How-To: JNDI Names and EJB References with WebSphere for z/OS

The purpose of this "How-To" document is to illustrate how to write servlets and EJBs in WebSphere Studio Application Developer (WSAD) for deployment on WebSphere Version 4.0.1 for z/OS and OS/390. These applications look up EJBs using the `"java:comp/env"` construct. This technique allows applications to be deployed without the components having any knowledge of the actual JNDI (Java Naming and Directory Interface) names of the components they reference.

To illustrate this, an application will be created and tested in WSAD, and then exported as a J2EE .ear file. The .ear file will be imported into the Application Assembling Tool for WebSphere for z/OS (AAT-z/OS) for final verification and code generation, and then installed into an existing WebSphere for z/OS server using the WebSphere for z/OS Systems Management Administration application (also known as the Systems Management End-User Interface, or "SM EUI").

Notices to the Reader:

- This document assumes the reader has a working knowledge of the WebSphere for z/OS runtime, WSAD, AAT, and Administration application (SM EUI) tools.
- For an overview of JNDI naming, see "WebSphere for z/OS JNDI Naming Concepts" (WP100268) at http://www.ibm.com/support/techdocs .
- This document may be updated; you may retrieve the latest copy from the same techdocs web site at http://www.ibm.com/support/techdocs/atsmastr.nsf/PubAllNum/WP100268

Part 1: Create the Simple Application.

This J2EE application consists of one servlet and two session beans. The details of the components are:

JNDIServlet — A servlet whose init( ) method looks up the Bean1Home and whose doGet() method creates an instance of Bean1 and drives the doIt( ) method on the Bean1 instance. The servlet doesn't return any data to the browser.

Bean1 — A stateless session bean whose doIt( ) method looks up the Bean2Home, creates an instance of Bean2 and drives the doIt( ) method on the Bean2 instance.

Bean2 — A stateless session bean whose doIt( ) method prints a message and returns. Bean2 is essentially a copy of Bean1.

The reason to have a servlet and two session beans is that the process of defining ejb-refs in WSAD is slightly different for a servlet than for an EJB. Both will be shown.

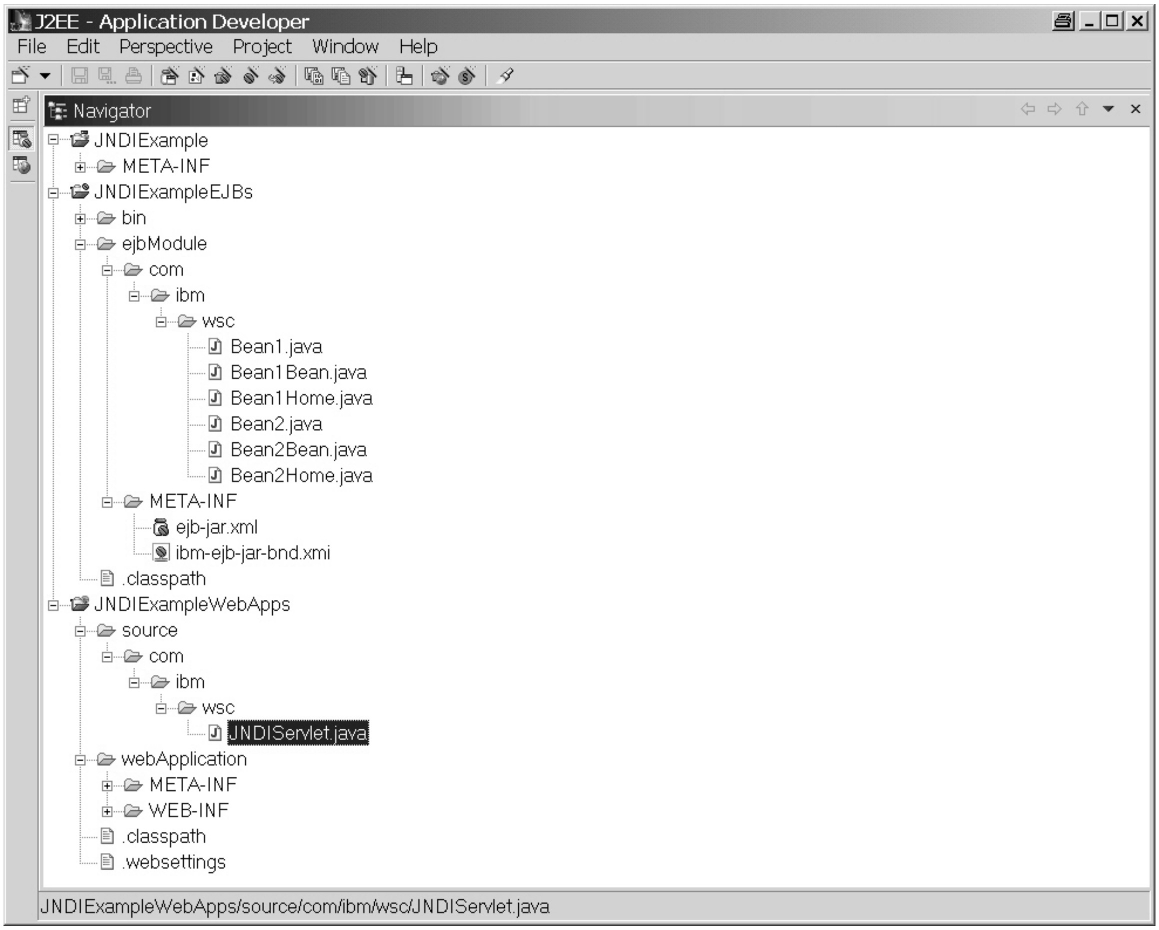
The following steps were done in preparation for this exercise but will not be shown.

1. A new WSAD workspace was created.
2. A J2EE Application named JNDIExample was created in the workspace.
3. An EJB project named JNDIExampleEJBs was created and associated with the J2EE Application that was created in the prior step. Two session beans were created in this EJB project: com.ibm.wsc.Bean1 and com.ibm.wsc.Bean2. In each

case, a doIt( ) method was defined on the bean and promoted to the remote interface as the only remote business method. The contents of the doIt( ) methods will be discussed later.

4. A Web Application project named JNDIExampleWebApps was created and associated with the J2EE application created previously. This web app project is dependent upon the EJB project. The Web App was assigned a context root of: JNDI. A single servlet was created in this project: com.ibm.wsc.JNDIServlet, with a URI mapping of: LookUp. The contents of the init( ) and doGet( ) method will be discussed later.

Once all these tasks are accomplished in WSAD the process of deploying the application can be started. Select the Navigator tab of the J2EE perspective. Below are the contents of the workspace.


J2EE - Application Developer
File Edit Perspective Project Window Help
Navigator

```
JNDIExample
    META-INF
JNDIExampleEJBs
    bin
    ejbModule
        com
            ibm
                wsc
                    Bean1.java
                    Bean1Bean.java
                    Bean1Home.java
                    Bean2.java
                    Bean2Bean.java
                    Bean2Home.java
        META-INF
            ejb-jar.xml
            ibm-ejb-jar-bnd.xmi
    .classpath
JNDIExampleWebApps
    source
        com
            ibm
                wsc
                    JNDIServlet.java
    webApplication
        META-INF
        WEB-INF
    .classpath
    .websettings
```

JNDIExampleWebApps/source/com/ibm/wsc/JNDIServlet.java

It is important to understand from an application coding perspective, the proper technique to "lookup" an EJB Home in a J2EE server. The code in the JNDIServlet to lookup the Bean1Home and the code in Bean1 to lookup Bean2Home is provided below:

JNDIServlet

The servlet is structured so that the Bean1Home is looked up only once in the init( ) method and the reference saved in a class variable. The doGet( ) method will create a Bean1 instance and invoke the doIt( ) method on that bean. This was done to prevent repeated lookups of the home in the doIt( ) method.

The **init( )** method: The flow in this method is to:

- create an InitialContext object,
- obtain a reference to the Bean1Home by looking up "java:comp/env/ejb/TheBean1Home"
- narrow the returned object to the correct type,
- save the reference for use by the doGet( ) method callers.

The remainder of the code is for debugging and housekeeping. Here is the code segment.

```
public class JNDIServlet extends HttpServlet {

   private com.ibm.wsc.Bean1Home myHome = null;
   private javax.naming.InitialContext myCtx = null;

   public void init() throws javax.servlet.ServletException {

      try {
         System.out.println("JNDISevlet.init() Creating initial Context and "+
                           " looking up home.");
         myCtx = new InitialContext();
         java.lang.Object objHome =
            myCtx.lookup("java:comp/env/ejb/TheBean1Home");
         myHome = ((com.ibm.wsc.Bean1Home)
                  (javax.rmi.PortableRemoteObject.narrow(
                    objHome, com.ibm.wsc.Bean1Home.class)));
      } catch (NamingException ex) {
         System.out.println(
            "JNDISevlet.init() Error looking up JNDISessionHome,\n" +
             ex.getExplanation());
         ex.printStackTrace();
         throw new ServletException(ex.toString());
      } finally {
         try {
            if (myCtx != null) {
               myCtx.close();
               myCtx = null;
            }
         } catch (NamingException ex1) {
            System.out.println("JNDISevlet.init() Failed to close myCtx");
         }
      }
   }
```

The **doGet( )** method: The flow in this method is to:

- create an instance of Bean1, and
- invoke the doIt( ) method on that bean instance.

The remainder of the code is for debugging and housekeeping. Here is the code segment.

```
   public void doGet(
      javax.servlet.http.HttpServletRequest request,
      javax.servlet.http.HttpServletResponse response)
      throws javax.servlet.ServletException, java.io.IOException {
      try {
         Bean1 myBean = myHome.create();
         myBean.doIt();
      } catch (Exception e) {
         System.out.println("JNDISevlet.doGet() Accessing Bean1 error");
         e.printStackTrace();
         throw new ServletException(e.toString());
      }
   }
```

Bean1

This contains only one business method to be invoked by the servlet. Its only function is to drive a method on Bean2.

The **doIt( )** method: The flow in this method is to:

- create an InitialContext object,
- obtain a reference to the Bean2Home by looking up "java:comp/env/ejb/TheBean2Home",
- narrow the returned object to the correct type,
- create an instance of Bean2,
- invoke the doIt( ) method is driven.

The flow looks remarkably like the code in the JNDIServlet, only it resides in a single method. The remainder of the code is for debugging and housekeeping. Here is the code segment.

```
   public void doIt() throws javax.ejb.EJBException {
      System.out.println("Bean1.doit() Entering");
      InitialContext myCtx = null;
      Bean2Home myHome = null;
      try {
         System.out.println(
            "Bean1.doit() Creating initial Context and looking up Bean2home.");
         myCtx = new InitialContext();
         java.lang.Object objHome =
            myCtx.lookup("java:comp/env/ejb/TheBean2Home");
         myHome = (com.ibm.wsc.Bean2Home)
                  (javax.rmi.PortableRemoteObject.narrow(
                   objHome, com.ibm.wsc.Bean2Home.class)));
         System.out.println(
            "Bean1.doit() Creating instance of Bean2 and driving doit().");
         Bean2 myBean2 = myHome.create();
         myBean2.doIt();
      } catch (NamingException ex) {
         System.out.println(
            "Bean1.doit() Error looking up Bean2Home,\n" +
             ex.getExplanation());
         ex.printStackTrace();
         throw new EJBException(ex.toString());
      } catch (CreateException ex) {
         System.out.println("Bean1.doit() Error creating Bean2,\n" +
             ex.toString());
         ex.printStackTrace();
         throw new EJBException(ex.toString());
      } catch (RemoteException ex) {
         System.out.println("Bean1.doit() Error accessing Bean2,\n" +
             ex.toString());
         ex.printStackTrace();
         throw new EJBException(ex.toString());
      } finally {
         System.out.println("Bean1.doit() Exiting");
         try {
            if (myCtx != null) {
               myCtx.close();
               myCtx = null;
            }
         } catch (NamingException ex1) {
            System.out.println("Bean1.doit()Failed to close myCtx");
         }
      }
   }
```

Bean2

This bean's sole purpose is to be located by Bean1.

The doIt( ) method:

Below is the doIt( ) method on Bean2 EJB, which merely prints a message.

