Things You Should Know about SAP NetWeaver BRM

Applicable Releases:
SAP NetWeaver CE 7.2

Topic Area:
Business Process Management and Composition

Capability:
Business Rules Management

Version 1.0
June 2010
These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement.

SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

SAP NetWeaver “How-to” Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

Disclaimer
Some components of this product are based on Java™. Any code change in these components may cause unpredictable and severe malfunctions and is therefore expressively prohibited, as is any decompilation of these components. Any Java™ Source Code delivered with this product is only to be used by SAP’s Support Services and may not be modified or altered in any way.

THE BEST-RUN BUSINESSES RUN SAP™
<table>
<thead>
<tr>
<th>Document Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>First official release of this guide</td>
</tr>
</tbody>
</table>
### Typographic Conventions

<table>
<thead>
<tr>
<th>Type Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Example Text</strong></td>
<td>Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>Emphasized words or phrases in body text, graphic titles, and table titles</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>File and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.</td>
</tr>
<tr>
<td><strong>Example text</strong></td>
<td>User entry texts. These are words or characters that you enter in the system exactly as they appear in the documentation.</td>
</tr>
<tr>
<td><strong>&lt;Example text&gt;</strong></td>
<td>Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.</td>
</tr>
<tr>
<td><strong>EXAMPLE TEXT</strong></td>
<td>Keys on the keyboard, for example, F2 or ENTER.</td>
</tr>
</tbody>
</table>

### Icons

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>🚨</td>
<td>Caution</td>
</tr>
<tr>
<td>🎨</td>
<td>Note or Important</td>
</tr>
<tr>
<td>🌟</td>
<td>Example</td>
</tr>
<tr>
<td>🛠️</td>
<td>Recommendation or Tip</td>
</tr>
</tbody>
</table>
Table of Contents

1. Business Scenario .................................................................................................................. 1
2. Background Information ........................................................................................................ 1
3. Prerequisites.......................................................................................................................... 1
4. Designing rules with SAP NetWeaver BRM........................................................................ 2
   4.1 Rules based on XML schema vs. rules based on Java classes ........................................... 2
   4.2 Java classes and rules running on same SAP NetWeaver CE Server vs. on different Application Servers ........................................................................................................... 3
       4.2.1 Same server: Java object model and rules running on same SAP NetWeaver CE server ................................................................................................................................. 3
       4.2.2 Different server: Java object model and rules running on same different Servers .......................................................................................................................... 5
       4.2.3 Common to both scenarios .......................................................................................... 6
   4.3 Application Façade ............................................................................................................ 7
       4.3.1 Application Façade as a stateless session bean......................................................... 10
       4.3.2 Application Façade as a Web Service ....................................................................... 12
5. Appendix................................................................................................................................ 13
   Appendix A - Defining Public Parts ......................................................................................... 13
   Appendix B - Creating Dependencies between Rules Composer DC and Object Model DC .......................................................................................................................... 14
   Appendix C - Creating a DC of type ‘External Library’ in NWDS ........................................... 15
   Appendix D - Creating a Session façade to invoke rules in NWDS ......................................... 18
1. Business Scenario

This guide will cover basic scenarios to help you understand how these aspects may influence the way you define your rules. This guide won’t cover basic steps to create rules on neither Java classes nor XML schema.

2. Background Information

SAP NetWeaver Composition Environment offers SAP NetWeaver BRM (Business Rules Management) to help you manage your business rules, and to maintain more flexible and agile applications that reflect these changing rules.

You can use SAP NetWeaver BRM on its own in any composite or non-SAP applications to handle complex business logic and you can also leverage it in conjunction with SAP NetWeaver BPM (Business Process Management) to integrate, model, and modify rules within the context of a business process. The tight integration between SAP NetWeaver BPM and SAP NetWeaver BRM facilitates the separation of decision logic from process logic.

SAP NetWeaver BRM enables organizations to manage business rules for decision automation. For more information about SAP NetWeaver BRM, please visit Business Rules Management Main Page and Business Rules Management Tutorial Center.

3. Prerequisites

The following is a list represents the prerequisites to use this guide

- You know how to create rules written on Java classes and written on XML schema
- You are familiar with Business Rules Management System
- SAP NetWeaver Composition Environment 7.2 AS Java and NWDS
4. Designing rules with SAP NetWeaver BRM

There are several aspects that need to be considered when creating rules in SAP NetWeaver BRM:

1. If rules should be created based on XML Schema or based on Java classes
2. If rules are based on Java classes, then Java object model and rules deployed on same Application Server vs. on different Application Servers
3. How client application are going to access the rules

4.1 Rules based on XML schema vs. rules based on Java classes

The purpose of an XML Schema is to define the legal building blocks of an XML document. SAP NWDS (NetWeaver Developer Studio) provides an editor that allows developers to create and integrate XML schemas in the definition of the rules.

