()) {
        logPoint(procLogEntry, getUsername(), ProcLogEntry.OPERATION_LOGIN, "", "", true);
    }
    try {
        DemoHssNBImpl.getAlarmService().sendAlarm(AlarmConstants.LOGIN_FAILURE, AlarmConstants.LOGIN_FAILURE_MESSAGE);
    } catch (NBException mpe) {
        LOGGER.warn("Failed to send event " + AlarmConstants.LOGIN_FAILURE_MESSAGE, mpe);
        throw mpe;
    }
    throw e;
}
```

- **Logout:** Get an authentication service by `DemoHssNBImpl.getAuthenticationService` and execute the logout process.
- **Execute command:** The main processing logic is to map a SOAP request to an internal Northbound Interface request. Then refer to the methods `getCreateResponse`, `getSetResponse`, `getQueryResponse`, and `getDeleteResponse` methods. To build a Northbound Interface request, assign the following attributes:
  - **SessionData:** A set of session-related data, such as username, proclog ID, and session ID. `SessionData` carries the session-specific data, which is used by the Dynamic Activation framework. For details on its implementation, refer to the `buildSessionData` method.
  - **Operation:** Assign operation attributes according to the SOAP request type (Create, Get, Set, and Delete).
  - **MO\_NAME:** Assign value according to the WSDL definition. `MO_NAME` is the logical name of the target MO. In CAI3G protocol, The value of `MO_NAME` is as same as the one of MO type. A tailored HSS EPS WSDL is used in this demo, and HSS EPS only receives requests if it fulfills the CAI3G. Therefore, assign `EPSMultiSC@http://schemas.ericsson.com/ma/HSS/` to `MO_NAME`. Customers can set their own `MO_NAME` only if NE can parse it. `MO_NAME` is used to configure the routing on Dynamic Activation GUI.

For example:

The `MO_NAME` of CAI protocol: `AUCSUB`



The `MO_NAME` of CAI3G protocol: `EPSPMultiSC@http://schemas.ericsson.com/ma/HSS/`

- `RawRequest`: The original request sent from the Northbound Interface. Get SOAP raw requests from the `HttpSession` object.
- `MoAttributes`: It is an XML document that contains all required information for the BL. The `MoAttributes` is used to transfer data from the Northbound Interface to proxy, and it is transparent to the current Dynamic Activation service, because it contains BL-related information.

In this example, a tailored HSS EPS WSDL and provisioning HSS EPS are used, so `MoAttributes` must follow the CAI3G rule in this demo. Generate Payload to the CAI3G `MoAttribute` object. For example, add the `MoAttribute` tag, change the namespace from `http://schemas.ericsson.com/ma/HSS/ext/` to `http://schemas.ericsson.com/ma/HSS/` to fulfill the requirement of CAI3G requests. Refer to the `buildRequestMoAttributes` method of the `DemoHSSUtil` class.

An example of `MoAttributes` of CAI3G protocol is shown as follows:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<MoAttributes>
  <MoAttributes>
    <CreateEPSPMultiSC imsi="494500100000"
      xmlns="http://schemas.ericsson.com/ma/HSS/">
      <imsi>494500100000</imsi>
      <epsProfileId>Profile900C</epsProfileId>
      <epsRoamingAllowed>true</epsRoamingAllowed>
    </CreateEPSPMultiSC>
  </MoAttributes>
  <Extension/>
  <NAMESPACE>
    default-ns-c9e9111b-fa9e-454a-b691-4578c8e14c27=
    http://schemas.ericsson.com/ma/HSS/ext/
  </NAMESPACE>
</MoAttributes>
```

- Parse the Dynamic Activation response: Parse the Northbound Interface response and map its attribute to corresponding SOAP response attributes. For details, refer to the `buildCreateResponse`, `buildDeleteResponse`, and `buildGetResponse` methods.