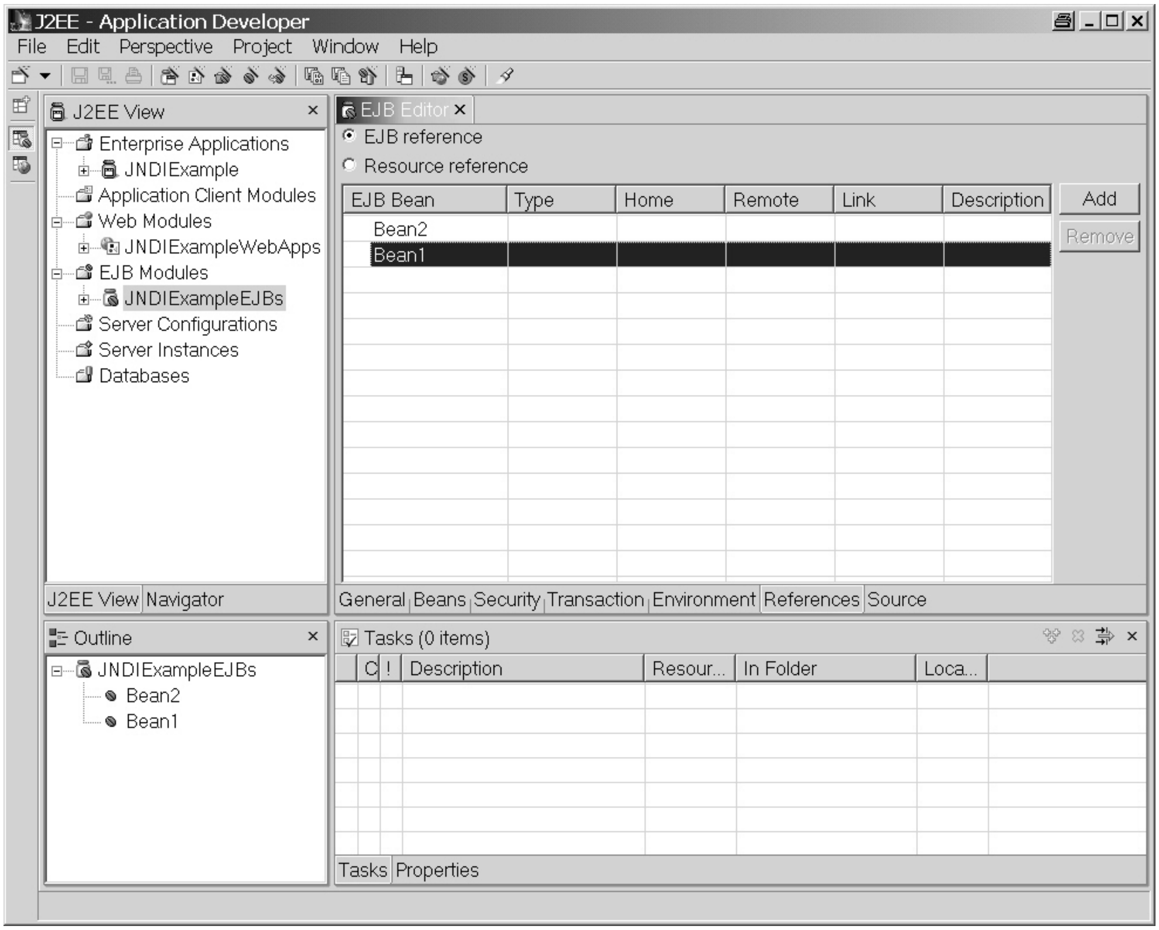
```
   public void doIt() throws javax.ejb.EJBException {
      System.out.println("Bean2.doit() Entering and Exiting");
   }
```

The key point from examining the above code snippets is that the application code does not contain the actual JNDI name of the object being looked up. However, there is a structure to the string used to lookup the bean. It will consists of a prefix of "java:comp/env/" and a suffix of the application developers choosing. In this example, we choose names that are meaningful to the developer.

In fact, there is nothing "magic" about the suffix: "ejb/TheBean1Home" or "ejb/TheBean2Home" which is appended to the "java:comp/env/" prefix. The EJB specification strongly recommends that the string begin with "ejb/"; but it could be something as complicated as "ejb/WSC/JNDI/Example/BeanOneHome" or as simple as "JoesBakedBean1". It is just a string that the developer and assembler must remember to declare as an EJB reference when creating the deployment descriptors for the EJB .jar file and the servlet .war file.

The "java:comp/env" prefix is indeed "magic". This prefix is the key to having the J2EE runtime intercede in the lookup of the desired object's home and provide the actual JNDI name rather than just pass the string across to the JNDI naming service. The actual JNDI name can be assigned at deployment time and thus the application doesn't have a need to know the actual name. This makes EJBs much more portable between deployment environments.

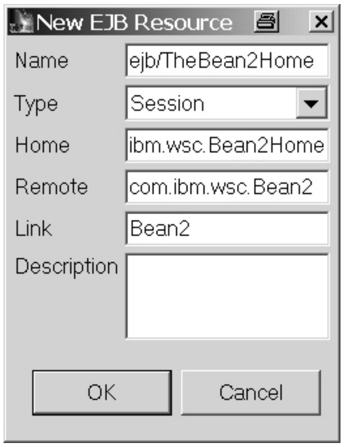
From a J2EE application construction perspective, it is necessary to declare the ejb-ref in the Web Application (in order for the JNDIServlet to find Bean1Home), and in the EJB module (for Bean1 to find Bean2Home). Turn to the J2EE view tab of the J2EE perspective. In the top left hand pane, select JNDIExampleEJBs → Open with → EJB Editor and then select the References tab to see the following:



● EJB reference
○ Resource reference

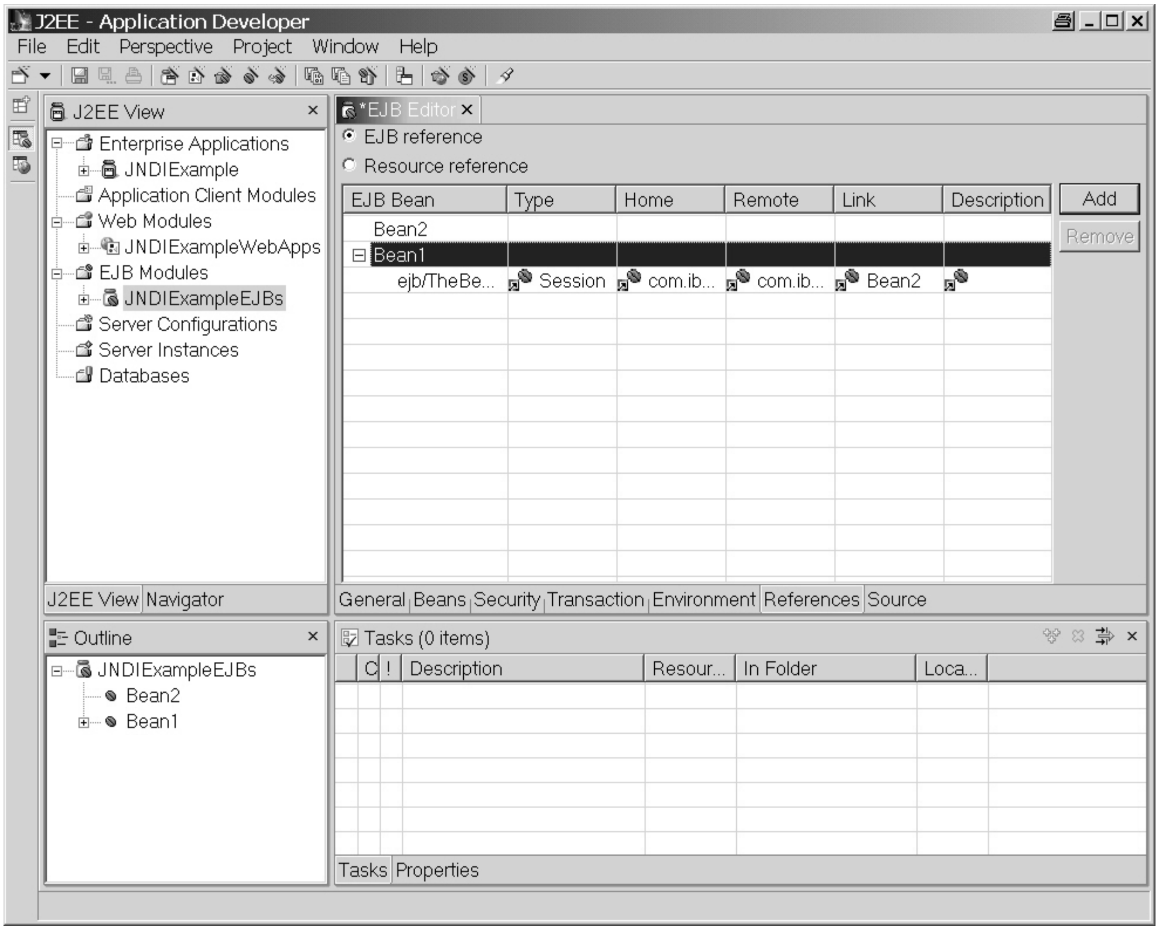
EJB Bean	Type	Home	Remote	Link	Description
Bean2					
Bean1					

Now, add an EJB Reference to Bean1 for Bean2. Select Bean1 and click on the "Add" button to get the following popup that you must input information (we have filled this in):



Name	ejb/TheBean2Home
Type	Session
Home	ibm.wsc.Bean2Home
Remote	com.ibm.wsc.Bean2
Link	Bean2
Description	

The Name value is "ejb/TheBean2Home". This is the suffix to the "java:comp/env/" part of the string, which is used to lookup the Bean2Home in the doIt( ) method in Bean1. (i.e. `java.lang.Object objHome = myCtx.lookup("java:comp/env/ejb/TheBean2Home");`). The bean Type, the Home, and Remote interface class names are also specified to aid the container at a later time. Since Bean2 is contained within this J2EE application, a Link value for Bean2 is specified. Later, this will cause the deployment tool to automatically associate the ejb-ref value to the correct JNDI name. Press the save button and you will see the ejb-ref you have created:

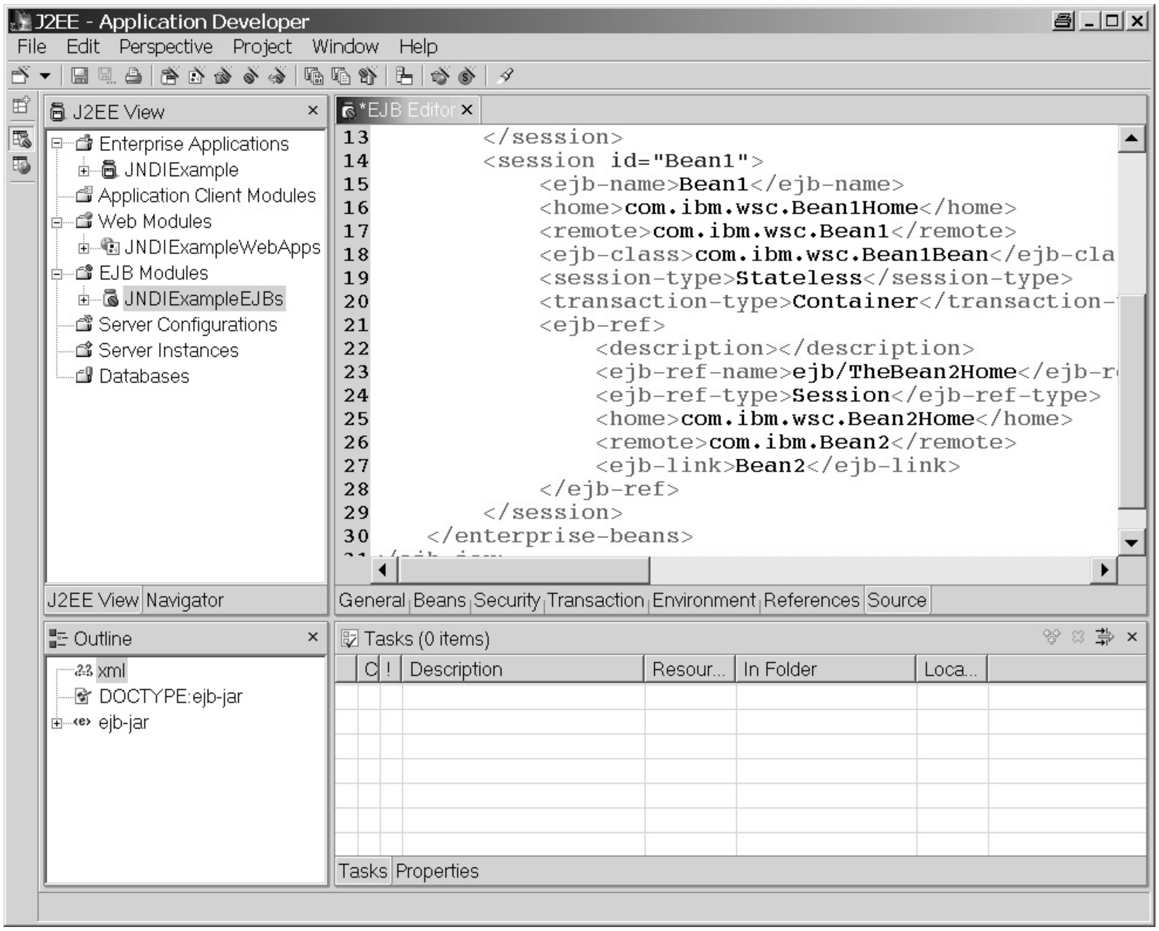


● EJB reference

○ Resource reference

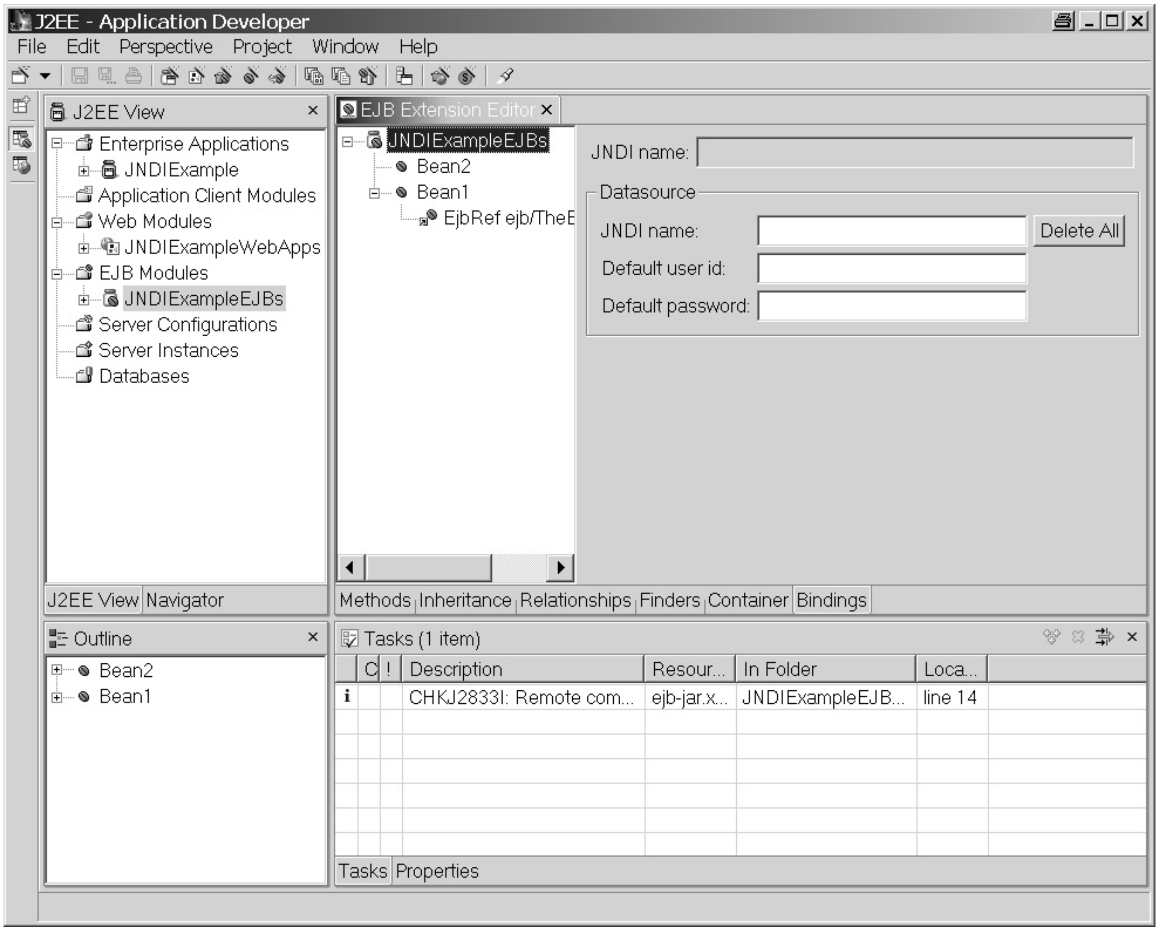
EJB Bean	Type	Home	Remote	Link	Description
Bean2					
Bean1					
ejb/TheBe...	Session	com.ib...	com.ib...	Bean2	

Select the "Source" tab, to see the actual ejb-jar.xml file contents the WSAD tooling just created:



```
        </session>
        <session id="Bean1">
            <ejb-name>Bean1</ejb-name>
            <home>com.ibm.wsc.Bean1Home</home>
            <remote>com.ibm.wsc.Bean1</remote>
            <ejb-class>com.ibm.wsc.Bean1Bean</ejb-cla
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-
            <ejb-ref>
                <description></description>
                <ejb-ref-name>ejb/TheBean2Home</ejb-r
                <ejb-ref-type>Session</ejb-ref-type>
                <home>com.ibm.wsc.Bean2Home</home>
                <remote>com.ibm.Bean2</remote>
                <ejb-link>Bean2</ejb-link>
            </ejb-ref>
        </session>
    </enterprise-beans>
```

Now save your work (CTRL-S) and close the EJB editor. Select JNDIExampleEJBs → Open with → EJB extension editor and turn to the Bindings tab. If you explode the 'plus' boxes, you should see:



	C	!	Description	Resour...	In Folder	Loca...
i			CHKJ2833I: Remote com...	ejb-jar.x...	JNDIExampleEJB...	line 14

The task is to assign JNDI names for the EJB homes and to associate the ejb-refs previously specified with the correct JNDI names. This assignment is done to allow testing in the WSAD environment. The JNDI name assignments in WSAD have nothing to do with installation of the J2EE application in the WebSphere for z/OS environment and the JNDI name assignment in that environment. The JNDI names that you assign in WSAD could be carried over into the WebSphere for z/OS environment, but probably shouldn't for reasons discussed later.

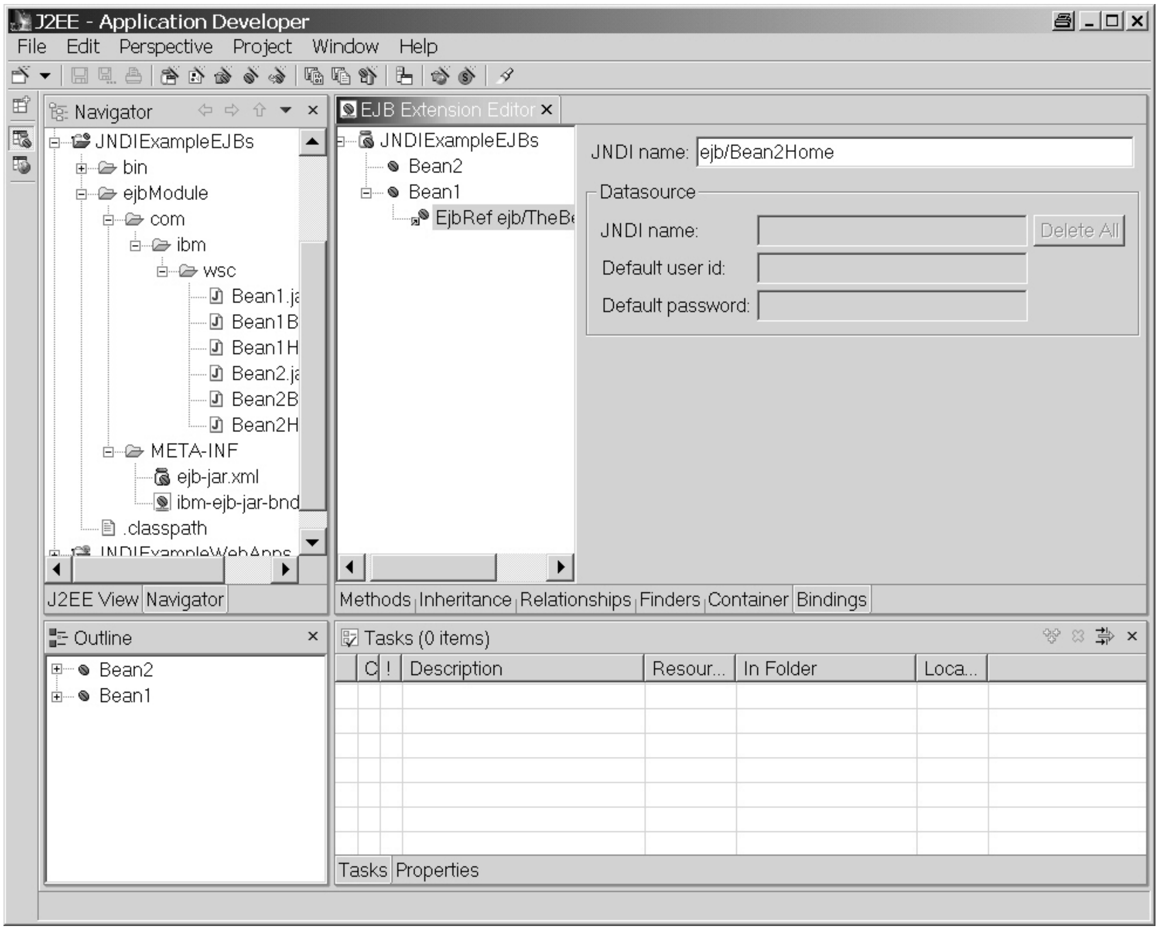
Perform the following steps to assign the JNDI names of the EJBs

1. Select Bean2 and assign a JNDI name for Bean2. In the JNDI name box enter: ejb/Bean2Home (erase the compound name that is there, WSAD seems to have trouble with compound names at least at my service level).
2. Select Bean1 and assign a JNDI for Bean1. In the JNDI name box enter: ejb/Bean1Home.

Perform the following step to associate the ejb-ref to the correct JNDI name.

1. Select the ejbRef/TheBean2Home object beneath Bean1 and in the JNDI name box enter the actual JNDI name for Bean2 (which you just set to ejb/Bean2Home in step 1).

