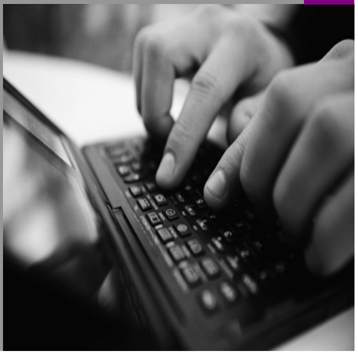
How-to Guide
SAP NetWeaver '04



How To... EJB: Accessing EJB Applications Using JNDI

Version 1.00 – April 2005

Applicable Releases:
SAP NetWeaver '04
(J2EE Engine)

THE BEST-RUN BUSINESSES RUN SAP

Contents

Introduction

When developing your enterprise bean, you should bear in mind how it will be accessed, and then define which interfaces you will expose to your bean's clients, that is, define its client view.
In the EJB client source code, you usually need to obtain an object reference to the bean in order to invoke its business methods. There are several things you should consider to implement the EJB reference lookup:

- The type of EJB client that you want to use
- Where the client will reside, that is, whether it will access your beans locally or remotely

The next steps are to create your EJB client, get a reference to the bean and then to invoke the bean's methods.
Limitations: Note that this guide focuses on the ways of accessing session and entity enterprise beans by a variety of EJB clients using JNDI, the remote versus local client view, and on the techniques and steps that you have to implement in the EJB clients to be able to access the beans and to invoke their methods. Therefore this guide will illustrate with code samples the actual access code only, it is not intended to demonstrate how you develop an entire EJB client of a specific type. Also, this guide assumes that you are using J2EE Engine 6.30 or 6.40, and EJB 2.0.

Types of EJB Clients

One of the advantages of the EJB technology is that once it has been developed, an EJB application can be accessed by a variety of clients. To begin with, the EJB clients can be the standard, defined by the J2EE specification, J2EE-compliant clients – that is, Web components, J2EE application clients, and other enterprise beans. Secondly, your EJB application can be accessed by non-J2EE components – namely, you can access the EJB application from within a WebDynpro application, Portal application, or even from within a library deployed on the J2EE Engine. Finally, you can access your beans from within an ABAP application and as Web Services.
Moreover, you are not limited to the programming language when creating your EJB client – enterprise beans can be accessed by both an arbitrary Java program and by non-Java clients (such as CORBA clients that are not written in the Java programming language).

J2EE Components

J2EE components (Web components, other enterprise beans, J2EE application clients) are defined by the J2EE standard.
Advantages: Defined by the specification; declarative reference to the enterprise bean.
Disadvantages: Limited to Java and the J2EE standard.

Non-J2EE Java Components

In this document we refer to non-J2EE components as either J2EE Engine libraries, WebDynpro or Portal applications, or a standalone Java application.
Advantages: Allow integration between the different NetWeaver technologies and with Java applications residing outside of the J2EE Engine.
Disadvantages: Not standardized; not portable.

Other Clients

ABAP programs can access EJB applications via JCo RFC. The other non-Java clients (such as CORBA clients) and Web Services access the enterprise beans in their own specific way without using the JNDI. Therefore, accessing enterprise beans from these clients is not described in this guide.
Advantages: Provide integration between ABAP and Java applications, provide interoperability.
Disadvantages: No local access, complicated to implement.

Local vs Remote Access

The pros and cons of both:

- Java Virtual Machine – the first thing to consider is the location of the beans and the clients. To access the beans locally, your clients must reside in the same JVM, that is, they must be Java components deployed on the J2EE Engine. With the remote access, however, your clients are not limited to the same JVM.
- Sharing of objects passed through the bean interfaces – with remote access, the arguments and the results from the methods of the remote and remote home interfaces are passed by value. With local access, however, the arguments and the results from the methods of the local and local home interfaces are passed by reference, therefore you must code your beans assuming that the arguments and results may be shared between the bean and the bean client.
- The types of method arguments – when you use remote access, you have to make sure that the objects that are passed as parameters and the returned values of the bean's methods are correct RMI-IIOP types, that is, they can be serialized. With local access, there are no limitations to the type of method arguments.
- Performance – using remote access when the target bean and the client reside in the same JVM will generate additional performance overhead due to parameters replication and should be avoided.