## 4.2.4 Using Statistics Service

Dynamic Activation can provide statistics service to monitor provisioning performance. The performance can be shown in the **Dashboard** of the Dynamic Activation GUI.

This section describes the relevant codes in the demo project, which can be used as examples to develop your own NBIA.



#### 4.2.4.1 Providing Statistics Service

To provide the statistics service, the following Maven dependency is added to the `pom.xml` of the demo project:

```
<dependency>
  <groupId>com.ericsson.mpe</groupId>
  <artifactId>mpe-platform-common</artifactId>
  <scope>provided</scope>
</dependency>
```

#### 4.2.4.2 Getting Statistics Service

To get the statistics service, the `DemoHssNBImpl` class includes the following codes:

1. A variable is added in the class.

```
private static StatisticsService statisticsService = null;
```

2. The `start` method of the class includes the following line.

```
setStatisticsService(context.getStatisticService());
```

Example:

```
@Override
public void start(NBIContext context) throws NBIException {
  setAlarmService(context.getAlarmService());
  setAuthenticationService(context.getAuthenticationService());
  setMBeanService(context.getMBeanService());
  setOSGiService(context.getExtensionService());
  setProcLogService(context.getProcLogService());
  setProvisionService(context.getProvisionService());
  setStatisticsService(context.getStatisticsService());
}
```

3. Corresponding Setter and Getter methods for the added lines are generated in the class.

#### 4.2.4.3 Generating Reports

In the `DemoHSSOperation` class, the `reportStatistics` method is defined as shown in Figure 10.



```
private void reportStatistics(ProcLogEntry procLogEntry) {  
    String status = procLogEntry.getResponseCode();  
    String hostname = procLogEntry.getHostname();  
    String moType = procLogEntry.getTarget();  
    String operation = procLogEntry.getOperation();  
    String userName = procLogEntry.getUser();  
    long executionTime = procLogEntry.getStopTime() - procLogEntry.getStartTime();  
    if (DemoHssNBImpl.getStatisticsService() != null) {  
        DemoHssNBImpl.getStatisticsService().reportCSOResponse(status, hostname, moType, operation, userName, executionTime);  
    }  
}
```

Figure 10 *reportStatistics Method*

To generate statistics report, call the `reportStatistics` method from the desired place where reports are generated, normally the same place where the `proclog` is reported.

Figure 11 shows a code example from the `DemoHSSOperation` class.



```
public SetResponse getSetResponse(Set set) throws FaultException {
    LOGGER.debug("HSS: Set request");
    ProcLogEntry procLogEntry = ProcLogEntry.createNorthBoundPr
    try {
        // Login successful
        loginPM(getUserName(), getPassword());
        // execute the command
        Response response = execute(set, userSessionID, procLog
        SetResponse setResponse = DemoHSSUtil.buildSetResponse(
        if (isLogEnabled()) {
            logPoint(procLogEntry, getUserName(), ProcLogEntry.
        }
        procLogEntry.succeed();
        reportStatistics(procLogEntry);
        // Logout
        logoutPM();
        return setResponse;
    } catch (NBIException e) {
        FaultException soapFault = DemoHSSUtil.createSOAPFault(
        procLogEntry.fail(String.valueOf(e.getErrorCode()));
        reportStatistics(procLogEntry);
        if (isLogEnabled()) {
            logPoint(procLogEntry, getUserName(), ProcLogEntry.
        }
        try{
            logoutPM();
        }
        catch(NBIException e1){
            LOGGER.error(MESSAGE_FAILED_LOGOUT, e1.getErrorCode
        }
        throw soapFault;
    }
}
```

Figure 11 Example of Calling reportStatistics Method

#### 4.2.5 Packaging Northbound Adapter

Use Maven to make the tar.gz package of NBIA. Confirm the Maven configuration in the `assembly.xml` file, which is like the following one:



```
<assembly
  xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.2"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/xsd/assembly-1.1.2.xsd">
  <id>sources</id>
  <includeBaseDirectory>false</includeBaseDirectory>
  <formats>
    <format>tar.gz</format>
  </formats>
```

Figure 12 assembly.xml File

Build the `tar.gz` package by right-clicking the file and selecting **Run As > Maven install** in Dynamic Activation IDE.