On the other hand, you may already have Java object model that are being used by existing Java applications. SAP NWDS provides dependency mechanisms that allow developers to integrate existing java objects within the rules definition.

<table>
<thead>
<tr>
<th>Rules based on XML schema</th>
<th>Rules based on Java classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>All required information to modify rules is centralized in the Rules Composer DC within NWDS</td>
<td>All required information to modify rules is scattered in different DC within NWDS</td>
</tr>
<tr>
<td>Two new features in CE 7.2 are available for rules based on XML schema: Web Service Generation menu and Rule Testing</td>
<td>Manual steps are needed in order to generate Web service and to test rules</td>
</tr>
<tr>
<td>Ideal for non-application specific creation of rules</td>
<td>Ideal for complex use cases: use of collection, use of external Web service, integration with backend systems, among others.</td>
</tr>
</tbody>
</table>
4.2 Java classes and rules running on same SAP NetWeaver CE Server vs. on different Application Servers

**Important**
These use cases apply to rules based on Java classes

Making availability of Java classes (object model) to the rules engine depends on where the model is running. There are two variants in this scenario: whether the Java object model is running on same server in which rule engine is running or running on a different server.

<table>
<thead>
<tr>
<th>Same Server</th>
<th>Different Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>The type of DC used to define the Java object model can be: “Java DC”, “Ejb Module DC” or “Web Module DC”</td>
<td>A DC of type “External Library” is needed to include the jar files that contain the Java object model</td>
</tr>
<tr>
<td>A public part of type COMPILATION needs to be created</td>
<td>The External Library DC already has a public part of type COMPILATION called api</td>
</tr>
<tr>
<td>The Java classes have to be added to the public part</td>
<td>The Jar file has to be added to the public part</td>
</tr>
</tbody>
</table>

4.2.1 Same server: Java object model and rules running on same SAP NetWeaver CE server

Let's take a look at the dependency between the Java object model DC and the Rules Composer DC.

When Java object model and rules running on same SAP NetWeaver CE Server the type of DC used to define the Java object model on which rules are implemented can be “Java DC”, “Ejb Module DC” or “Web Module DC”. For simplicity purposes let’s assume our Java object model DC is a Java DC.

1. In the Development Infrastructure perspective, select the Development Component that contains the Java classes and create a public part of purpose ‘COMPILATION’ in the Component Properties view. For more details on how to create public parts go to Appendix A - Defining Public Parts

**Important**

The purpose ‘COMPILATION’ Provides an API for developing/compiling other DCs. The objects in the public part of a DC are needed to compile the Rules Composer DC

2. Using the context menu of the public node, select Manage Entities and select the java class you which to add to public parts

3. The Public Part tab should look like the image below
4. In the Development Infrastructure perspective, select the *Rules Composer DC* and under Dependency Details section, select the Design Time checkbox. For more details on how to create dependencies, go to Appendix B - Creating Dependencies between *Rules Composer DC* and *Object Model DC*.
4.2.2 Different server: Java object model and rules running on same different Servers

When the java object model runs in a different Application server than the rules engine, the object model needs to be packaged in a Jar file and included into an “External Library” DC to make it accessible to the Rules Composer DC

1. Create External Library DC and add the object model as a jar file. Check Appendix C for more details.

2. In the Development Infrastructure perspective, select the External Library DC select the public part api and using the context menu of the public node api, select Manage Entities and select the java class you which to add to public parts

3. In the Development Infrastructure perspective, select the Rules Composer DC and under Dependency Details section, only select the Design Time checkbox
4.2.3 **Common to both scenarios**

Once the dependency is defined, developers can add the Java class to the Aliases and use the class to model business rules.
4.3 Application Façade

A JEE application can lookup and invoke the Rule Engine as a stateless session bean directly if the application which is invoking the rules and the Rule Engine are running on same JVM (same SAP CE Server). However if the application which is calling the rules is running in a different server in which Rule Engine is running, it’s mandatory to create a session bean (Application Façade) that will invoke the Rule Engine as a stateless session bean internally. The Session Façade can also be exposed as a reusable Web service.

**Important**

The creation of the Application Façade is more suitable for rules based on Java classes.