At this point, you should see the following:

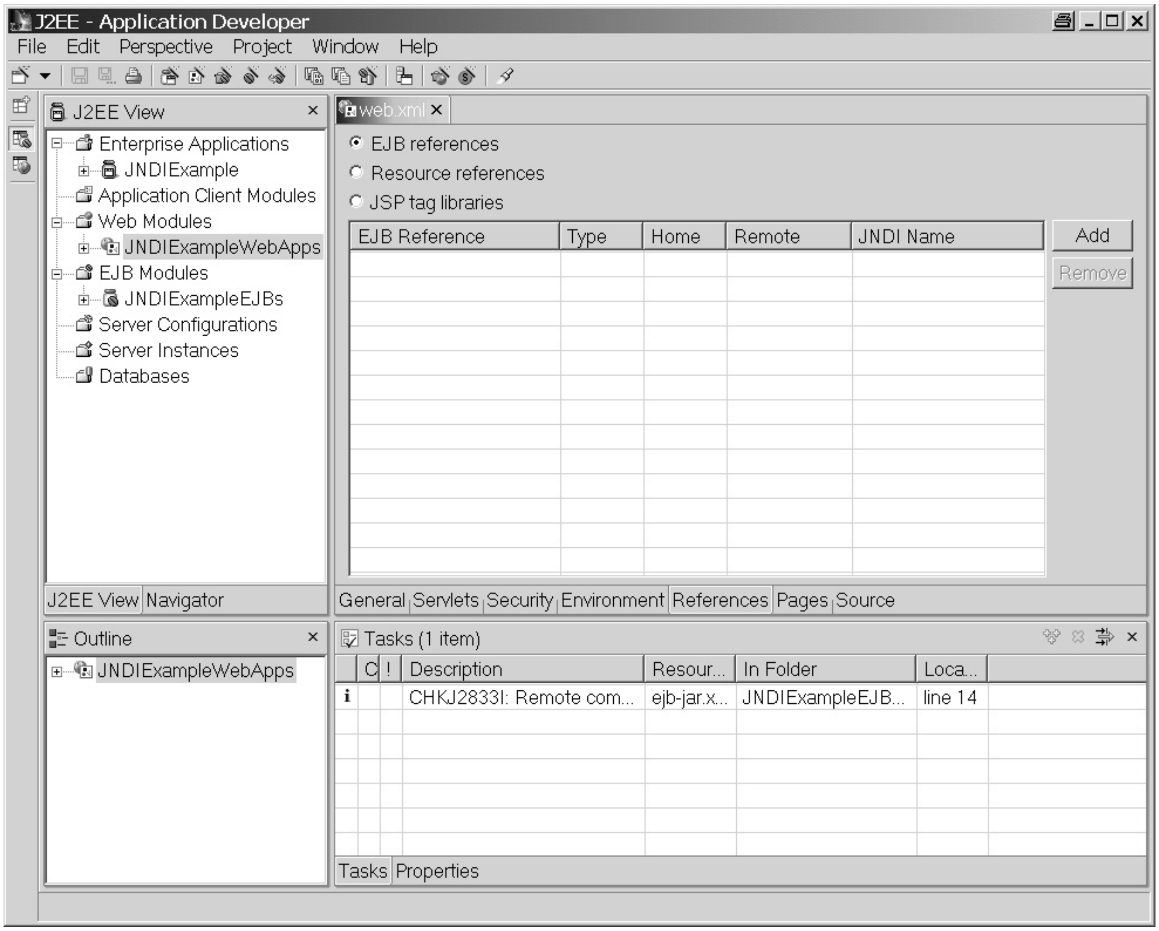


JNDI name: ejb/Bean2Home

Save (CRTL-S) your work and close the ejb extension editor.

Now, an ejb-ref must be defined for the web application and assigned to the Bean1 EJB JNDI name for testing.

In the top left hand pane, select JNDIExamplesWebApps → Open with →web.xml Editor, turn to the References tab and you should see:



● EJB references
○ Resource references
○ JSP tag libraries

EJB Reference	Type	Home	Remote	JNDI Name

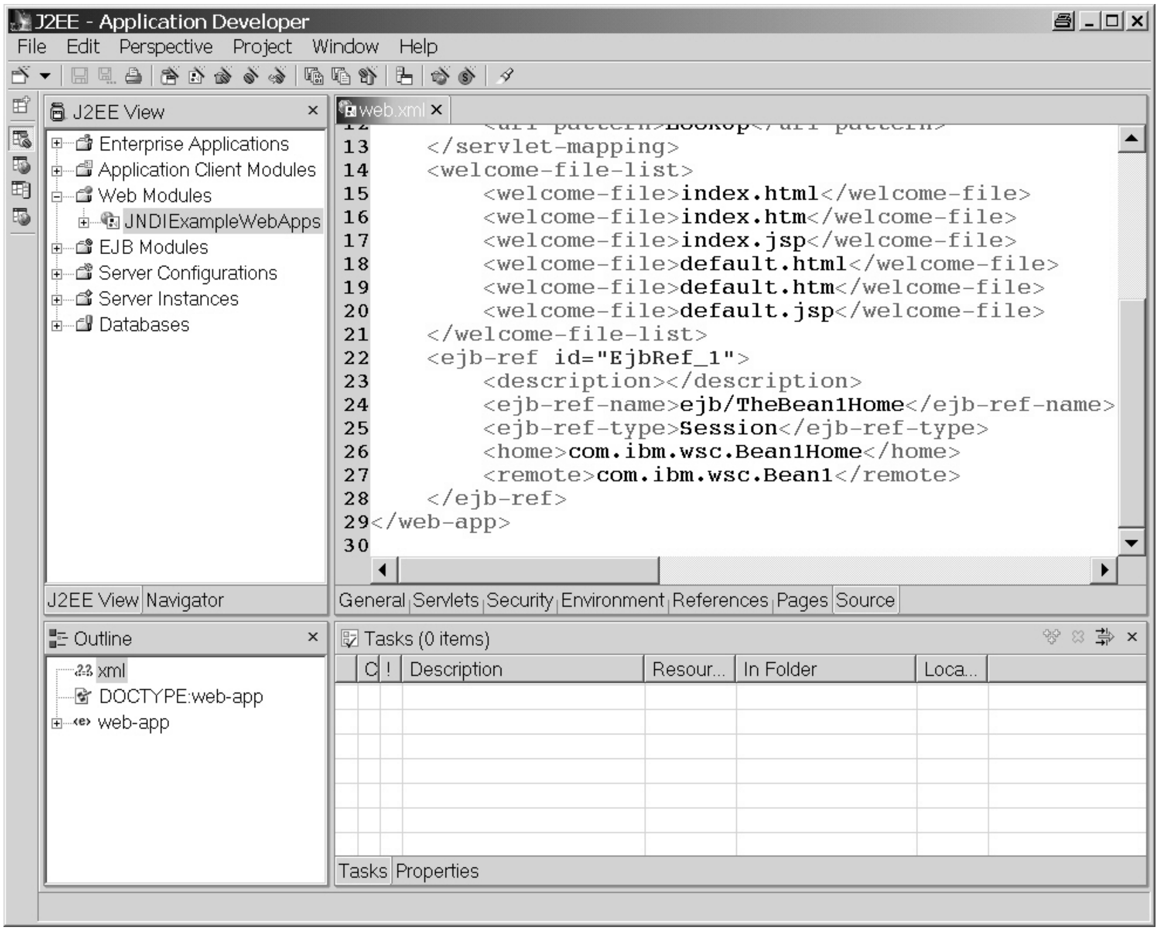
	C	!	Description	Resour...	In Folder	Loca...
i			CHKJ2833I: Remote com...	ejb-jar.x...	JNDIExampleEJB...	line 14

To add an EJB reference, make certain the "EJB References" option is selected, then click on the "Add" button. You now have the opportunity to create the ejb-ref needed by the servlet and associate the EJB reference to the real JNDI name for Bean2Home in this one panel by filling in the small boxes. The information you must type in is:

EJB Reference	= ejb/TheBean1Home
Type	= Session
Home	= com.ibm.wsc.Bean1Home
Remote	= com.ibm.wsc.Bean1
JNDI Name	= ejb/Bean1Home

This information is basically the same as the information you previously provided when declaring the ejb-ref on Bean1 for Bean2. There isn't a nice pop-up window.

Save this information (CTRL-S) and turn to the source tab to see the contents of the web.xml file just created by the WSAD tooling:



```
        <url-pattern>Lookup</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
        <welcome-file>default.html</welcome-file>
        <welcome-file>default.htm</welcome-file>
        <welcome-file>default.jsp</welcome-file>
    </welcome-file-list>
    <ejb-ref id="EjbRef_1">
        <description></description>
        <ejb-ref-name>ejb/TheBean1Home</ejb-ref-name>
        <ejb-ref-type>Session</ejb-ref-type>
        <home>com.ibm.wsc.Bean1Home</home>
        <remote>com.ibm.wsc.Bean1</remote>
    </ejb-ref>
</web-app>
```

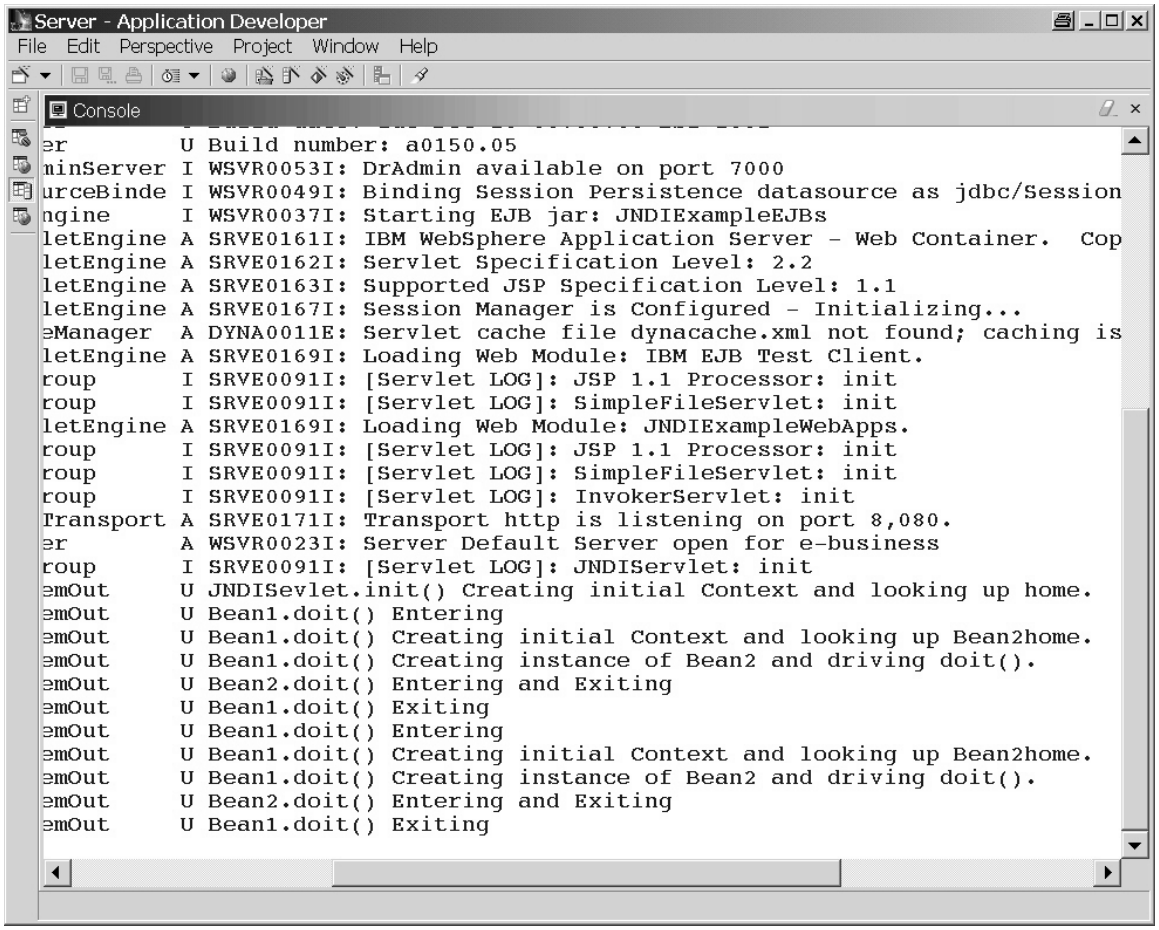
Note: WSAD does not allow you to specify the <ejb-link>Bean2</ejb-link> descriptor in the panel where ejb-ref descriptor information was supplied. For the application being constructed, it is desirable to have the servlet use the Bean1Home residing in the same ear file. While WSAD doesn't make this easy, you can set the ejb-link descriptor by hand editing the .xml file. Setting the ejb-link will actually be done as a part of the AAT for z/OS activity.

The web.xml file does not contain the JNDI name to ejb-ref association information needed by WSAD to test the application. The same is true for the ejb-jar.xml file as well. This information is maintained by WSAD in other .xmi files.

At this point, the application must have the EJB runtime supporting classes generated in order to be deployed in the WSAD test environment. In the Navigator tab of the J2EE perspective, select the JDNIExampleEJBs → Generate → Deploy and RMIC code. At the pop-up, press "Select all" and "Finish". Time will pass. When this processing completes, run "Validation" on each of the three projects.

To test the application in the WSAD test environment, in the upper left-hand pane, select JNDIExample → Run on Server. A server instance should start. Once the server starts, your application is ready to test. Press the button to open a web browser and enter the URL: http://localhost:8080/JNDI/LookUp and press "Go" twice.