## 5 Deployment

After completing the development of NBIA, a `tar.gz` package is ready for deployment. Place the `tar.gz` file under the directory `/home/bootloader/repository`. And run the following command to change the owner and the group of the submodule `tar.gz` file.

```
# chown actadm:activation /home/bootloader/repository/<submodule tar file>
```

### 5.1 Administrating the Northbound Interface Adapter

Perform the steps in this section on both PL nodes.

#### 5.1.1 Installing the Northbound Interface Adapter

Run the following command to install an NBIA:

```
# bootloader.py submodule add --name/-n <submodule tar file>
--type/-t nbi --parent/-p nbia-module --host <PL hostname>
```

**Note:** `<PL hostname>` is the hostname of PL-3 and PL-4.

For example:

```
# bootloader.py submodule add -n mpe-business-nbi-demoHSS-
15.0.0.tar.gz -t nbi -p nbia-module --host EBS-PL-3
```



### 5.1.2 Activate the Northbound Interface Adapter

Run the following command to activate the NBIA:

```
# bootloader.py node activate --host <PL hostname>
```

For example:

```
# bootloader.py node activate --host EBS-PL-3
```

### 5.1.3 Checking the Submodule Status

Run the following command to check the submodule version and status of the NBIA:

```
# bootloader.py node status --host <hostname>
```

For example:

```
# bootloader.py node status --host all
```

The status of `nbia-module` must be Running as follows.

Host	Module	Version	Status	Bindings
=====	=====	=====	=====	=====
node1	nbia-module	1.3.2	Running	OK(2/2)
node1	-> mpe-business-nbi-demoHSS	15.0.0		

### 5.1.4 Uninstalling the Northbound Interface Adapter

Run the following command to uninstall the NBIA:

```
# bootloader.py submodule delete --name/-n \  
<submodule name> --parent/-p nbia-module --host <PL  
hostname>
```

For example:

```
# bootloader.py submodule delete -n \  
mpe-business-nbi-demoHSS -p nbia-module --host EBS4-PL-3
```

After uninstalling the submodule, the submodule must be removed from the module list. For how to check module status, refer to Section 5.1.3 on page 28.

### 5.1.5 Updating the Northbound Interface Adapter

Follow these steps to update an NBIA:

1. From an SC node, run the following command to update the NBIA:



```
# bootloader.py submodule update --name/-n \
<submodule tar file> --type/-t \
nbi --parent/-p nbia-module --host <PL hostname>
```

*<PL hostname>* is the hostname of the PL node to which the NBIA is updated.

For example, update demoHSS NBIA on PL-3:

```
# bootloader.py submodule update -n \
mpe-business-nbi-demoHSS-15.0.0.tar.gz -t nbi \
-p nbia-module --host EBS-PL-3
```

2. From an SC node, run the following command to activate the PL node:

```
#bootloader.py node activate --host <PL hostname>
```

*<PL hostname>* is the hostname of the PL node to which the NBIA is updated.

For example:

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

## 5.2 Setting Properties for the Northbound Interface Adapter

After creating a project in Dynamic Activation IDE, two configuration files (a .cfg file and a .conf file) are generated automatically for setting the user properties.

- .cfg - This file contains a list of all the configurable parameters.
- .conf - This file contains the setting of the default values of the parameters which are loaded by bootloader.

The following project DemoHSS is used as an example in this section to describe the process of setting properties.

