SAP NetWeaver® Identity Management
Identity Center

Implementation guide
- Extension Framework

Version 7.2 Rev 4
Preface

The product

SAP NetWeaver Identity Management Identity Center is a high-end identity management solution, capable of handling a large amount of repositories containing an unlimited amount of information. The Identity Center offers a robust, flexible and scalable high-availability solution for workflow, provisioning, data synchronization and joining for a large number of data repositories.

The reader

This manual is written for people who are to extend an Identity Center configuration with 3rd party Java classes, like input validation, with the Extension Framework.

Prerequisites

To get the most benefit from this manual, you should have the following knowledge:

- Knowledge of the Identity Center.
- Basic knowledge about the SAP NetWeaver AS Java and its tools.
- Thorough knowledge about the SAP NetWeaver Developer Studio.

The following software is required:

- SAP NetWeaver Identity Management Identity Center 7.2 SP3 or newer is correctly installed and licensed.
- One of the following SAP NetWeaver versions must be correctly installed and licensed:
  - SAP NetWeaver AS Java as of Release 7.0 SP14 or newer.
  - Enhancement Package 1 for SAP NetWeaver Composition Environment 7.1.
  - SAP NetWeaver Composition Environment 7.2.
  - SAP NetWeaver 7.3.
  - SAP NetWeaver Developer Studio 7.0 or 7.1.

The manual

This document describes how the Java class is implemented as a project in the SAP NetWeaver Developer Studio, how the class is deployed on the SAP NetWeaver running the User Interface. It also describes how to include the reference to the class from the task configuration in the Management Console.

This tutorial is not a substitution for training.
Related documents

You can find useful information in the following documents:

- SAP NetWeaver Identity Management Identity Center tutorials and help file.
- Relevant documentation for SAP NetWeaver AS Java and its tools.
# Table of contents

Introduction .......................................................................................................................................... 1  
Section 1: Creating the project .............................................................................................................. 3  
Section 2: Implementing and deploying a TaskProcessing class........................................................... 19  
Section 3: Task configuration and class reference................................................................................ 34
Introduction

The Identity Center Extension Framework provides a way to extend an Identity Center configuration with the features from a separately developed Java class, and it is utilized when executing a task in the Identity Management User Interface. The extension class may be configured for any User Interface task in the Identity Center. It allows customers to write their own code for validating attributes entered by the user, using the SAP NetWeaver Identity Management User Interface. The framework allows handling of single-value and multivalue attributes except for attributes of type Entry Reference. For example, when the user selects "Submit" on the task, the built-in validation is performed first. Then the user code is called and can perform additional validation. This code may return "OK", or return an error message, which is presented to the user. Only when the user code returns "OK" the data is stored in the identity store.

In the current implementation, there are two possible extension points: "onLoad" and "onSubmit", corresponding to extension class methods `onLoad` and `onSubmit`.

The `onLoad` method is executed prior to displaying the entry in the User Interface. It provides a way to add dynamic default values for the entries in a given task. The `onSubmit` method is executed, just before the task itself is executed. It provides a way to implement input attribute validation for a given task, as well as modifying of the input values before storing them.

The implementation process looks like this:

1. The Java class is implemented as a project in the SAP NetWeaver Developer Studio.
2. Deploy the extension class on the SAP NetWeaver running the User Interface.
3. Include the reference to the class and the methods from the task configuration in the Management Console. Here you can also supply the additional custom parameters into the extension framework.

Combination of versions

The Identity Management User Interface can be deployed on the following versions of SAP NetWeaver:

- SAP NetWeaver AS Java as of Release 7.0
- EHP 1 for SAP NetWeaver CE 7.1
- SAP NetWeaver CE 7.2
- SAP NetWeaver 7.3

The following versions of SAP NetWeaver Developer Studio are supported:

- SAP NetWeaver Developer Studio 7.0
- SAP NetWeaver Developer Studio 7.1

The following combinations are legal:

<table>
<thead>
<tr>
<th></th>
<th>Developer Studio 7.0</th>
<th>Developer Studio 7.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP NW 7.0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>EHP 1 for SAP NW CE 7.1</td>
<td>Not supported</td>
<td>X</td>
</tr>
<tr>
<td>SAP NW CE 7.2</td>
<td>Not supported</td>
<td>X</td>
</tr>
<tr>
<td>SAP NW 7.3</td>
<td>Not supported</td>
<td>X</td>
</tr>
</tbody>
</table>
In case the procedure is different for the different combinations, there are separate sections for each of them.