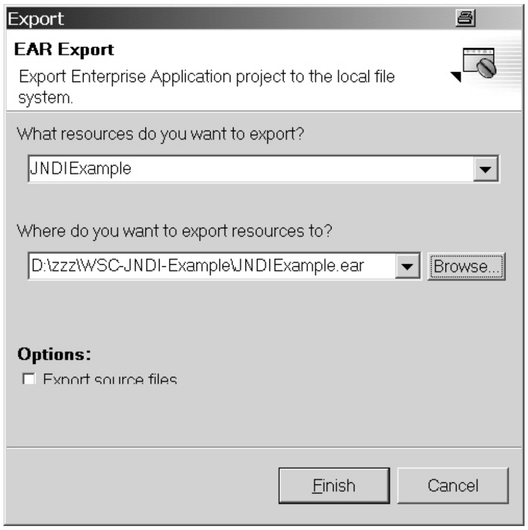
No results information will appear on the browser if everything is successful; however if you look at the console window you should see:


Server - Application Developer
File Edit Perspective Project Window Help
Console

```
er         U Build number: a0150.05
minServer  I WSVR0053I: DrAdmin available on port 7000
urceBinde  I WSVR0049I: Binding Session Persistence datasource as jdbc/Session
ngine      I WSVR0037I: Starting EJB jar: JNDIExampleEJBs
letEngine  A SRVE0161I: IBM WebSphere Application Server - Web Container.  Cop
letEngine  A SRVE0162I: Servlet Specification Level: 2.2
letEngine  A SRVE0163I: Supported JSP Specification Level: 1.1
letEngine  A SRVE0167I: Session Manager is Configured - Initializing...
eManager   A DYNA0011E: Servlet cache file dynacache.xml not found; caching is
letEngine  A SRVE0169I: Loading Web Module: IBM EJB Test Client.
roup       I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
roup       I SRVE0091I: [Servlet LOG]: SimpleFileServlet: init
letEngine  A SRVE0169I: Loading Web Module: JNDIExampleWebApps.
roup       I SRVE0091I: [Servlet LOG]: JSP 1.1 Processor: init
roup       I SRVE0091I: [Servlet LOG]: SimpleFileServlet: init
roup       I SRVE0091I: [Servlet LOG]: InvokerServlet: init
Transport  A SRVE0171I: Transport http is listening on port 8,080.
er         A WSVR0023I: Server Default Server open for e-business
roup       I SRVE0091I: [Servlet LOG]: JNDIServlet: init
emOut      U JNDISevlet.init() Creating initial Context and looking up home.
emOut      U Bean1.doit() Entering
emOut      U Bean1.doit() Creating initial Context and looking up Bean2home.
emOut      U Bean1.doit() Creating instance of Bean2 and driving doit().
emOut      U Bean2.doit() Entering and Exiting
emOut      U Bean1.doit() Exiting
emOut      U Bean1.doit() Entering
emOut      U Bean1.doit() Creating initial Context and looking up Bean2home.
emOut      U Bean1.doit() Creating instance of Bean2 and driving doit().
emOut      U Bean2.doit() Entering and Exiting
emOut      U Bean1.doit() Exiting
```

At this point, we are reasonably certain the application is properly constructed and does indeed work. We have demonstrated how to associate a java:comp/env/ejb/<value> to a JNDI name in the WSAD test environment. More importantly, the application is ready to be exported from WSAD and deployed into another J2EE server; WebSphere for z/OS V4.0.1.

WSAD development and testing are complete. To export the JNDIExample J2EE application as an .ear file do the following. Select JNDIExample → Export ear. You will be presented with the following popup where you specify the project to be exported and the name and location of the .ear file. For this example, the .ear file is named JNDIExample.ear and is placed in the d:\zzz\WSC-JNDI-Example directory of the workstation.



EAR Export

Export Enterprise Application project to the local file system.

What resources do you want to export?

JNDIExample

Where do you want to export resources to?

D:\zzz\WSC-JNDI-Example\JNDIExample.ear

Options:

□ Export source files

Press "Finish" to get an .ear file. When this processing is complete, close the WSAD tool.

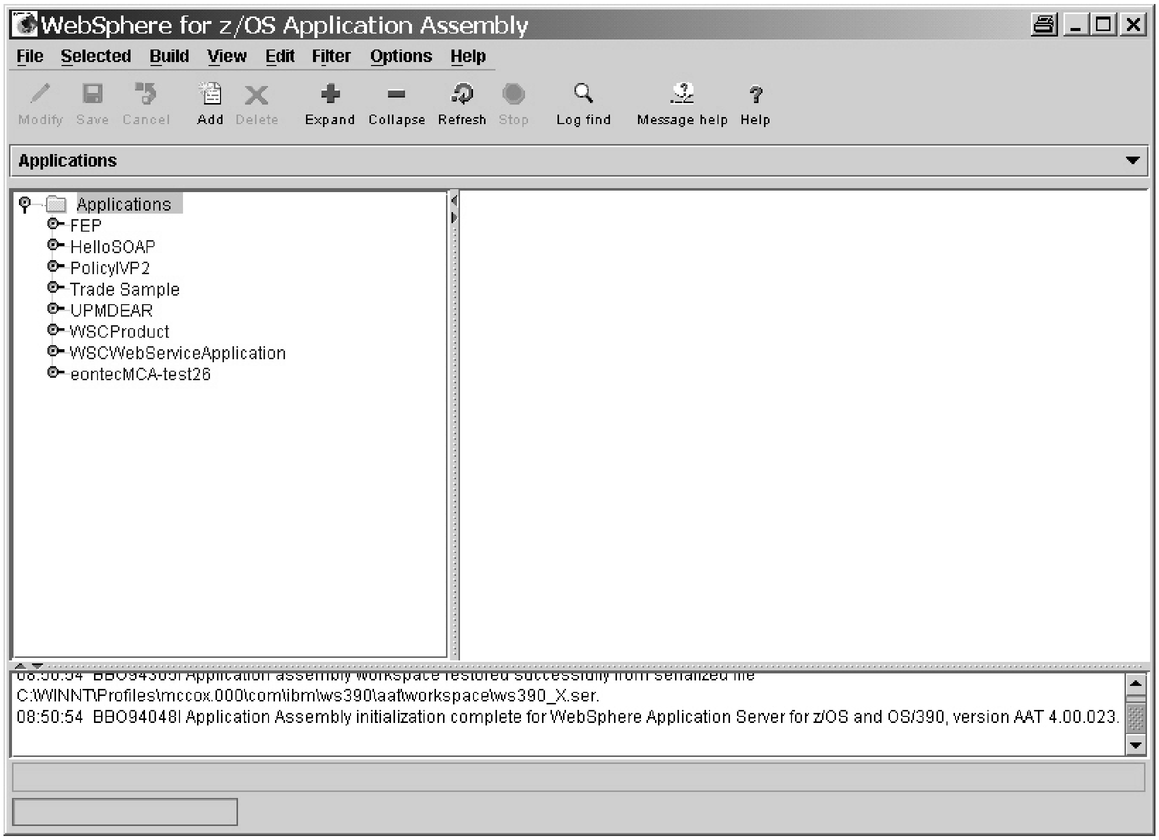
Part 2: Assemble the Simple Application.

Now that we have an .ear file, it is necessary to process it through the AAT for z/OS tooling. The AAT for z/OS actually does some code generation for Web Applications. The tooling also validates deployment settings as specified by the EJB specification. WSAD does not enforce some of the J2EE specification rules. For example, the transaction attributes on the session beans were not set and the Context Root for the Web Application should be prefixed with a slash "/". Additionally, we want to restrict the servlet to access Bean1, which resides in this .ear file. To do this we will add an ejb-link (The intent of this action is to reduce the options for the person installing the .ear file into a J2EE server).

Some of these tasks could have been done in WSAD, but weren't. For tutorial purposes, we will go through the process of taking the .ear file which was exported from WSAD without errors, and "fault our way" through the creation of a .ear file to be deployed in WAS for z/OS, correcting errors/ omissions along the way. If these deployment descriptors had been set correctly in WSAD, no changes to the ear file would be required.

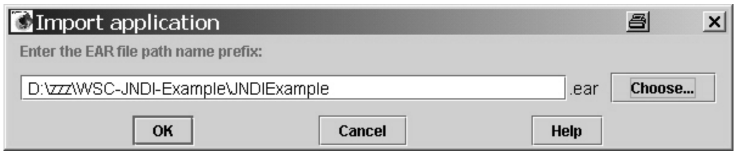
Note: The need for the AAT for z/OS to "re-process" the .ear file is a "point in time" solution that should be unnecessary in a future release of WebSphere for z/OS.

Start the AAT for z/OS tool and you should see the following:


WebSphere for z/OS Application Assembly
File Selected Build View Edit Filter Options Help
Modify Save Cancel Add Delete Expand Collapse Refresh Stop Log find Message help Help
Applications
Applications
FEP
HelloSOAP
PolicyIVP2
Trade Sample
UPMDEAR
WSCProduct
WSCWebServiceApplication
eontecMCA-test26

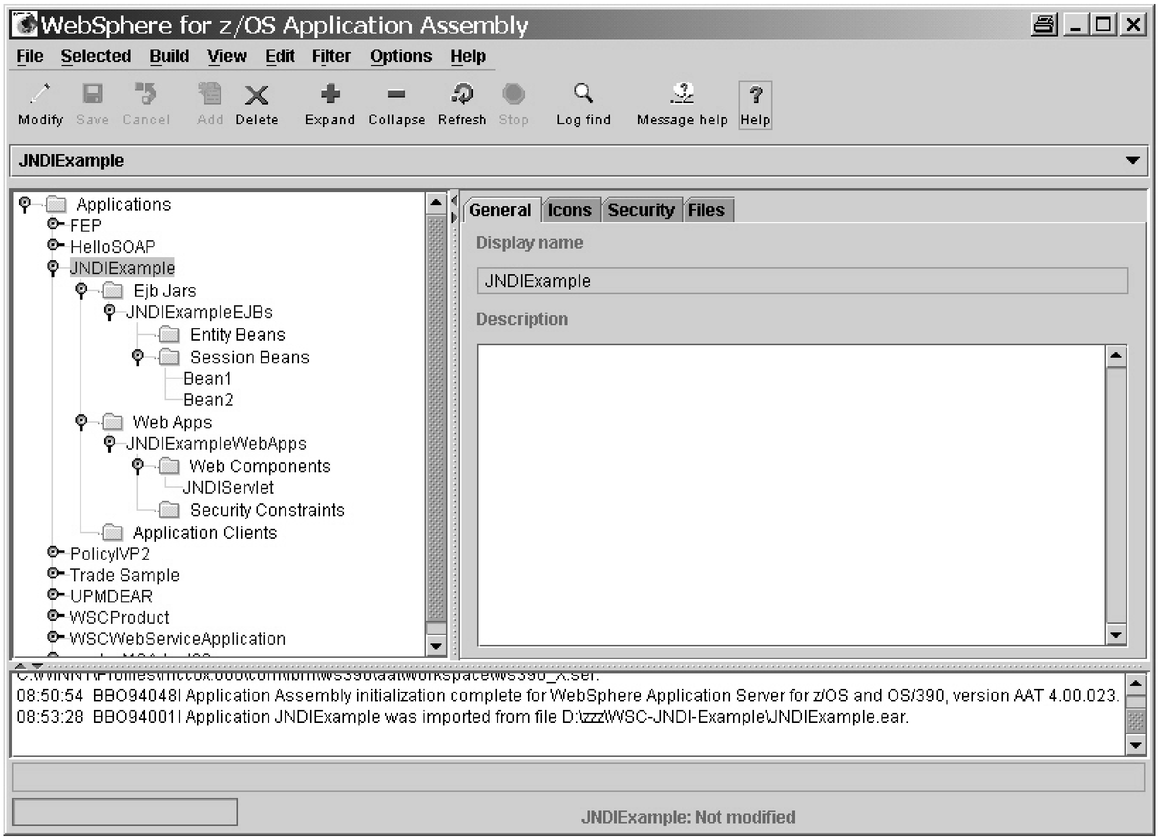
```
08:50:54  BBO94305I Application assembly workspace restored successfully from serialized file
C:\WINNT\Profiles\mccox.000\com\ibm\ws390\aat\workspace\ws390_X.ser.
08:50:54  BBO94048I Application Assembly initialization complete for WebSphere Application Server for z/OS and OS/390, version AAT 4.00.023.
```