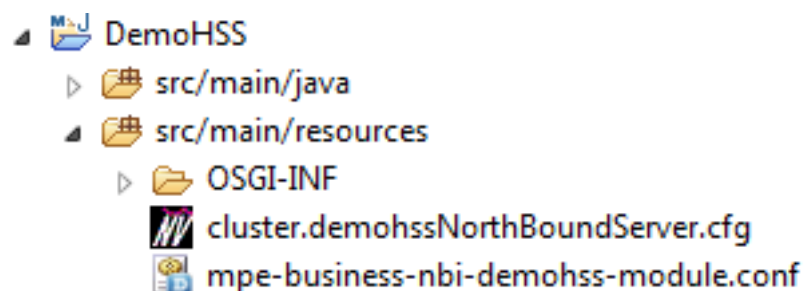


Figure 13 DemoHSS Project



### 5.2.1 Creating Properties for the Cluster Northbound Interface Adapter

The properties are created by editing the two configuration files as the following steps:

1. Create the property names in `Cluster.DemoHssNorthBoundServer.cfg` as the following example.

```
param1=@PARAM1@  
param2=@PARAM2@
```

2. Define the property values in `mpe-business-nbi-demoHSS-module.conf`.

Edit the configuration block in `mpe-business-nbi-demoHSS-module.conf` as follows.

```
cluster.DemoHssNorthBoundServer.cfg  
  
param1=@PARAM1@  
param2=@PARAM2@  
  
mpe-business-nbi-demoHSS-module.conf  
{  
  "configuration": [  
    {  
      "file": "/cluster.demohssDemoHssNorthBoundServer.cfg"  
      "parameter": "@PARAM1@"  
      "defaultvalue": "test_param1" — define a new value  
    },  
    {  
      "file": "/cluster.demohssDemoHssNorthBoundServer.cfg",  
      "parameter": "@PARAM2@",  
      "defaultvalue": "test_param2"  
    }  
  ]  
}
```

**"file"** - name of the configuration file that contains this property name

**"parameter"** - name of the property

**"defaultvalue"** - default value of the property

### 5.2.2 Modifying Properties for the Cluster Northbound Interface Adapter

1. Run the following command to modify the value of a property:

```
# bootloader.py config set --parameter \  
<parameter> --value <value>
```



For example:

```
# bootloader.py config set --parameter \
@PARAM1@ --value true
```

2. Run the following command to activate the configuration on PL nodes.

```
# bootloader node activate --host <hostname>
```

### 5.2.3

## Restore the Default Value of the Property for the Cluster Northbound Interface Adapter

**Note:** The default values are defined in the `.conf` file.

1. Run the following command to restore the default value of a property:

```
# bootloader.py config remove \
--parameter <parameter>
```

For example:

```
# bootloader.py config remove \
--parameter @PARAM1@
```

2. Run the following command to activate the configuration on PL nodes.

```
# bootloader node activate --host <hostname>
```

## 5.3

## Configuring ESA Alarms

### 5.3.1

## Adding the Alarm Definition

Add the alarm definition in the file `/home/dveadm/nodes/<nodeid>/ESA/conf/fmAlarmDefinitions/nbia-module-alarmsDef.xml` on each Payload node. Alarm definition must be consistent with the definition in project source code.

An example of the alarm definition is as follows:



```

<alarmSpecification active="yes">
  <moduleId>NBI</moduleId>
  <errorCode>7701</errorCode>
  <severity>4</severity>
  <modelDescription>Failed to create subscription.</modelDescription>
  <activeDescription>Failed to create subscription.</activeDescription>
  <eventType>2</eventType>
  <probableCause>600</probableCause>
  <documentation>
    <description>Failed to create subscription.</description>
    <alarmingObject>NBI</alarmingObject>
    <raisedBy>NBI create</raisedBy>
    <clearedBy>No clear, stateless alarm.</clearedBy>
    <proposedRepairAction>Check the northbound request.</proposedRepairAction>
  </documentation>
</alarmSpecification>

```

### Example 1 Alarm Definition

For more information about ESA alarms, refer to Reference [5].

## 5.3.2 Configuring Alarm Resource ID

The resource ID .1.1.1.1.6.5 is provided for NBIsAs.