In the document, the procedures are described for all legal combinations as shown in the table above.

**Use cases**

In this document two use cases are described, one for implementing each of the extension class methods (**onLoad** and **onSubmit**):

- **Use case for **onLoad**: The **onLoad** method makes sure that a "Create" task for creating a user in a given department loads in the User Interface with pre-filled information about the department (here "DEVELOPMENT") and an e-mail address for the user is created (here hardcoded "alternative@demo.com").

- **Use case for **onSubmit**: In this use case, the **onSubmit** method makes sure that the telephone number (mobile) of a user that is about to be created (by a "Create" task) is prefixed with the international country code and is valid. Here, the international country code prefix is the hardcoded value "+47". An alternative to using a hardcoded value is to define a custom parameter in the task configuration in the Identity Center Management Console (see section **Referencing the class from the task configuration** on page 34), and use `task.getParameter` in the code to read the defined custom parameter.

**Preparing the environment**

To be able to implement an extension class, the following is needed:

- The Identity Management UI must be deployed on a server running one of the mentioned versions of SAP NetWeaver.

- One of the mentioned versions of SAP NetWeaver Developer Studio must be installed.

- The Identity Management Extension Framework Library (`sap.com~tc~idm~jmx~extfwk~default.jar`) must be present. The file is located in `DesigntimeComponents\Misc` on the install set.

**Section overview**

The tutorial consists of the following sections:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1: Creating the project</td>
<td>In this section we are going to create the project in Developer Studio.</td>
</tr>
<tr>
<td>Section 2: Implementing and deploying a TaskProcessing class</td>
<td>This section describes the necessary steps to implement and deploy the extension class.</td>
</tr>
<tr>
<td>Section 3: Task configuration and class reference</td>
<td>This section shows how to configure the task in the (SAP NW Identity Management) Identity Center Management Console and reference the extension class.</td>
</tr>
</tbody>
</table>
Section 1: Creating the project

This section describes how you create the project in Developer Studio. There are separate sections for Developer Studio 7.0 and 7.1.

The section for Developer Studio 7.0 describes the procedure for the only combination legal as shown in table on page 1, i.e. Developer Studio 7.0 used towards SAP NW 7.0.

In the section for Developer Studio 7.1, the procedures for the following combinations are described:

- Developer Studio 7.1 used towards SAP NW 7.0
- Developer Studio 7.1 used towards EHP 1 for SAP NW CE 7.1, SAP NetWeaver CE 7.2 and SAP NetWeaver 7.3 (the procedures are the same).

Developer Studio 7.0

Creating the project with Developer Studio 7.0 involves the following steps:

- Creating the project: Enterprise Application Project (EAR) and Web Module Project (WAR).
- Adding the Identity Management Extension Framework Library (file sap.com~tc~idm~jmx~extfwk~default.jar located in DesignTimeComponents\Misc on the install set) to the Web Module Project classpath.
- Adding the Web Module Project to the EAR project.
Creating the project

1. Start Developer studio.
2. Create a project of type J2EE/Enterprise Application Project.
3. Choose "Next >".

Enter a name for the project, for instance IdMFwk_EAR.
4. Choose "Finish".

5. Create new project of type J2EE/Web Module Project.

![Image](image1.png)

6. Choose "Next >".

![Image](image2.png)

Enter a name for the project, for instance *IdMFwk_WAR*.

7. Choose "Finish".

---

© Copyright 2015 SAP AG. All rights reserved.
Adding the Identity Management Extension Framework Library to the Web Module Project classpath

After creating the EAR and WAR projects, the IdM Extension Framework Library (interfaces) needs to be added to the Web Module Project classpath. Do the following:

1. Select the Web Module Project "IdMFwk_WAR" and choose "Properties" from the context menu.

![Properties for IdMFwk_WAR](image)

Select "Java Build Path", and then select the "Libraries" tab.
2. Choose "Add External JARs…".