For rules based on XML schema, there’s a simple menu option that generates the wsdl artifact

<table>
<thead>
<tr>
<th>Application Façade as a stateless session bean</th>
<th>Application Façade as a Web service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only Java application can invoke the Application Façade as a stateless session bean</td>
<td>Any type of application can invoke the application façade as a Web service</td>
</tr>
<tr>
<td>The location of the application façade it’s important when using EJB Lookup Scheme. Some properties need to be added when the application client is in a different SAP NetWeaver Application Server Java than the session façade.</td>
<td>The location of the web Service (remote or local) is transparent to the application client</td>
</tr>
</tbody>
</table>

**Note**

Application Façade is not necessary if the rules and the application which is calling the rules are running on same JVM (same SAP CE Server). As BRM rule engine is deployed while installing SAP CE Server and it will be running in EJB mode. We generally look up the rule engine running on Server and then call a method(invokeRuleset()) with list of objects to invoke a ruleset, now in SAP BRM this look up should be ‘local look up’ only and ‘remote look up’ will not work if we are using java classes to write rules.

The solution to this is that we can have an intermediate session bean between our application and SAP BRM engine which loop up the BRM engine locally and our application look up and call this intermediate session bean remotely. This intermediate bean is deployed into SAP server so it can call BRM engine locally.

**More background information:**

RuleEngine is a Session Bean with both remote and local interfaces (i.e., RuleEngineRemote and RuleEngineLocal respectively). Both remote and local interfaces extend RuleEngine interface.

An instance of RuleEngine can be obtained through JNDI lookup of either remote or local interface. Local interface lookup can only be done if both the RuleEngine Session Bean and the RuleEngine Client [Application which invokes RuleEngine] are in same JVM. Otherwise only remote interface can be used.
Local Interface

When an instance of RuleEngine is obtained through local interface and a ruleset is invoked, there is no remote method call involved and thus no serialization/deserialization of method parameters and return value happens. In this case, the RuleEngine will be able to get the Java classes used in rules from the input objects.

Remote Interface

But when it comes to remote interface lookup, the method call is remote and thus serialization/deserialization of method parameters and return value happens. But the Enterprise Application [EAR] containing the RuleEngine Session Bean is pre-deployed into the server. So when the method parameters arrive at the RuleEngine, deserialization of the method parameters fails as the corresponding Java classes are not available.

Creating Session Bean (Application Façade)

Please go to Appendix D - Creating a Session façade to invoke rules in NWDS for more details.

Looking Up the Rules Engine

The session façade will invoke the Rule Engine internally. The Rules Engine provides an environment to evaluate, infer, and execute rules. You can look up the Rules Engine either locally or remotely.
Local interfaces are meant to be used from the same JVM as the application server on which your bean is deployed. Remote interfaces can be used from other (remote) JVMs. For more information visit Executing Rules.

**Looking up the Rules Engine remotely**

- **Note**
  - RuleEngineHome.JNDI_NAME is the JNDI Name used for Remote Lookup. Use the following code for remote lookup of rules engine

```java
InitialContext context = getInitialContext();
Object obj = context.lookup(RuleEngineHome.JNDI_NAME);
RuleEngineHome home = (RuleEngineHome) PortableRemoteObject.narrow(obj, RuleEngineHome.class);
RuleEngineRemote remote = home.create();
RuleEngine engine = (RuleEngine) remote;
```

**Looking up the Rules Engine locally**

- **Note**
  - RuleEngineLocalHome.JNDI_NAME is the JNDI Name used for Local Lookup. Use the following code for local lookup of rules engine

```java
InitialContext context = getInitialContext();
RuleEngineLocalHome home = (RuleEngineLocalHome) context.lookup(RuleEngineLocalHome.JNDI_NAME);
RuleEngineLocal local = home.create();
RuleEngine engine = (RuleEngine) local;
```

**Looking up the Rules Engine using annotations**

- **Note**
  - Use the @EJB annotation to inject either a session bean’s or entity bean’s interfaces, which can be either local or remote. You can use the @EJB annotation to look up both EJB 3.0 and EJB 2.1 bean references

1. Use the @EJB annotation to declare the RuleEngineHome interface
   ```java
   @EJB(name=RuleEngineHome.JNDI_NAME, beanInterface=RuleEngineHome.class)
   private RuleEngineHome rulesEngineHome;
   ```

2. Create the RuleEngine reference
   ```java
   RuleEngine ruleEngine = (RuleEngine) rulesEngineHome.create();
   ```

**Invoking rules**

It doesn’t matter how you look up the Rules Engine. Once you get the RuleEngine reference, you can use the method / contract defined as invokeRuleset. Use the invokeRuleset contract to invoke the Engine.
List <list of changed objects> =
engine.invokeRuleset(<PROJECTNAME>,<RULESETNAME>,<List of business objects>);  

**Important**
If an action uses an object called OrderItem, it succeeds only if an OrderItem object is available in the Business Objects asserted to the engine.