Add the alarm resource ID definition in the file /opt/dve-repository/LOT C-CONFIG/esa/PG.xml on each Payload node.

An example of alarm resource ID definition is as follows.

Append an alarm resource ID .1.1.1.1.6.5 to SNOSNE cimname="Advanced Functionality" for PG1 node, which is shown as follows:

```

<SNOSNE cimname="Advanced Functionality" snoshostip="PG1_IP_ADDRESS"
  snoshostname="" snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6"
  snosnmppcommunity="SNOS-PE" snosnmppport="161">
  <SNOSNE cimname="NBI" snoshostip="PG1_IP_ADDRESS" snoshostname=""
    snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.1"
    snosnmppcommunity="SNOS-PE" snosnmppport="161" />
  <SNOSNE cimname="Cassandra" snoshostip="PG1_IP_ADDRESS" snoshostname=""
    snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.3"
    snosnmppcommunity="SNOS-PE" snosnmppport="161" />
  <SNOSNE cimname="ARH" snoshostip="PG1_IP_ADDRESS" snoshostname=""
    snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.4"
    snosnmppcommunity="SNOS-PE" snosnmppport="161" />
  <SNOSNE cimname="Replay Service" snoshostip="PG1_IP_ADDRESS" snoshostname=""
    snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.6"
    snosnmppcommunity="SNOS-PE" snosnmppport="161" />
  <SNOSNE cimname="NBI" snoshostip="PG1_IP_ADDRESS" snoshostname=""
    snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.5"
    snosnmppcommunity="SNOS-PE" snosnmppport="161" />
</SNOSNE>
<

```

### Example 2 Appending .1.1.1.1.6.5 for PG1 Node

Append an alarm resource ID .1.1.1.1.6.5 to SNOSNE cimname="Advanced Functionality" for PG2 node, which is shown as follows:



```
<SNOSNE cimname="Advanced Functionality" snoshostip="PG2_IP_ADDRESS"
snoshostname="" snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6"
snosnmpcommunity="SNOS-PE" snosnmpport="161">
<SNOSNE cimname="NBIA" snoshostip="PG2_IP_ADDRESS" snoshostname=""
snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.1"
snosnmpcommunity="SNOS-PE" snosnmpport="161" />
<SNOSNE cimname="Cassandra" snoshostip="PG2_IP_ADDRESS" snoshostname=""
snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.3"
snosnmpcommunity="SNOS-PE" snosnmpport="161" />
<SNOSNE cimname="ARH" snoshostip="PG2_IP_ADDRESS" snoshostname=""
snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.4"
snosnmpcommunity="SNOS-PE" snosnmpport="161" />
<SNOSNE cimname="Replay Service" snoshostip="PG2_IP_ADDRESS" snoshostname=""
snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.6"
snosnmpcommunity="SNOS-PE" snosnmpport="161" />
<SNOSNE cimname="NBI" snoshostip="PG2_IP_ADDRESS" snoshostname=""
snosversion="R1" cimotheridentifyinginfo=".1.1.1.1.6.5"
snosnmpcommunity="SNOS-PE" snosnmpport="161" />
</SNOSNE>
```

### Example 3 Appending .1.1.1.1.6.5 for PG2 Node

## 5.3.3 Raising an Alarm

Raise an alarm if login fails in this demo. For example:

```
catch (NBException e) {
    procLogEntry.fail(String.valueOf(e.getErrorCode()));
    if (isLogEnabled()) {
        logPoint(procLogEntry, getUsername(), ProcLogEntry.OPERATION_LOGIN, "", "", true);
    }
}
try {
    DemoHssNBImpl.getAlarmService().sendAlarm(AlarmConstants.LOGIN_FAILURE, AlarmConstants.LOGIN_FAILURE_MESSAGE)
} catch (NBException mpe) {
    LOGGER.warn("Failed to send event " + AlarmConstants.LOGIN_FAILURE_MESSAGE, mpe);
    throw mpe;
}
throw e;
```

Figure 14 Code for Raising an Alarm

## 5.4 Configuring Logs

The following OSGi log files are available on Payload nodes:

/var/log/dve/nbia.log

Configure the log level manually by editing the logback configuration file:

1. Open the configuration file “/usr/local/pgngn/nbia-module-\*/etc/org.ops4j.pax.logging.cfg” with a text editor.
2. Add logger definition for Northbound Interface.