To import the J2EE application packaged in the JNDIExample.ear file, select Applications → Import and enter the location and name of the .ear file, which was just exported from WSAD, in the popup window:



Enter the EAR file path name prefix: D:\zzz\WSC-JNDI-Example\JNDIExample .ear

Press "OK" to have the file imported into the AAT workspace. Once the application is imported, expand the JNDIExample application to see the following:



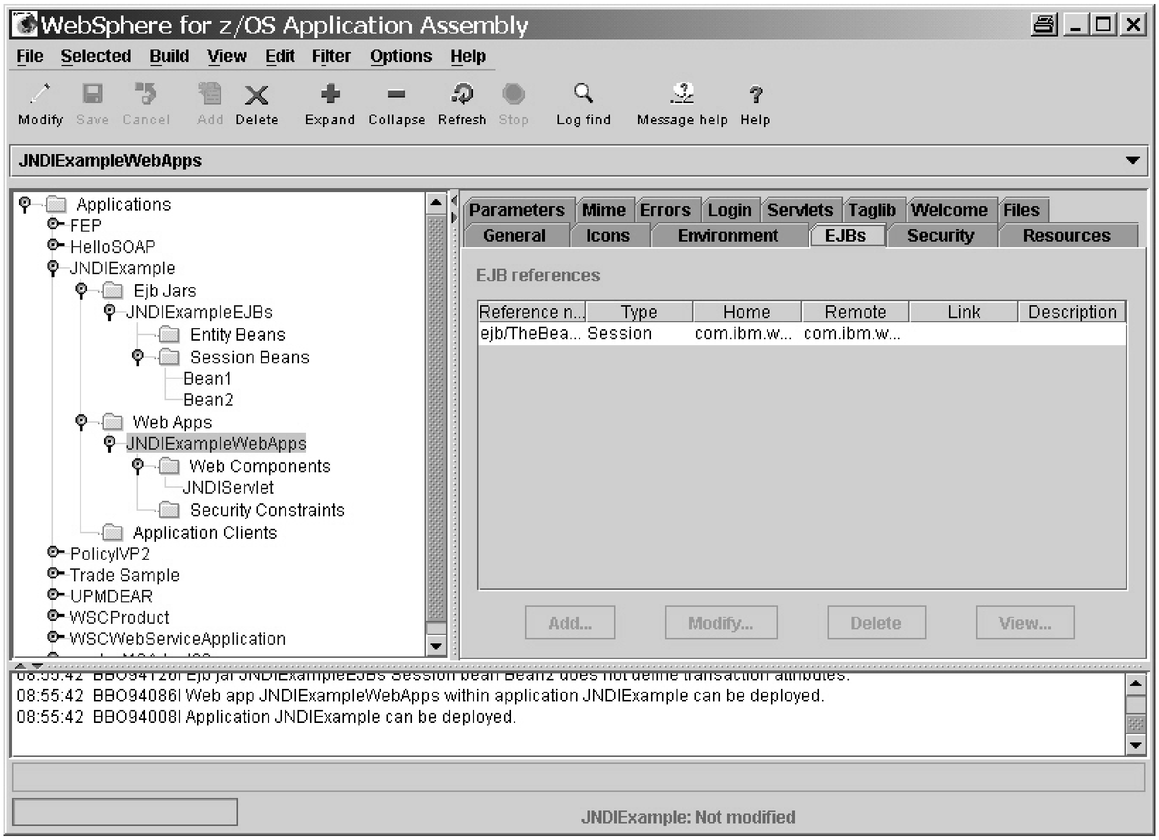
Display name: JNDIExample

Description:

```
08:50:54  BBO94048I Application Assembly initialization complete for WebSphere Application Server for z/OS and OS/390, version AAT 4.00.023.
08:53:28  BBO94001I Application JNDIExample was imported from file D:\zzz\WSC-JNDI-Example\JNDIExample.ear.
```

JNDIExample: Not modified

First, set the ejb-link on the ejb-reference for the web application. We want to restrict the deployer to using the Bean1 which is in this .ear file as opposed to a Bean1 residing in some other server. (This is the assignment that WSAD didn't allow when setting the ebj-ref on the web application). Select the JNDIExampleWebApps and go to the EJB tab. You will see:

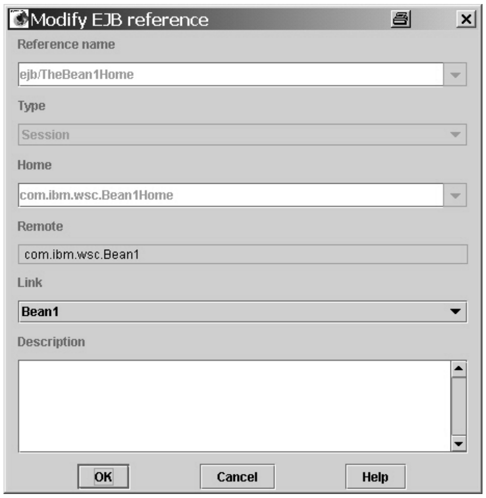

WebSphere for z/OS Application Assembly
File Selected Build View Edit Filter Options Help
Modify Save Cancel Add Delete Expand Collapse Refresh Stop Log find Message help Help
JNDIExampleWebApps
Applications
FEP
HelloSOAP
JNDIExample
Ejb Jars
JNDIExampleEJBs
Entity Beans
Session Beans
Bean1
Bean2
Web Apps
JNDIExampleWebApps
Web Components
JNDIServlet
Security Constraints
Application Clients
PolicyIVP2
Trade Sample
UPMDEAR
WSCProduct
WSCWebServiceApplication
Parameters Mime Errors Login Servlets Taglib Welcome Files
General Icons Environment EJBs Security Resources
Add... Modify... Delete View...
JNDIExample: Not modified

EJB references

Reference n...	Type	Home	Remote	Link	Description
ejb/TheBea...	Session	com.ibm.w...	com.ibm.w...		

```
08:55:42  BBO94120I Ejb jar JNDIExampleEJBs Session bean Bean2 does not define transaction attributes.
08:55:42  BBO94086I Web app JNDIExampleWebApps within application JNDIExample can be deployed.
08:55:42  BBO94008I Application JNDIExample can be deployed.
```

Click on the Modify Icon to allow changes to the application. Select the EJB reference of interest (i.e., ejb/TheBean1Home), press the "Modify.." button, and in the Link Selection list, choose Bean1 as in the following:



Reference name: ejb/TheBean1Home

Type: Session

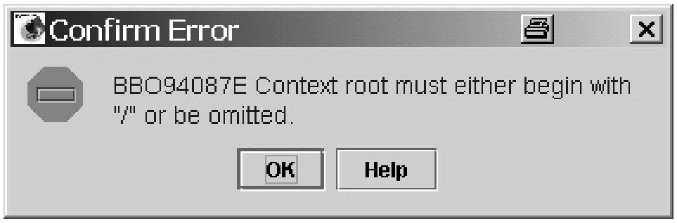
Home: com.ibm.wsc.Bean1Home

Remote: com.ibm.wsc.Bean1

Link: Bean1

Description:

Now, press "OK" and then press the Save Icon. At this point the AAT tool will remind you that the Context Root needs to be prefixed with a slash "/".

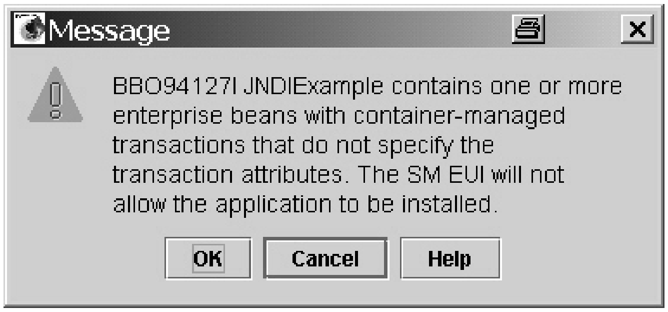


Confirm Error

BBO94087E Context root must either begin with "/" or be omitted.

Press the OK button and select the General tab. In the Context root box set the value: "/JNDI" instead of just "JNDI", then press the Save Icon.

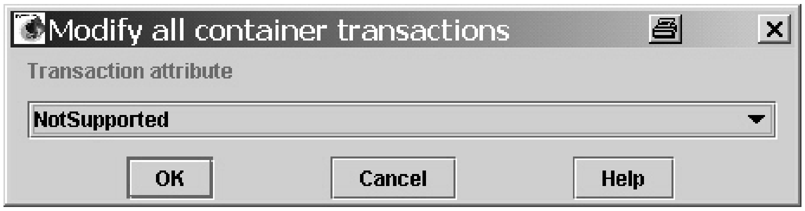
The application is ready to be deployed. On the task bar select Options → Enable EJBDeploy, making certain the box is checked. Now, select JNDIExample → Deploy. You will see the following popup:



Message

BBO94127I JNDIExample contains one or more enterprise beans with container-managed transactions that do not specify the transaction attributes. The SM EUI will not allow the application to be installed.

It seems that the transaction attributes were not set on either of the EJBs. Press "Cancel". Then select Bean1 and click on the Modify Icon. Turn to the transaction tab and press the "Modify all" button to get the following popup. In the window set the transaction attribute to NotSupported:



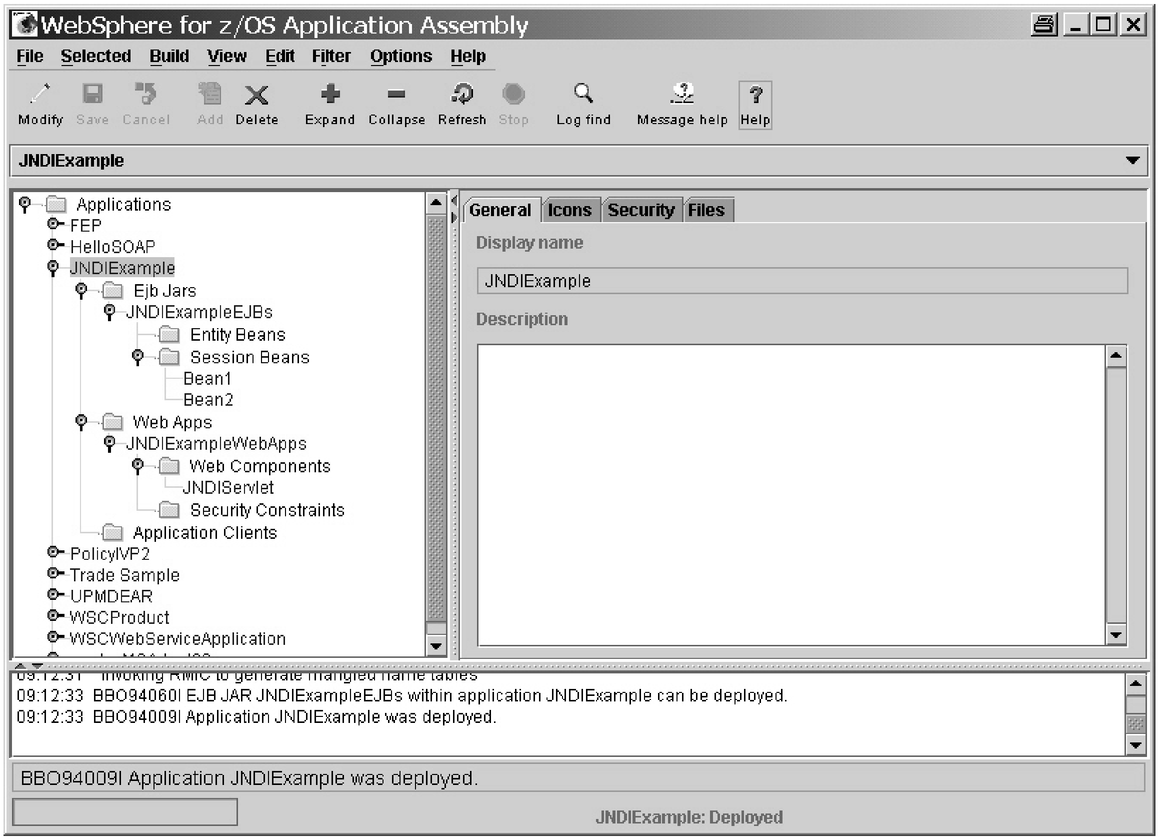
Modify all container transactions

Transaction attribute

NotSupported

Press the "OK" Button and the Save Icon. Then, repeat the process for Bean2.