If an object that is needed for execution of a Ruleset is not passed in Business Objects asserted to the engine, but created during the execution, the latest object is used for firing rules. If multiple objects are passed, Ruleset will use all the objects.

### 4.3.1 Application Façade as a stateless session bean

Only Java application can invoke the Application Façade as a stateless session bean. The location of the application façade it's important when using EJB Lookup Scheme. Some properties need to be added when the application client is in a different SAP NetWeaver Application Server Java than the session façade.

Getting Application Façade's JNDI Information...

1. Open the NetWeaver Administrator [http://<host>:<port>/nwa](http://<host>:<port>/nwa)
2. Navigate through **Problem Management** → **Java** → **JNDI Browser**

3. Look for the corresponding Application Façade interface, in this example we are going to use the Remote interface to demonstrate how to look up Application Facade from remote clients
4. Select the Remote interface and

5. Keep this values in mind for Accessing Application Facade from Remote Clients

Accessing Application Facade

To access Enterprise JavaBeans deployed in an application you use the JNDI API. JNDI Registry Service is the standard way to associate names with objects and to find objects by their names. For more information visit Accessing Enterprise JavaBeans from Remote Clients

1. You create the InitialContext object by specifying the JNDI properties. In addition to the standard properties you have to set the new property:
Things You Should Know about SAP NetWeaver BRM

props.put(Context.URL_PKG_PREFIXES, "com.sap.engine.services"), which is obligatory when you use the EJB lookup scheme.

```java
Properties props = new Properties();
props.put(Context.INITIAL_CONTEXT_FACTORY, "com.sap.engine.services.jndi.InitialContextFactoryImpl");
props.put(Context.PROVIDER_URL, "<host>:<port>");
props.put(Context.SECURITY_PRINCIPAL, "<user>");
props.put(Context.SECURITY_CREDENTIALS, "<pass>");
props.put(Context.URL_PKG_PREFIXES, "com.sap.engine.services");
InitialContext ctx = new InitialContext(props);
```

2. If your client is running on the same SAP NetWeaver Application Server Java, you do not need the Properties object while creating the InitialContext:

```java
InitialContext ctx = new InitialContext();
```

3. You look up the bean using the specific lookup string in the root context ctx.

```java
Object obj = ctx.lookup(<lookup-string>);
```

You have to replace the `<lookup-string>` with the Application Facade JNDI information, for example:

```java
Object obj = context.lookup
("ejb:/appName=demo.sap.com/example~sessionfacade~ear,beanName=RulesFacade,
interfaceName=com.sap.example.ejb.RulesFacadeRemote");
```

### 4.3.2 Application Façade as a Web Service

You can expose a session Enterprise Java Bean (EJB), or a pure Java class as a Web service. For more details, please visit [Creating an Inside-Out Web Service from an Enterprise Java Bean](#).

1. In the Package Explorer of the Java EE perspective, open the project and select the EJB you created.

2. From the context menu, choose Web Services ® Create Web Service. The Web Services wizard opens.

3. Move the slider to the Develop service position,

   **Note**

   From Configuration, select SAP server and SAP NetWeaver Web service runtime. Make sure that the service project and service EAR project are set correctly.

4. Follow the wizard instructions and choose Finish

5. The Developer Studio adds Java EE 5 annotations for Web services to the implementation bean, as well as to the service endpoint interface, if applicable

6. Build and deploy the Enterprise Application Project of the Web service on the SAP NetWeaver Application Server
5. Appendix

Appendix A - Defining Public Parts

1. Make sure you are in the Development Infrastructure perspective.
2. In the Component Browser view, expand the Local Development node, MyComponents [demo.sap.com] node, and double-click the Development Component where the Java classes are.
3. In the Component Properties view, choose the Public Parts tab and in the page that appears, choose Add.
4. In the screen that appears, enter ppComp in the Name field and COMPILATION in the purpose field. Choose Finish.
5. You should see the public node under the Defined Public Parts section.
6. In the context menu of the public node, choose Manage Entities.
7. In the screen that appears, under the Entities section, expand the Java Class and select the java class you which to add to public parts.