![Properties for IdMFwk_WAR](image)

Select the file `sap.com~tc~idm~jmx~extfwk~default.jar`.

**Note:**
Also make sure that you have SAP Libraries. To add the libraries, choose "Add Library..." from the menu on the right. Select "Server Runtime" and "SAP Libraries, then choose "Finish...".

3. Choose "OK".
4. Set a reference to the Identity Management application from the Enterprise Application Project.

Open the file *application-j2ee-engine.xml*.

Create new reference by selecting "References" and "Add".

Fill in the following information:

**Reference target**
Enter "tc/idm/jmx/app".

**Reference type**
Select "hard".

**Reference target type**
Select "application".

**Provider name**
Enter "sap.com".

5. Save the file.

6. Alternatively, this can be done by adding the following section to the file *application-j2ee-engine.xml*:

```xml
<reference reference-type="hard">
  <reference-target
    target-type="application">
    tc/idm/jmx/app
  </reference-target>
</reference>
```

7. Save the file.
Adding the Web Module Project to the EAR project

To add the Web (WAR) Module to the EAR project, do the following:

1. Select the Enterprise Application Project (EAR).
2. Add the Web project to the EAR project by choosing "Add modules" from the context menu.

![Add Modules Window](image)

Select the WAR project (IdMFwk_WAR).

3. Choose "OK".

© Copyright 2015 SAP AG. All rights reserved.
Developer Studio 7.1

Creating the project with Developer Studio 7.1 involves the following steps:

- Creating the project: Enterprise Application Project (EAR) and Dynamic Web Project (WAR). The Dynamic Web Project is added to the EAR project when created.

- Adding the Identity Management Extension Framework Library (file `sap.com~tc~idm~jmx~extfwk~default.jar` located in `DesignTimeComponents\Misc` on the install set) to the Dynamic Web Project classpath.

The procedure described in this section applies to legal combinations for the Developer Studio 7.1 towards SAP NetWeaver 7.0, EHP 1 for SAP NetWeaver CE 7.1, SAP NetWeaver CE 7.2 and SAP NetWeaver 7.3. The procedure is the same for all combinations if not specified otherwise, and the combination specific steps are described where the procedure for the combinations differs.

Creating the project

1. Start Developer studio.
2. Create a project of type J2EE/Enterprise Application Project.
3. Choose "Next >".

![Eclipse screen shot with EAR Application Project dialog window]  

Fill in the following:

**Project name:**
Enter a name for the project, for instance *IdMFwk_EAR*.

**Project contents:**
Make sure that "Use default" for project contents is selected.

**Target Runtime:**
Make sure that "SAP Libraries" is selected in the "Target Runtime" field.

**Configurations:**
Make sure that the following is selected:

- **For Developer Studio 7.1 in combination with EHP 1 for SAP NW CE 7.1/SAP NW CE 7.2/SAP NW 7.3:** Select "SAP EAR Java EE 5 Project" in the "Configurations" field.

- **For Developer Studio 7.1 in combination with SAP NW 7.0:** Select "SAP EAR J2EE 1.4 Project" in the "Configurations" field.
4. Choose "Next >".

Make sure that the following project facets are selected:

**For Developer Studio 7.1 in combination with EHP 1 for SAP NW CE 7.1/SAP NW CE 7.2/SAP NW 7.3:**

- Create SAP JMS resources descriptor (Version 1.0)
- EAR (Version 5.0)
- SAP Specific Ear Module (Version 5.0)

**For Developer Studio 7.1 in combination with SAP NW 7.0:**

- EAR (Version 1.4)
- SAP Specific Ear Module (1.4)

5. Choose "Next >" and then "Finish".

7. Choose "Next >".
Enter the following:

**Project name:**
Enter a name for the project, for instance *IdMFwk_WAR*.

**Project contents:**
Make sure that "Use default" is selected.

**Target Runtime:**
Make sure that "SAP Libraries" is selected.

**Configurations:**
Make sure that the following is selected:

- **For Developer Studio 7.1 in combination with EHP 1 for SAP NW CE 7.1/SAP NW CE 7.2/SAP NW 7.3:** Select "SAP Web Java EE 5 Project" in the "Configurations" field.