Enterprise beans that will be accessed locally must provide a local interface and a local home interface. Enterprise beans that will be accessed remotely must provide a remote interface and a remote home interface. Enterprise beans that will provide both a local and a remote client view must provide both pairs of interfaces.

Accessing Enterprise Beans

Let us assume that you have created a stateless session bean, *MyExampleBean*. The bean's package is *my.ejbproject*. The bean will provide both a local and a remote view. The bean is packaged in a JAR file named *MyEJBJAR.jar*, which in turn is packaged in an EAR file named *MyEAR.ear*. The excerpt from the bean's *ejb-jar.xml* deployment descriptor describing *MyExampleBean* looks as follows:

```
<session>
    <ejb-name>MyExampleBean</ejb-name>
    <home>my.ejbproject.MyExampleHome</home>
    <remote>my.ejbproject.MyExample</remote>
    <local-home>my.ejbproject.MyExampleLocalHome</local-home>
    <local>my.ejbproject.MyExampleLocal</local>
    <ejb-class>my.ejbproject.MyExampleBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
</session>
```

where:

- `<ejb-name>` is the bean's name.
- `<home>` is the bean's home interface.
- `<remote>` is the bean's remote interface.
- `<local-home>` is the bean's local home interface.
- `<local>` is the bean's local interface.
- `<ejb-class>` is the bean's class.
- `<session-type>` specifies that the session bean is a stateless session bean.
- `<transaction-type>` specifies the bean's transaction management type.

The Bean's JNDI Name

This is the name with which the beans' home interface is bound in the JNDI namespace. By default, the JNDI name is ***<application provider>/<application name>/<EJB name>***, where

- <application provider> is the name of the application provider as specified in the `<provider-name>` element in *application-j2ee-engine.xml*,
- <application name> is the name of the application as specified in the `<display-name>` element in *application.xml*,
- <EJB name> is the name of the bean as specified in the `<ejb-name>` element in *ejb-jar.xml*.

The local home interface of the bean is registered in the JNDI namespace with a */localejbs* prefix; that is, */localejbs/<application provider>/<application name>/<EJB name>*.

Hint: If you are not sure of the JNDI name with which your bean is registered in the JNDI, check it using the Visual Administrator tool:

1. Start the Visual Administrator (from the */usr/sap/<system ID>/<instance name>/j2ee/admin* directory) and connect to the J2EE Engine.
2. Browse the tree structure on the left to locate the EJB Container Service (*Server <n>* → *Services* → *EJB Container*).
3. Select the *Beans* tree structure on the right and choose <your deployed application> → <the bean's JAR> → <your bean>.

The JNDI name of the selected bean is displayed in the *JNDI Name* field on the right.

In our case, the bean is registered in the JNDI as *sap.com/MyEAR/MyExampleBean*.

The general recommendations when you access enterprise beans are as follows:

- When you look up the beans from J2EE clients, declare EJB references in their standard deployment descriptors and look up the beans using these references;
- When you look up the beans from non-J2EE clients, use the default JNDI names (described above).

For detailed examples, see below.

Example 1: Accessing Enterprise Beans by J2EE Components (when the client component and the referenced bean are packaged in the same application)

1. Declare an EJB reference to the bean that you want to access.

   The enterprise bean client uses the enterprise bean's reference name (which you defined in the client's deployment descriptor) to look up the bean.

