March 26, 2002

The Name Game: WebSphere z/OS JNDI Naming Concepts

Kenneth J. Muckenhaupt
IBM Design Center for e-transaction processing
kenjm@us.ibm.com

Table of Contents

Acknowledgments

The following people provided valuable technical and editorial input during the writing of this technical paper:

Don Bagwell, IBM Washington Systems Center
Mike Cox, IBM Washington Systems Center
John Gates, IBM WebSphere Enablement Team - z/OS and OS/390
Andrew Mauer, IBM WebSphere z/OS Development
Hong Min, IBM Design Center for e-transaction processing
Bart Tague, IBM Design Center for e-transaction processing
Holger Wunderlich, IBM International Technical Service Organization (ITSO)

Purpose of this Technical Paper

Java® Naming and Directory Interface™ (JNDI) naming concepts can be one of the most confusing aspects of developing or porting Java 2 Enterprise Edition (J2EE) applications on the WebSphere® z/OS™ platform. This technical paper will dispel some of the mystery surrounding J2EE JNDI naming concepts by

- *Providing an overview of JNDI namespace principles*
- *Describing how the namespace is implemented in WebSphere z/OS*
- *Addressing some common naming problems*

What Is A JNDI Namespace?

A JNDI namespace can be thought of as a specialized environment which identifies and describes the attributes of an application's Enterprise JavaBeans™ (EJBs). Within this specialized environment are the names by which EJBs are referenced by other code within an application. Typically, this other "code" is servlets and other EJBs that make up a J2EE application.

According to the *Enterprise JavaBeans Specification, v1.1,* every EJB has an associated environment that is declared in the EJB's **deployment descriptor**. Recall that an EJB's deployment descriptor is a special "profile" that describes the attributes of an EJB. This descriptive information, also known as "meta data," is used by application assembly and deployment tools such as the IBM® Application Assembly Tool (AAT) and Systems Management Extended User Interface (SMEUI) for z/OS to install EJBs into a container. Furthermore, the container uses the deployment descriptor meta data to manage an EJB during runtime. Since the environment description is outside of the EJB's source code, an EJB can be customized without ever changing the EJB's code. Therefore, the platform transparency of EJBs can be maintained while only the deployment information needs to be altered for each target platform.

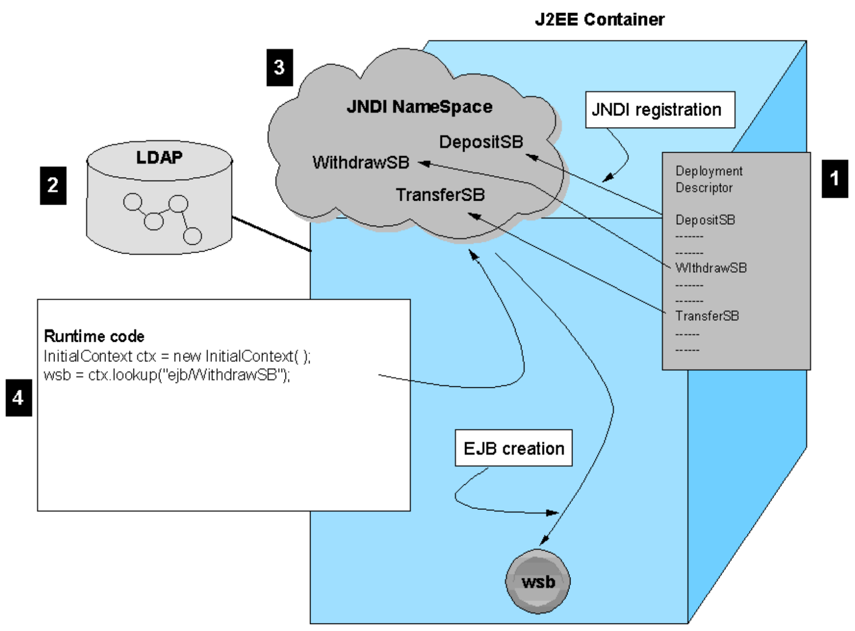
JNDI Naming Under WebSphere z/OS

So what does all this have to do with the JNDI namespace? One of the attributes in an EJB's deployment descriptor identifies the name of the EJB itself. The name of an EJB is used by application code to find or look up an EJB in a namespace. When a deployment tool installs an EJB into a WebSphere z/OS container, it gathers information about each EJB in an application, and determines where in the JNDI namespace the home for that bean should be stored. Then, as part of the application installation process, references to the EJB homes are bound into the JNDI namespace with the supplied name. Since WebSphere z/OS utilizes an LDAP database as the backing store for its JNDI implementation, an LDAP entry is created for the home references. Therefore, when an application such as a servlet or an access bean wants to create an instance of an EJB, it simply looks up the name of the desired EJB in the namespace created by the deployment process.