- **For Developer Studio 7.1 in combination with SAP NW 7.0:** Select "SAP Web J2EE 1.4 Project" in the "Configurations" field.

**EAR Membership:**
Select "Add project to an EAR" and enter the EAR project name (*IdMFwk_EAR*) in the "EAR Project Name" field.

8. Choose "Next >".

![New Dynamic Web Project](image)

Make sure that the following project facets are selected:

**For Developer Studio 7.1 in combination with EHP 1 for SAP NW CE 7.1/SAP NW CE 7.2/SAP NW 7.3:**

- Dynamic Web Module (Version 2.5)
- Java (Version 5.0)
- SAP Specific Web Module (Version 2.5)

**For Developer Studio 7.1 in combination with SAP NW 7.0:**

- Dynamic Web Module (Version 2.4)
Section 1: Creating the project

SAP NetWeaver Identity Management Identity Center Implementation guide - Extension Framework

- Java (Version 1.4)
- SAP Specific Web Module (Version 2.4)

9. Choose "Next >"

Make sure that the following is filled in:

**Context Root:**
The context root is `IdMFwk_WAR`.

**Content Directory:**
Here, "WebContent" should be selected.

**Java Source Directory:**
Make sure that "src" is entered.

**Generate Deployment Descriptor:**
Do the following:

- **For Developer Studio 7.1 in combination with EHP 1 for SAP NW CE 7.1/SAP NW CE 7.2/SAP NW 7.3:** make sure that this is selected.
- **For Developer Studio 7.1 in combination with SAP NW 7.0:** this option is not present.

10. Choose "Finish".
Adding the Identity Management Extension Framework Library to the Dynamic Web Project classpath

After creating the EAR and WAR projects, the IdM Extension Framework Library (interfaces) needs to be added to the Dynamic Web Project classpath. Do the following:

1. Select the Dynamic Web Project "IdMFwk_WAR" and choose "Properties" from the context menu.

Select "Java Build Path", and then select the "Libraries" tab.
2. Choose "Add External JARs…".

Select the file sap.com~tc~idm~jmx~extfwk~default.jar.

Note:
Also make sure that you have SAP Libraries. To add the libraries, choose "Add Library..." from the menu on the right. Select "Server Runtime" and "SAP Libraries, then choose "Finish...".

3. Choose "OK".
4. Set a reference to the Identity Management application from the Enterprise Application Project. Open the file `application-j2ee-engine.xml` placed in `EarContentMETA-INF` of the `IdMFwk_EAR` project.

Add the following section to the file:

```xml
<reference reference-type="hard">
    <reference-target provider-name="sap.com" target-type="application">tc/idm/jmx/app</reference-target>
</reference>
```
Section 2: Implementing and deploying a TaskProcessing class

This section describes the necessary steps to implement and deploy the extension class. The javadoc extfwk_javadoc.zip, located in DesigntimeComponentsMisc on the install set, contains information about the classes.

Implementing the class in both Developer Studio 7.0 and Developer Studio 7.1 involves the following steps:

- Creating the class.
- Implementing the class methods `onLoad` and `onSubmit`.
- Adding the class, and the methods `onLoad` and `onSubmit`, to the runtime part of the Identity Management User Interface.
- Building and deploying the application.