   - When the client accesses the bean through the bean's local interfaces (the client can be a Web component or another enterprise bean):

     Declare a local EJB reference in the `<ejb-local-ref>` element of *web.xml* (for Web component clients) or *ejb-jar.xml* (for enterprise bean clients). Example:

```
<ejb-local-ref>
      <ejb-ref-name>ejb/MyLocalRef</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <local-home>my.ejbproject.MyExampleLocalHome</local-home>
      <local>my.ejbproject.MyExampleLocal</local>
      <ejb-link>MyExampleBean</ejb-link>
</ejb-local-ref>
```

where:

- `<ejb-ref-name>` is the EJB reference name by which you will look up the referenced bean. This name is an entry in the client's environment (relative to the java:comp/env context) and it must be unique within the client.
- `<ejb-ref-type>` is the expected type of the referenced bean. Possible values: `Session` and `Entity`.
- `<local-home>` is the referenced bean's local home interface.
- `<local>` is the referenced bean's local component interface.
- `<ejb-link>` is an optional element, additionally identifying the referenced bean. It is meaningful only if the referenced bean and the client component belong to the same application. The value of this element is one of the following:
  - The `ejb-name` of the referenced bean (if the client is another bean packaged in the same JAR file as the referenced bean)
  - `<The name of the target bean's JAR file>#<ejb-name>` (when the referenced bean belongs to another JAR file, but still to the same application as the client)

- When the client accesses the bean through the bean's remote interfaces (the client can be a Web component, another enterprise bean, or a J2EE application client):

  Declare a remote EJB reference in the `<ejb-ref>` element of *web.xml*, *ejb-jar.xml*, or *application-client.xml* accordingly. Example:

```
<ejb-ref>
      <ejb-ref-name>ejb/MyRemoteRef</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>my.ejbproject.MyExampleHome</home>
      <remote>my.ejbproject.MyExample</remote>
      <ejb-link>MyEJBJAR.jar#MyExampleBean</ejb-link>
</ejb-ref>
```

  where:

  - <home> is the referenced bean's home interface.
  - `<remote>` is the referenced bean's remote interface.

2. In the source code of the client, create the JNDI *InitialContext* object:

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

3. Look up the home interface of the bean via the *InitialContext*. Use the `<ejb-ref-name>` defined in the EJB references.
   - To look up the bean locally, use:

```
MyExampleLocalHome myExampleLocalHome = (MyExampleLocalHome)
ctx.lookup("java:comp/env/ejb/MyLocalRef");
```

   - To look up the bean remotely, use:

```
MyExampleHome myExampleHome = (MyExampleHome)
javax.rmi.PortableRemoteObject.narrow(ctx.lookup("java:comp/e
nv/ejb/MyRemoteRef"), MyExampleHome.class);
```

   Note that when you look up the remote home interface of the bean, you must narrow the remote reference after the lookup.

Example 2: Accessing Enterprise Beans by J2EE Components (when the client component belongs to a different application)

This example slightly differs from Example 1 in that you have to declare a reference from the client application to the target application:

1. Declare the reference to the referenced bean's application in the *application-j2ee-engine.xml* deployment descriptor of the client application:

```
<reference reference-type="hard">
      <reference-target
            provider-name="sap.com"
            target-type="application">MyEAR</reference-target>
</reference>
```

   where the value of the `<reference-target>` element must be the same as the value of the `<display-name>` element in the *application.xml* of the referenced bean's application.

2. Follow all the steps described in Example 1.

   When declaring the EJB references, the `<ejb-link>` element should not be specified.

3. (Only if required) Specify the JNDI name of the referenced bean.

   This step is required only if the information in the `<ejb-ref>` or `<ejb-local-ref>` elements is not sufficient to locate the target bean (for example, when more than one bean with the same interfaces is deployed on the J2EE Engine). In such cases, you can identify the target bean by declaring its JNDI name. This name is specified in the `<ejb-ref>` or `<ejb-local-ref>` element in the client's additional deployment descriptors – *ejb-j2ee-engine.xml*, *web-j2ee-engine.xml*, or *appclient-j2ee-engine.xml*. Example:

```
<ejb-ref>
      <ejb-ref-name>ejb/MyRemoteRef</ejb-ref-name>
      <jndi-name>sap.com/MyEAR/MyExampleBean</jndi-name>
</ejb-ref>
```

   where:

   - `<ejb-ref-name>` must be the same as the `<ejb-ref-name>` specified in the standard deployment descriptor (*ejb-jar.xml*, *web.xml*, or *application-client.xml*).
   - `<jndi-name>` is the JNDI name of the referenced bean with which it is registered in the JNDI namespace. See *The Bean's JNDI Name* above.