An example of logger definition is as follows:

```
log4j.logger.com.ericsson.mpe.business.nbi.ext.demoHS
S=INFO
```



3. Save the file. It takes some seconds before the changes take effect.
4. Repeat Step 1 to Step 3 for all Payload nodes in the system.

## 5.5 Configuring Security

The default acceptable list of ciphers for HTTPS connections are:

- AES-CTR 128 / 256 bits
- AES-CBR 128 / 256 bits
- RC4 128 / 256 bits (also called arcfour)

To further strengthen the security, it is possible to change the list of ciphers with the parameter `EXTERNAL_SSL_CIPHERS` as in Section 5.2.2 on page 30.

## 6 Verifying the Northbound Adapter

The adapted SOAP WSDL file can be imported into SOAP tools such as SOAPUI to generate requests for the verification of Northbound Interface.

### 6.1 Creating a Subscription

The `Create` operation creates an `EPSCMultiSC` in HSS EPS.

An example of creating `EPSCMultiSC` requests and responses is shown as follows.



```

Request:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ext="http://schemas.ericsson.com/ma/HSS/ext/">
  <soapenv:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:UsernameToken
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsse:Username>sogadm</wsse:Username>
            <wsse:Password>sogadm</wsse:Password>
          </wsse:UsernameToken>
        </wsse:Security>
      </soapenv:Header>
    <soapenv:Body>
      <ext:Create>
        <ext:key>
          <ext:imsi>12345678</ext:imsi>
        </ext:key>
        <ext:Payload>
          <ext:CreateEPSMultiSC imsi="12345678">
            <ext:imsi>12345678</ext:imsi>
            <ext:epsProfileId>1</ext:epsProfileId>
            <!--Optional:-->
            <ext:epsRoamingAllowed>true</ext:epsRoamingAllowed>
          </ext:CreateEPSMultiSC>
        </ext:Payload>
      </ext:Create>
    </soapenv:Body>
  </soapenv:Envelope>

Response:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <CreateResponse xmlns="http://schemas.ericsson.com/ma/HSS/ext/">
      <response>0</response>
    </CreateResponse>
  </S:Body>
</S:Envelope>

```

#### Example 4 Creating a Subscription

## 6.2 Setting a Subscription

The Set operation modifies an EPSMultiSC in HSS EPS

An example of setting EPSMultiSC requests and responses is shown as follows.



```

Request:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ext="http://schemas.ericsson.com/ma/HSS/ext/">
  <soapenv:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:UsernameToken
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsse:Username>sogadm</wsse:Username>
            <wsse:Password>sogadm</wsse:Password>
          </wsse:UsernameToken>
        </wsse:Security>
      </soapenv:Header>
      <soapenv:Body>
        <ext:Set>
          <ext:key>
            <ext:imsi>12345678</ext:imsi>
          </ext:key>
          <ext:Payload>
            <ext:SetEPSMultiSC imsi="12345678">
              <!--Optional:-->
              <ext:epsProfileId>2</ext:epsProfileId>
              <!--Optional:-->
              <ext:epsRoamingAllowed>>false</ext:epsRoamingAllowed>
            </ext:SetEPSMultiSC>
          </ext:Payload>
        </ext:Set>
      </soapenv:Body>
    </soapenv:Envelope>

Response:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <SetResponse xmlns="http://schemas.ericsson.com/ma/HSS/ext/">
      <response>0</response>
    </SetResponse>
  </S:Body>
</S:Envelope>

```

### Example 5 Setting a Subscription

## 6.3 Getting a Subscription

The `Get` operation fetches the data of the `EPSMultiSC` from HSS EPS.