The code

The following code is used to implement the class methods `onLoad` and `onSubmit` according to given use cases (see section Use cases on page 2):

```java
package testcustomer.idm.impl;

import java.util.ArrayList;
import java.util.Locale;
import com.sap.idm.extension.IdMExtensionException;
import com.sap.idm.extension.TaskProcessingAdapter;
import com.sap.idm.extension.api.IdMLoadData;
import com.sap.idm.extension.api.IdMSubmitData;
import com.sap.idm.extension.api.IdMValue;
import com.sap.idm.extension.api.IdMValueChange;
import com.sap.idm.extension.api.Task;

public class MyTaskProcessing extends TaskProcessingAdapter {

    /**
     * This onLoad method loads the task with default value for MX_DEPARTMENT
     * (DEVELOPMENT) and adds a new value to attribute MX_MAIL_PRIMARY
     * (alternative@demo.com).
     */
    public IdMValue[] onLoad(Locale locale, int subjectMSKEY, int objectMSKEY,
                            Task task, IdMLoadData data) throws IdMExtensionException {
        // New container, later converted to array of IdMValues
        ArrayList al = new ArrayList();
        // Change of the existing value. Setting value
        // "DEVELOPMENT" as default for attribute MX_DEPARTMENT
        IdMValue[] loadedIdmValues = data.getValues();
        for (int i = 0; i < loadedIdmValues.length; i++) {
            IdMValue aIdmValueToFix = loadedIdmValues[i];
            String aAttr = aIdmValueToFix.getAttributeName();
```
if (aAttr.equalsIgnoreCase("MX_DEPARTMENT")) {
    String origValue = aIdmValueToFix.getOldAttributeValue();
    if (origValue.length() == 0) {
        // Set default department if missing
        // Use value: DEVELOPMENT
        aIdmValueToFix.replaceValue("DEVELOPMENT");
        al.add(aIdmValueToFix);
    }
}

// Add three values to the multivalue attribute MX_MAIL_ADDITIONAL
if (aAttr.equalsIgnoreCase("MX_MAIL_ADDITIONAL")) {
    String origValue = aIdmValueToFix.getOldAttributeValue();
    if (origValue.length() == 0) {
        //add first additional mail
        IdMValue additionalMail = new IdMValue("MX_MAIL_ADDITIONAL", "");
        additionalMail.addValue("firstAdditionalMail@demo.com");
        al.add(additionalMail);

        //add second additional mail
        additionalMail = new IdMValue("MX_MAIL_ADDITIONAL", "");
        additionalMail.addValue("secondAdditionalMail@demo.com");
        al.add(additionalMail);

        //add third additional mail
        additionalMail = new IdMValue("MX_MAIL_ADDITIONAL", "");
        additionalMail.addValue("thirdAdditionalMail@demo.com");
        al.add(additionalMail);
    }
}

// Add new value
IdMValue aIdmValueToAdd = new IdMValue("MX_MAIL_PRIMARY", null);
aIdmValueToAdd.addValue("alternative@demo.com");
al.add(aIdmValueToAdd);

// As the final step, create the array IdMValue[]
IdMValue[] changesInLoad = (IdMValue[]) al.toArray(new IdMValue[al.size()]);

return changesInLoad;

/**
 * This onSubmit method makes sure that the mobile phone number is prefixed
 * with the international country code if it is missing (here "+47").
 */
public IdMValueChange[] onSubmit(Locale locale, int subjectMSKEY,
        int objectMSKEY, Task task, IdMSUBMITData validate)
Section 2: Implementing and deploying a TaskProcessing class

The existing sample code needs to be extended with the bold text above. Also make sure that you have the following Java imports:

- testcustomer.idm.impl
- java.util.ArrayList
- java.util.Locale
- com.sap.idm.extension.IdMExtensionException
- com.sap.idm.extension.TaskProcessingAdapter
- com.sap.idm.extension.api.IdMLoadData
- com.sap.idm.extension.api.IdMSubmitData
- com.sap.idm.extension.api.IdMValue
- com.sap.idm.extension.api.IdMValueChange
- com.sap.idm.extension.api.Task

For the `onLoad` method the input `IdMLoadData data` is actually the `IdMValue[]`, an array consisting of all entry values as read from the identity store. The expected return values are:

- An array of the `IdMValues` that are changed, i.e. a new array consisting of the `IdMValues` is created and returned.
- Null, meaning that no changes are applied to the original data.

The changes applied to the original data may be replacing of the existing value, changing the properties (e.g. of a role/privilege) or adding a new attribute (where the attribute must be a part of the task schema, i.e. the attribute must be selected in the "Attributes" tab of the task).