Once again, select JNDIExample application → Deploy. This time the AAT will not find any errors and will deploy the application:



Display name

JNDIExample

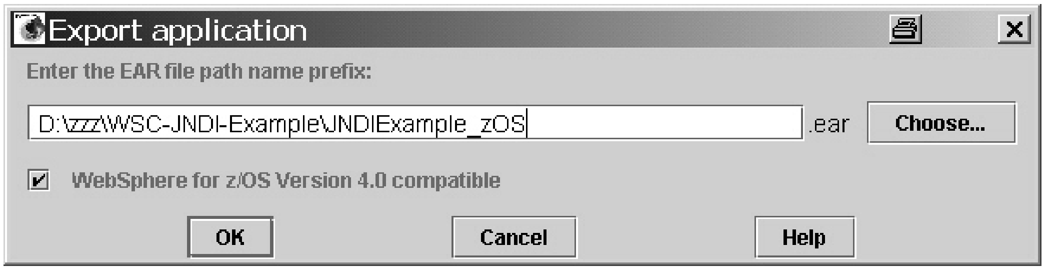
Description

```
09:12:31   Invoking RMIC to generate mangled name tables
09:12:33  BBO94060I EJB JAR JNDIExampleEJBs within application JNDIExample can be deployed.
09:12:33  BBO94009I Application JNDIExample was deployed.
```

BBO94009I Application JNDIExample was deployed.

JNDIExample: Deployed

Once the status bar indicates the application has been deployed, export the .ear file. Select JNDI Example application → Export. You will get the following popup:



Enter the EAR file path name prefix:

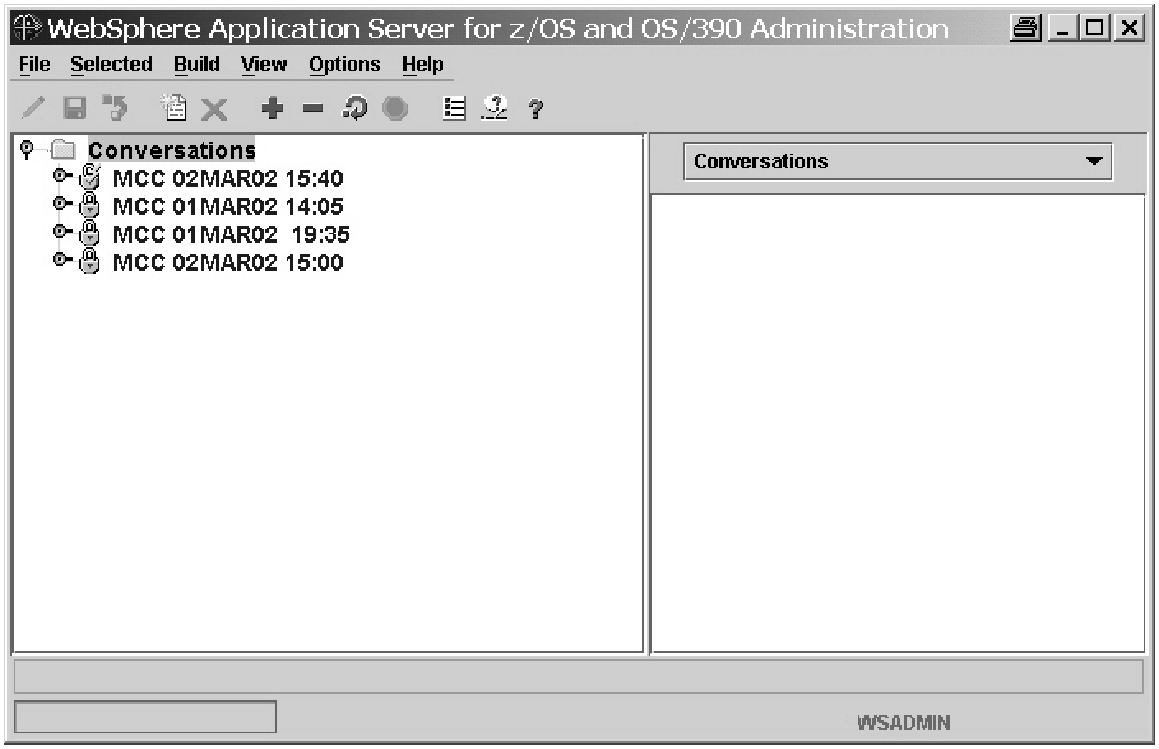
D:\zzz\WSC-JNDI-Example\JNDIExample_zOS .ear

☑ WebSphere for z/OS Version 4.0 compatible

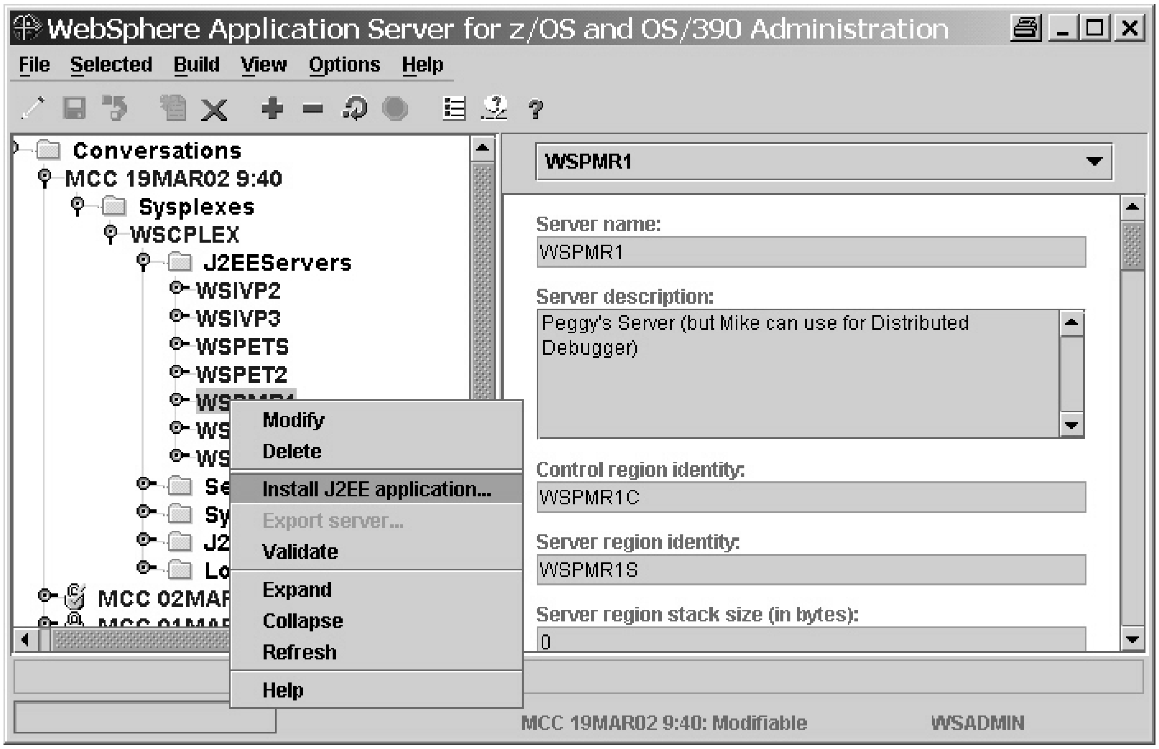
Name this exported .ear file: JNDIExample_zOS.ear, thus not altering the original .ear file exported from WSAD. Press the "OK" button and close the AAT for z/OS tool when the export process is complete.

Part 3: Install the Simple Application.

The final task is to install the JNDIExample_zOS.ear file into a WebSphere for z/OS server. Fortunately, a server (WSPMR1) is available in which to install the .ear file. Start the Administration application (SM EUI) specifying the correct host, and the userid/ password for the administrator. You should see the following:



Now, create a new conversation and select the WSPMR1 server as the target for the installation of the JNDIExample application. You should see something like the following:



Server name: WSPMR1

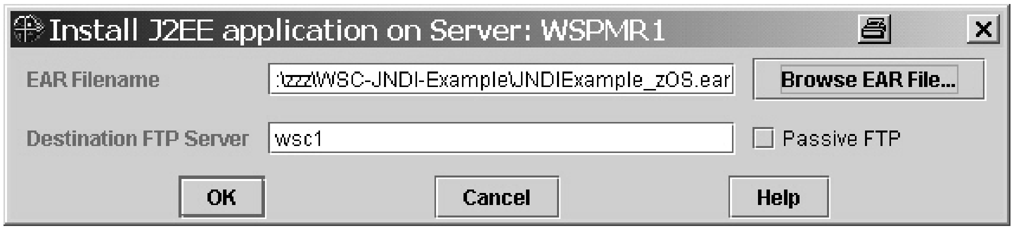
Server description: Peggy's Server (but Mike can use for Distributed Debugger)

Control region identity: WSPMR1C

Server region identity: WSPMR1S

Server region stack size (in bytes): 0

At the popup, enter the location and name of the .ear file to be installed (the .ear just exported from AAT for z/OS):