![Component Properties Screen]

1. Choose Add.
9. In the screen that appears, enter ppAsem in the Name field and ASSEMBLY in the purpose field. Choose Finish.
10. You should see the public node under the Defined Public Parts section.
11. In the context menu of the public node, choose Manage Entities.
12. In the screen that appears, under the Entities section, expand the Java Class and select the java class you which to add to public parts.
Appendix B - Creating Dependencies between Rules Composer DC and Object Model DC

1. In the Component Browser view, view, expand the Local Development node, MyComponents [demo.sap.com] node and double-click the Rules Composer DC node.

2. In the Component Properties view, choose the Dependencies tab and in the page that appears, choose Add.

3. In the dialog box that appears, expand the MyComponents node and select the Object model DC, “example/datasource” Java DC in this example. Choose Next.

4. In the screen that appears, under Dependency Details section, only select the Design Time checkbox.

5. Choose Finish.

**Important**

Only select the Design Time checkbox. Only the public part with purpose COMPILATION should be included.
Appendix C - Creating a DC of type 'External Library' in NWDS

1. To create a new External Library Development Component, select File > New > Others

2. In the pop up window that appears, select Development Component and click the Next button

3. Select External Library and click the Next button
4. Select the corresponding Software Component, for example *MyComponents* and enter the name of the External Library DC and click the Finish button.

5. Inside the NWDS workspace, copy the jar file in the Libraries folder in the External Library DC.
6. Refresh the project, so the jar file appears in the NWDS
Appendix D - Creating a Session façade to invoke rules in NWDS

Creating EJB Module

1. Create a new Development Component Project of type EJB Module. For this select ‘File’ → ‘New → Other’ from the menu. Next choose ‘Development Component’ underneath the ‘Development Infrastructure’ node. Press Next.

2. Choose ‘EJB Module’ and click on ‘Next’.

3. Select the corresponding Software Component and enter the name of the Ejb module DC and click the Finish button.
4. In the Project Explorer view, select the ejbModule folder and from the context menu choose New → Other → EJB → Session Bean

5. Enter Java package and Class name
6. Add the corresponding dependencies
   a. *tc/brms/facade* from BRMS-FACADE[sap.com] allows you to use the Rule engine
      Remote interface and look up the Rule Engine
   b. *example/datasource* allows you to use the classes define in the datasource DC

![Diagram showing dependencies]

7. Add an attribute of type RuleEngineHome and annotate it as follows

```java
package com.sap.example.ejb;

import java.util.ArrayList;

//@EJB(name="RuleEngineHome.JNDI_NAME", beanInterface=RuleEngineHome.class)
public class RulesFacadeLocal {
    private RuleEngineHome rulesEngineHome;
}
```

**Note**
Use the @EJB annotation to inject either a session bean's or entity bean's interfaces, which can be either local or remote. You can use the @EJB annotation to look up both EJB 3.0 and EJB 2.1 bean references

8. Define two constants with the project name and the ruleset name to invoke the rules
9. Define the method that will invoke the rules internally.

```
public PurchaseOrder getApprovalNecessity (PurchaseOrder pOrder)
{
    PurchaseOrder poResponse = new PurchaseOrder();
    try {
        // Preparing input
        List<PurchaseOrder> inputList = new ArrayList<PurchaseOrder>();
        inputList.add(pOrder);

        RuleEngine ruleEngine = (RuleEngine) rulesEngineHome.create();

        // Invoking rules
        List output = ruleEngine.invokeRuleSet(projectName, rulesetName, inputList);
        Object obj = null;
        if (output.size() == 0) {
            poResponse.setMessage("no output returned. input =" + inputList.size());
        } else {
            // Preparing output
            obj = output.get(0);
            if (obj instanceof PurchaseOrder) {
                poResponse = (PurchaseOrder) obj;
            }
        }
    } catch (Exception e) {
        poResponse.setMessage(e.getMessage());
    }
    return poResponse;
}
```

Creating Enterprise Application

1. Create a new Development Component Project of type EJB Module. For this select ‘File’ → ‘New’ → ‘Other’ from the menu. Next choose ‘Development Component’ underneath the ‘Development Infrastructure’ node. Press Next
2. Choose ‘Enterprise Application’ and click on ‘Next’

3. Select the corresponding Software Component and enter the name of the Enterprise Application module DC and click the Next button
4. Select the corresponding EJB Module

![New EAR Project](image)

5. Add the corresponding dependencies
   a. `tc/brms/facade` from BRMS-FACADE[sap.com] allows you to use the Rule engine Remote interface and look up the Rule Engine
   b. `example/datasource` allows you to use the classes define in the datasource DC

![Component Properties](image)

6. Build and Deploy. Your Session Bean is ready to use