An example of getting `EPSMultiSC` requests and responses is shown as follows.



```

Request:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ext="http://schemas.ericsson.com/ma/HSS/ext/">
  <soapenv:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:UsernameToken
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsse:Username>sogadm</wsse:Username>
            <wsse:Password>sogadm</wsse:Password>
          </wsse:UsernameToken>
        </wsse:Security>
      </soapenv:Header>
    <soapenv:Body>
      <ext:Get>
        <ext:key>
          <ext:imsi>12345678</ext:imsi>
        </ext:key>
      </ext:Get>
    </soapenv:Body>
  </soapenv:Envelope>

Response:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <GetResponse xmlns="http://schemas.ericsson.com/ma/HSS/ext/">
      <GetResponseMOAttributesType>
        <ns:GetResponseEPSMultiISC imsi="12345678" xmlns:ns="http://schemas.ericsson.com/ma/HSS/">
          <ns:imsi>12345678</ns:imsi>
          <ns:epsProfileId>2</ns:epsProfileId>
          <ns:epsOdb>NONE</ns:epsOdb>
          <ns:epsRoamingAllowed>false</ns:epsRoamingAllowed>
          <ns:epsLocationState>UNKNOWN</ns:epsLocationState>
        </ns:GetResponseEPSMultiISC>
      </GetResponseMOAttributesType>
    </GetResponse>
  </S:Body>
</S:Envelope>

```

### Example 6 Getting a Subscription

## 6.4 Deleting a Subscription

The Delete operation removes an EPSMultiISC from HSS EPS.

An example of deleting EPSMultiISC requests and responses is shown as follows.



```

Request:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:ext="http://schemas.ericsson.com/ma/HSS/ext/">
  <soapenv:Header>
    <wsse:Security
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
        <wsse:UsernameToken
          xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">
            <wsse:Username>sogadm</wsse:Username>
            <wsse:Password>sogadm</wsse:Password>
          </wsse:UsernameToken>
        </wsse:Security>
      </soapenv:Header>
    <soapenv:Body>
      <ext:Delete>
        <ext:key>
          <ext:imsi>12345678</ext:imsi>
        </ext:key>
      </ext:Delete>
    </soapenv:Body>
  </soapenv:Envelope>

Response:
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <DeleteResponse xmlns="http://schemas.ericsson.com/ma/HSS/ext/">
      <response>0</response>
    </DeleteResponse>
  </S:Body>
</S:Envelope>

```

### Example 7 Deleting a Subscription

## 6.5 Faults or Errors

An error response example for HSS EPS provisioning is shown as follows.

```

<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:Fault xmlns:ns2="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:ns3="http://www.w3.org/2003/05/soap-envelope">
      <faultcode>ns2:Server</faultcode>
      <faultstring>External error.</faultstring>
      <detail>
        <demoHssFault:demoHssFault
          xmlns="http://schemas.ericsson.com/ma/HSS/ext/"
          xmlns:demoHssFault="http://schemas.ericsson.com/ma/HSS/ext/">
          <faultcode>1101</faultcode>
          <faultreason>
            <reasonText>SERVICE NOT DEFINED</reasonText>
          </faultreason>
        </demoHssFault:demoHssFault>
      </detail>
    </ns2:Fault>
  </S:Body>
</S:Envelope>

```

### Example 8 Error Response



## Reference List

- [1] *Library Overview*, 18/1553-CSH 109 628 Uen
- [2] *Customization - Architectural Overview*, 20/1553-CSH 109 628 Uen
- [3] *Northbound Interface Adapter Reference Manual*, 1/2134-CSH 109 628 Uen
- [4] *Customer Adaptation Development Guide for Resource Activation*, 5/1553-CSH 109 628 Uen
- [5] *ESA Fault Management*, 2/1543-FAM 901 455 Uen