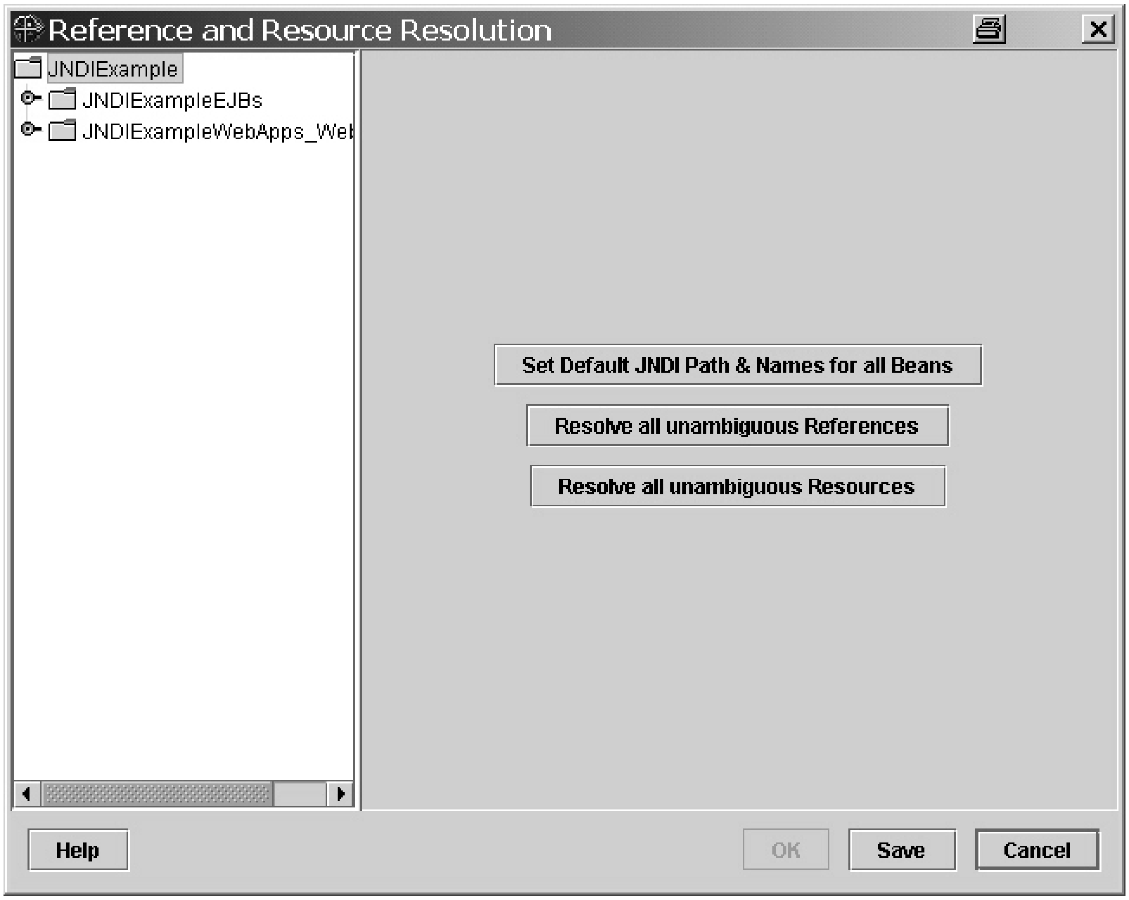

Install J2EE application on Server: WSPMR1

EAR Filename: :\zzz\WSC-JNDI-Example\JNDIExample_zOS.ear

Destination FTP Server: wsc1

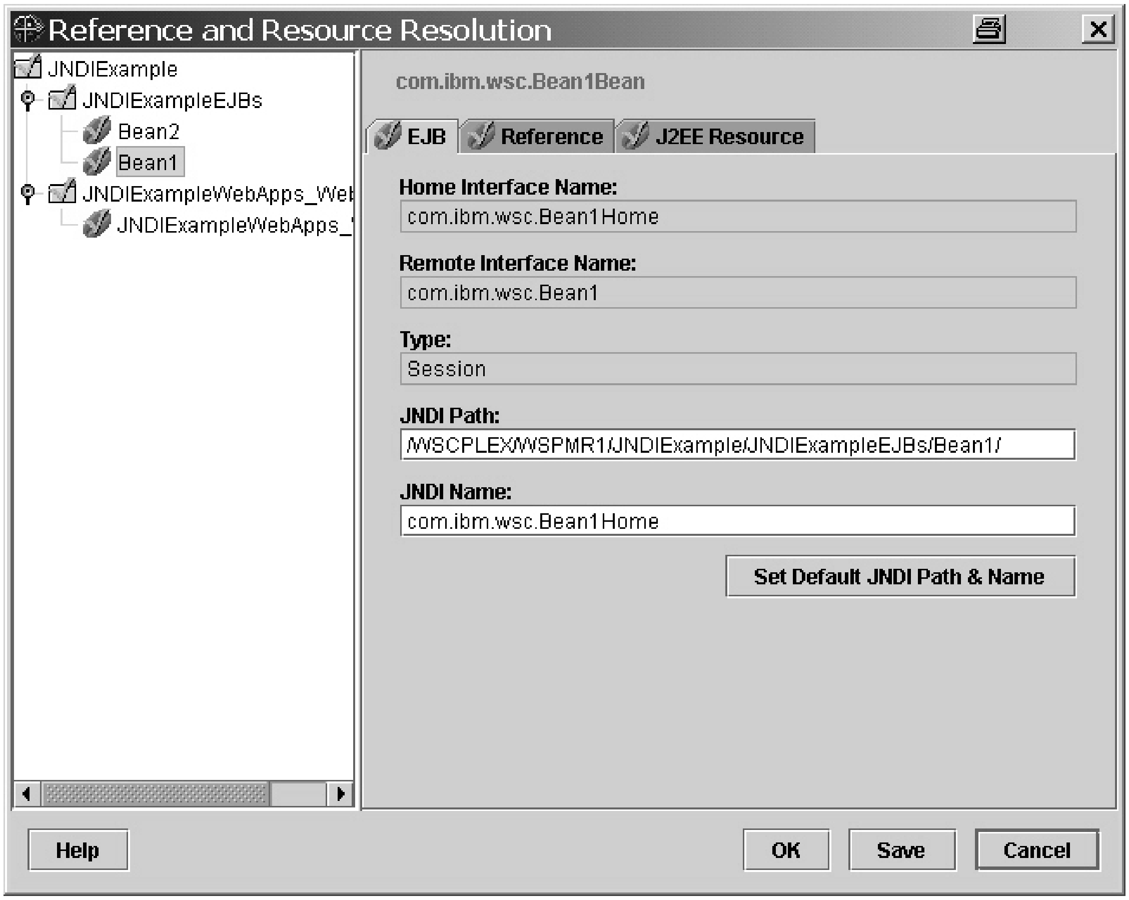
□ Passive FTP

Press "OK" and you will be presented with the following window:


Reference and Resource Resolution

JNDIExample
JNDIExampleEJBs
JNDIExampleWebApps_Wel
Set Default JNDI Path & Names for all Beans
Resolve all unambiguous References
Resolve all unambiguous Resources
Help
OK
Save
Cancel

There is no reason to choose a JNDI name for the components. Let the runtime do this task, thus insuring the JNDI names assigned to these objects are unique in the name space. The servlet and the EJBs have no need to know the actual JNDI names of the components. Only the container needs to know the actual names. So, press the "Set Default JNDI Path & Names for all Beans" button. This will cause the beans in the folders to be checked with a "green check mark" as in the following:



com.ibm.wsc.Bean1Bean

Home Interface Name: com.ibm.wsc.Bean1Home

Remote Interface Name: com.ibm.wsc.Bean1

Type: Session

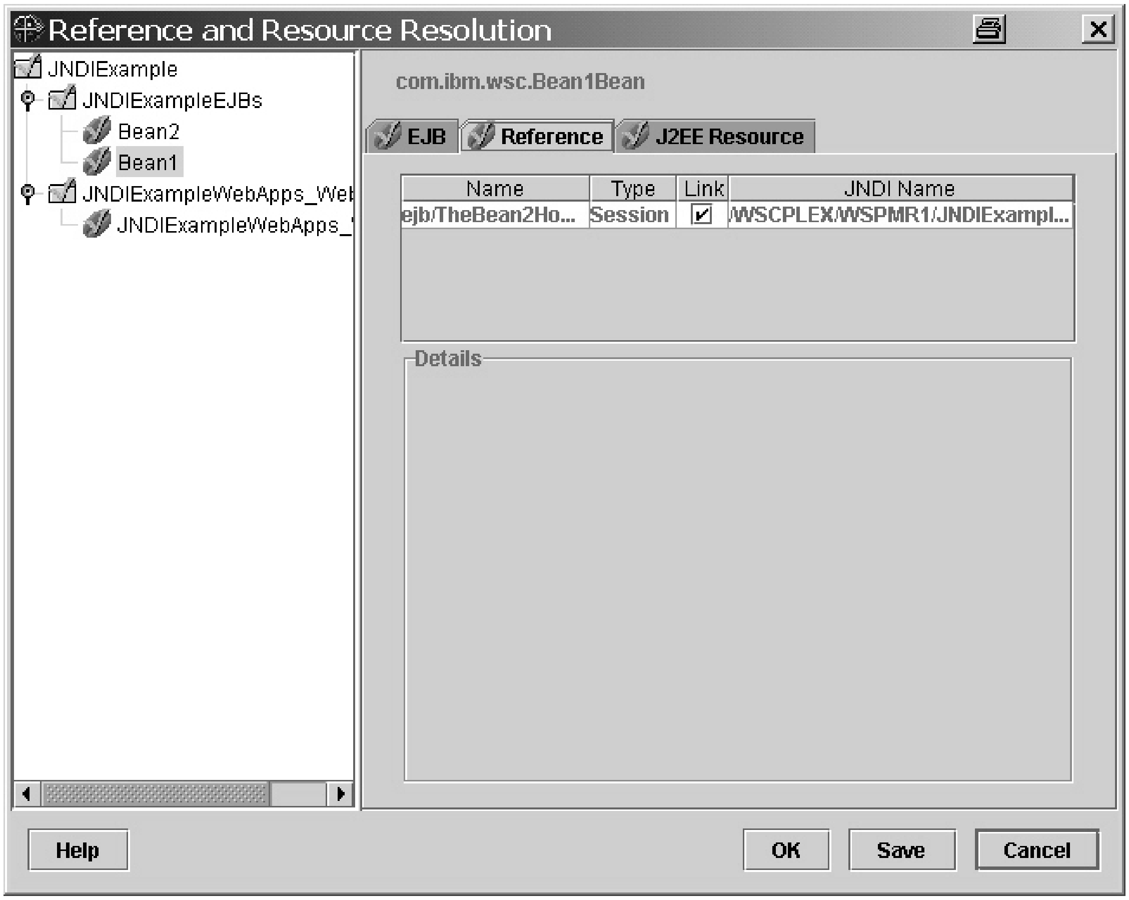
JNDI Path: /WSCPLEX/WSPMR1/JNDIExample/JNDIExampleEJBs/Bean1/

JNDI Name: com.ibm.wsc.Bean1Home

Notice the default JNDI name that was chosen for the Bean1Home. This is the convention used in WebSphere for z/OS to define a name that is unique to this sysplex, server, application, module, and component. It is comprised of the following parts:

The Sysplex Name	- WSCPLEX
The Server Name	- WSPMR1
The Application Name	- JNDIExample
The Module Name	- JNDIExampleEJBs
The Component Name	- Bean1
The Home Interface Name	- com.ibm.wsc.Bean1Home

Turn to the reference tab for Bean1 and you will see the following:



com.ibm.wsc.Bean1Bean

Name	Type	Link	JNDI Name
ejb/TheBean2Ho...	Session	☑	/WSCPLEX/WSPMR1/JNDIExampl...

Since the ejb-link was specified, the Administration tool associated the ejb-ref value (i.e., ejb/TheBean2Home) with the correct JNDI name for the referenced object (i.e., /WSCPLEX/WSPMR1/JNDIExample/JNDIExampleEJBs/Bean2/ com.ibm.wsc.Bean2Home) without prompting the administrator for input.

Had ejb-link not been specified, then the administrator would have had to choose the correct JNDI name. The administrator could choose a "Bean2" which is installed in this server or a different server.

Once all the JNDI names, EJB references and resource references are resolved, the application can be installed in the WebSphere for z/OS server. At this point, press the "OK" button to allow the FTP of the file: JNDIExample_zOS_resolved.ear, containing the resolved J2EE application, to the WebSphere for z/OS server. You will see the following popup:

Note: The original input .ear file, used as input for the deployment process, is not modified unless you press the "Save" button and specify the original .ear file name. A "copy" of the ear file is created on the workstation for transmission to the WebSphere for z/OS server.

When this process is complete, you will see message BBOU0470I in the status bar of the administration tool indicating the process is complete. For this conversation, perform the Commit, Complete → All tasks, and Activate sequence of events to cause the application to be installed in the WSPMR1 server. If the server is active, it will be terminated and restarted allowing the naming registration to occur. At this point, the application is ready to use. If the server was not already started, start the server manually to cause the naming registration to occur.

Once the server has been started, open the web browser of your choice and enter the URL for the servlet that has just been installed. In this environment, the URL is: http://wsc1.washington.ibm.com:8887/JNDI/LookUp (The context root and servlet mapping value are specified in the application. The host and port number are unique to this installation.)

When you press enter, the browser will return an error indicating no response was available. This is expected, as our Simple application does not return any output to the browser. Now look at the SYSPRINT dataset for the server address space for your server (i.e., WSPMR1S) and you will see:

```
JNDISevlet.init() Creating initial Context and looking up home.
Bean1.doit() Entering
Bean1.doit() Creating initial Context and looking up Bean2home.
Bean1.doit() Creating instance of Bean2 and driving doit().
Bean2.doit() Entering and Exiting
Bean1.doit() Exiting
```

Not surprisingly, the application works the exact same way in WebSphere for z/OS as it did in the WSAD test environment. Moreover, the EJBs have radically different JNDI names, AND the application has been deployed in two servers without the application developer needing to change any code to lookup the EJB homes. This same .ear file could be installed in another server on this WebSphere for z/OS node without change and both would work equally well and independent of each other.

An interesting activity would be to remove the ejb-link from the web application's deployment descriptor and install the modified .ear file into a second WebSphere for z/OS server (i.e., WSIVP2). The deployer would then be faced with the choice of specifying a JNDI name for Bean1Home for server WSPMR1 or WSIVP2.

Summary:

The process of building, testing and deploying a J2EE 1.2 application which takes advantage of the container environment naming context (ENC) and the java:comp/env construct to locate EJB homes is quite doable. The WSAD development and test environment supports the use of this technique. Using "java:comp/env/" removes the need to hardcode the name of the EJB Homes in the application, or to provide .properties files which can be interrogated by the application to locate other EJBs. The applications become much more portable among application servers using this technique.

End of Document