For the `onSubmit` method the input `IdMSubmitData validate` is actually the `IdMValueChange[]`, an array consisting of all changes made to the original data, both changes introduced during the `onLoad` and changes entered by the user in the User Interface. The expected return values are:

- A list of changes (the optionally manipulated input data).
- Null, meaning that no changes are applied to the original data.
Exception string displayed in the User Interface when validation was not successful.

Note:
Unlike the onLoad method, adding an attribute using the onSubmit method does not require the attribute to be a part of the task schema.

Note:
Also note that only the fields that are changed (the "dirty" fields), either by editing or by onLoad, are subject to validation (onSubmit).

Developer Studio 7.0
To implement the class in Developer Studio 7.0, complete the steps previously listed (on page 19).

Creating the class
Create a new Java Class implementing the TaskProcessingAdapter interface:
1. Select the "source" node of the project and select "New/Java class...":

Fill in the following information:
Make sure that the source folder is IdMFwk_WAR\source.
Enter a name for the package, in this example testcustomer.idm.impl.
Enter a name for the class, here MyTaskProcessing.
Select the name of the class you want to extend, com.sap.idm.extension.TaskProcessingAdapter.
2. Choose "Finish".

**Implementing the methods onLoad and onSubmit**

To implement the class methods `onLoad` and `onSubmit`, do the following:
1. Extend the sample code with the bold text presented in the section *The code* on page 19. Make sure that all imports are present before saving the class.
2. Save the class.

**Adding the TaskProcessing class to the runtime part of Identity Management User Interface**

Create a servlet that is used to initiate the class.
1. Create a servlet by selecting the "IdMFwk_WAR" project and choose "New/Servlet…":

   Enter a "Servlet Name", here *IdMRegisterServlet*.
   Select a "Servlet Package", *testcustomer.idm.impl*.
   Select "init()".
2. Choose "Finish".
3. Register your TaskProcessing class by adding the following line to the init method of your servlet:
IdMFactory.getInstance().registerTaskProcessingInterface(new MyTaskProcessing());

4. If necessary, import the `com.sap.idm.extension.IdMFactory` class and `javax.servlet.ServletException`.

5. Make sure the servlet is invoked on startup. Open the `web.xml` file in the WAR project and navigate to the "Web Objects" tab:

![Web Objects Tab](image)

Make sure that "load on startup" has the value 1 in "IdMRegisterServlet".

6. Save the file.
Building and deploying the application

The last step is to build and deploy the application containing the extension class:

1. If necessary, you must define a J2EE server by opening **Window/Preferences**... and configuring SAP J2EE engine:

   ![Preferences Window](image)

   Specify the host and port number of the J2EE engine.

2. Choose "OK".

3. Build the application by selecting the "IdMFwk_EAR" project and choosing "Build Application Archive" from the context menu.

4. Deploy your application by selecting the built application archive (typically \*IdMFwk_EAR.ear\*) and choosing "Deploy to J2EE engine" from the context menu.
Section 2: Implementing and deploying a TaskProcessing class

Developer Studio 7.1

To implement the class in Developer Studio 7.1, complete the steps previously listed (on page 19).

Creating the class

Create a new Java Class implementing the TaskProcessingAdapter interface:

1. Select the "source" node of the project and select "New/Java class."

   Make sure that the source folder is IdMFwk_WAR\src.

   Enter a name for the package, in this example testcustomer.idm.impl.

   Enter a name for the class, here MyTaskProcessing.

   Select the name of the class you want to extend, com.sap.idm.extension.TaskProcessingAdapter.

2. Choose "Finish".
Implementing the methods `onLoad` and `onSubmit`

To implement the class methods `onLoad` and `onSubmit`, do the following:

1. Extend the sample code with the bold text presented in the section *The code* on page 19. Make sure that all imports are present before saving the class.

2. Save the class.

Adding the TaskProcessing class to the runtime part of Identity Management User Interface

Create a servlet that is used to initiate the class.

1. Create a servlet by selecting the "IdMFwk_WAR" project and choose "New/Servlet":

   ![Create Servlet dialog](image)

   Select a "Java package", `testcustomer.idm.impl`. Enter a "Class name", here `IdMRegisterServlet`.
2. Choose "Next >".