   Note that although you specify the bean's JNDI name, you still have to use the EJB reference name (that you have defined in the <ejb-ref-name> element) to look up the bean.

Example 3: Accessing Enterprise Beans by Non-J2EE Components

This scenario differs from Example 1 and Example 2 in that non-J2EE components are not able to declare EJB references to the referenced bean. Instead, to access the bean, they use the bean's JNDI name. See *The Bean's JNDI Name* above.
Depending on the type of the client and where it is executed, some non-J2EE clients can access the beans both locally and remotely (for example, WebDynpro applications) and others can access the beans only remotely.

1. In the source code of the client, create the JNDI *InitialContext* object.
   - If the client resides on the same J2EE Engine on which the bean is deployed, use the following code:

```
javax.naming.Context ctx = new javax.naming.InitialContext();
```

- If the client is a standalone Java application, it has to connect to the JNDI context of the J2EE Engine on which the bean is deployed:

```
import javax.naming.*;
import java.util.*;
...
Properties props = new Properties();
props.put(Context.PROVIDER_URL,"localhost:50004");
props.put(Context.INITIAL_CONTEXT_FACTORY,
"com.sap.engine.services.jndi.InitialContextFactoryImpl");
Context ctx = new InitialContext(props);
```

2. Look up the bean.
   - To look up the bean locally, use:

```
MyExampleLocalHome myExampleLocalHome = (MyExampleLocalHome)
ctx.lookup("localejbs/sap.com/MyEAR/MyExampleBean");
```

   Note that when you look up the local home interface of the bean, you have to use the *localejbs/* prefix in front of the JNDI name.

   - To look up the bean remotely, use:

```
MyExampleHome myExampleHome = (MyExampleHome)
javax.rmi.PortableRemoteObject.narrow(ctx.lookup("sap.com/MyE
AR/MyExampleBean"), MyExampleHome.class);
```

   Note that when you look up the remote home interface of the bean, you must narrow the remote reference after the lookup.

3. Ensure that the non-J2EE client can load the referenced bean's classes.

   If the client component is deployed on the J2EE Engine, a reference between the client component and the bean's application is needed. A standalone application must be able to load the bean's classes as well as all client classes of the J2EE Engine.

Example 3a: Accessing Enterprise Beans by Non-J2EE Components (defining an arbitrary JNDI name for the referenced bean)

It is possible to define an arbitrary JNDI name for an enterprise bean to overwrite the default JNDI name. You can later use that arbitrary name to access the bean.
However, there are several limitations to that arbitrary JNDI name:

- The arbitrary JNDI name that you define must be unique within all deployed enterprise beans and all objects bound in the JNDI namespace of the J2EE Engine
- The prefix of the arbitrary JNDI name (if you specify any) must not coincide with the name of any other object bound in the JNDI namespace.

Due to these limitations, we recommend that you do not specify or use the arbitrary JNDI name.
Hint: Specifying and using this arbitrary JNDI name might facilitate migration in case you port an application from other application servers to the J2EE Engine.
If you consider using this option, follow the steps below:

1. Define the arbitrary JNDI name of the enterprise bean in the `<jndi-name>` element of the bean's *ejb-j2ee-engine.xml* deployment descriptor:

```
<enterprise-bean>
      <ejb-name>MyExampleBean</ejb-name>
      <jndi-name>ArbitraryJNDIname</jndi-name>
      ...
</enterprise-bean>
```

2. In the source code of the client, create the JNDI *InitialContext* object. See *Example 3, step 1* above.
3. Look up the bean.

- To look up the bean locally, use:

```
MyExampleLocalHome myExampleLocalHome = (MyExampleLocalHome)
ctx.lookup("localejbs/ArbitraryJNDIname");
```

- To look up the bean remotely, use:

```
MyExampleHome myExampleHome = (MyExampleHome)
javax.rmi.PortableRemoteObject.narrow(ctx.lookup("ArbitraryJND
Iname"), MyExampleHome.class);
```

4. Ensure that the non-J2EE client can load the referenced bean's classes. See *Example 3, step 3* above.