3. Choose "Next >".

Select "init()".

4. Choose "Finish".
5. Register your TaskProcessing by adding the following line to the init method of your servlet:

   ```java
   IdMFactory.getInstance().registerTaskProcessingInterface(new MyTaskProcessing());
   ```

6. If necessary, import the `com.sap.idm.extension.IdMFactory` class and `javax.servlet.ServletException`.

7. Make sure the servlet is invoked on startup. Open the `web.xml` file in the WAR project:

   Add the following line to the file as shown above:

   ```xml
   <load-on-startup>1</load-on-startup>
   ```

8. Save the file.
Building and deploying the application

The last step is to build and deploy the application containing the extension class:

1. If necessary, you must define a J2EE server by opening Window/Preferences..., and configuring SAP AS Java engine:

   ![Preferences window]

   Choose "Add new SAP System".

   ![Add new SAP System dialog]

   Specify the host and the instance number of the engine.
3. Choose "OK".

Make sure that "Default System" is selected.

4. Choose "OK".
5. Create an export by selecting the "IdMFwk_EAR" project and choosing "Export" from the context menu.

Make sure that "EAR application" (here IdMFwk_EAR), "Destination" and "Overwrite existing file" are selected.

6. Choose "Finish".

7. Deploy your application by selecting the created export file and choosing "Deploy" from the context menu.

Logon dialog box appears. Enter administrator credentials for the engine.

8. Choose "OK".

Information that the deployment finished successfully is given.

9. Choose "OK" to close the dialog box.
Troubleshooting

If you are experiencing problems with discovering your class after deployment:

- Check that this is not a caching issue.
- Consider restarting the service/application `sap.com/tc~idm~jmx~app`, e.g. in Visual Administrator.
Section 3: Task configuration and class reference

The class is referenced from the User Interface task configuration in the Identity Center Management Console, the User Interface task for creating a user (here used task *Create User*).

1. View the properties of the task and select the "Result handling" tab:

   Fill in the following fields:

   **Extension class**
   Enter the name of the extension class (here *MyTaskProcessing*).

   **onLoad**
   Select if you want the task to use the extension class method *onLoad*.

   **Note:**
   *If adding attributes in onLoad method the attributes must exist in the task schema, i.e. the attributes need to be selected in the "Attributes" tab of the task. This is not needed for the onSubmit method.*

   **onSubmit**
   Select if you want the task to use the extension class method *onSubmit*.

   **Custom parameters**
   To supply the custom parameters into the extension framework, configure the task specific parameters for the assignment request task by entering the parameter's name, type and value.
The use of custom parameters is an alternative to using hardcoded values. An example, based on the use case for the `onSubmit` method, is to define a custom parameter (e.g. `Countrycode_prefix` of type `String`) with the value "+47" in the task configuration and use `task.getParameter` in the code to read the defined custom parameter instead of adding the hardcoded value.

2. Choose "Apply".

The next time the task is executed, default values are pre-filled and/or the submitted values are validated using the mechanisms in the class (depending on the methods selected – `onLoad`, `onSubmit` or both).

**Testing the solution**

To test the implemented code, do the following:

1. Open the SAP NetWeaver Identity Management User Interface and the User Interface task for creating a user (here task `Create User`).

![Create User form](image)
Observe the following:

- the "Department" field is pre-filled with "DEVELOPMENT"
- e-mail address "alternative@demo.com" is created as specified in the onload method.

2. Fill in the necessary information about the user:

Here, a user Diana Anderson with a unique ID "i10" and a mobile number "90119190" is specified.

Note:
The attribute MX_MAIL_PRIMARY is here defined as a multi-valued attribute with "SingleLine" presentation and the number of rows in the table set to "5".

3. Choose "Save" to execute the task and create the specified user, then close the task.
4. Find the user ("i10") and display its details.

Observe that the primary mobile number for this user is prefixed with the international country code "+47", as specified in the onSubmit method.

**Note:**
Keep in mind that onSubmit executes on changed fields only, e.g. if the telephone number already exists for the user and is not edited in any way in the task, then the telephone number will not be prefixed with the international country code.