As shown in Figure 1, JNDI naming under WebSphere z/OS can be summarized with following fundamentals:

1. All EJBs have a deployment descriptor that, among other things, contains the name of the EJB.
2. During EJB deployment into a server, the SMEUI determines the name under which the home reference should be bound in the JNDI namespace. This namespace is backed in LDAP.
3. A global environment or **initial context** is the required starting point for locating EJBs in a JNDI namespace.
4. During bean development, if an EJB developer references an external EJB in their runtime code, then they must make that evident by defining an <ejb-ref> tag with /ejb/*home-name* in the EJB's deployment descriptor. This is the string that gets passed to the lookup( ) method to locate an external EJB under the java:comp JNDI context.

Figure 1. JNDI Namespace Fundamentals



Creating EJB Instances During Runtime

To access an EJB's registered deployment descriptor information, an application must first create an instance of the initial context. Once the initial context is obtained, then the application can look up the home or factory used to create an instance of the target EJB.

As you can see in Figure 1, the J2EE container's scope of control encompasses the deployment descriptor, the JNDI namespace, the application, and the instance of the target EJB. When an EJB is first accessed, for example when the first method is driven on an EJB, the container loads the EJB's meta data and establishes its java:comp namespace.

Creating EJB Instances With The Java: Logical Reference

WebSphere z/OS complies with the Enterprise JavaBeans Specification, v1.1 by providing a JNDI naming implementation based on the Lightweight Directory Access Protocol (LDAP). LDAP provides a directory structure for efficiently locating names in a namespace. When you deploy a J2EE application (servlets and EJBs) into a WebSphere z/OS container with the SMEUI, the Systems Management and Naming components of the WebSphere z/OS runtime work together to register the EJB home instances into the namespace. The registration process uses the deployment descriptor located in the EAR file to generate an indirect object reference (IOR) for the home interface which is stored in the LDAP database for each EJB.

Even though all EJB home references are stored in LDAP under WebSphere z/OS, EJB providers can designate a logical means for creating EJB instances during runtime. The **java:** name for a home is a logical name which an EJB provider specifies during runtime when they desire to create instances of a certain type of bean. Therefore, the EJB deployer must map the logical home to the physical home so that at runtime the lookup succeeds. Since EJB providers at development time do not know where in the namespace the home reference is going to be found, they can use the <ejb-ref> XML tag in the ejb-jar.xml file to declare that the EJB contains runtime code that is going to lookup and use a home. Therefore, the logical notation, java:comp/env/ejb*/home-name,* is passed on the lookup method call. To ensure a successful lookup, the deployer (the person responsible for creating the server with the SMEUI) must make sure that the declared java:comp name returns the EJB's home reference wherever it is located. The WebSphere z/OS container is ultimately responsible for determining how to return the home reference. You should understand that the java: namespace is a container-use only namespace that provides quick retrieval of important bean related objects during the container managed life cycle of an EJB.

Anatomy Of A WebSphere z/OS LDAP Entry

The fully-qualified JNDI namespace reference for the WithdrawSB EJB shown in Figure 1 could be represented by the following LDAP entry:

ibm-wsnName=sb, ibm-wsnName=Withdraw, ibm-wsnName=BankApp, ibm-wsnName=sample, ibm-wsnName=pok, ibm-wsnName=legacyRoot, ibm-wsnName=PELPLEXT, ibm-wsnName=domainRoots, ibm-wsnTree=t1, o=wasnaming,c=us

ibm-wsnentrytype: IORLeaf

ibm-wsnname: WithdrawSB

```
corbaior:IOR:0000000000000040524d493a646b2e64616e736b652e63722e6c6561646170706c2e70726
f62652e73622e50726f62655342486f6d653a30303030303030303030303030303000000000249424d0
f0000020400010000000000038d7f2f04040404040d7c5d3d7d3c5e7e301000000000000008c4c1c5d4d6d5
f0f100000011d7f2f0c24bc5e3d74bc9c2d44bc3d6d400000000000000175000000b4d6d9c2d200000002
c4d5e2d2e2d9e500b70216a75e04898e0000081400000140c0a8008c020000000000000000000040d9d4
c97a84924b848195a292854b83994b93858184819797934b97999682854ba2824bd799968285e2c2c896
94857af0f0f0f0f0f0f0f0f0f0f0f0f0f0f00000000003d84924b848195a292854b83994b9385818481979793
4b97999682854ba2824bc5d1e2d9859496a385e2a381a3859385a2a2d799968285e2c2c8969485000000
00000000b90000000000000000100000000000000000000000000000000000000009db7057b6cd73998a700
00086000000017c0a8008cb7057b6dd236b808000000860000000017c0a8008cb7057b6dd89b062a00000
86000000017c0a8008cc2d5c9c40000000100000001000000010000000db7057b6cd73998a7000008600
0000017c0a8008cb7057b6dd236b808000000860000000017c0a8008cb7057b6dd89b062a000008600000
0017c0a8008c6d6d88969485d686c8969485a2000000001300090000000300000000000000008000000004
9424d0049424d0400000000700050001020000000000000010000001c000000001002041700000001100204
171002041700000001100204170000000000000200000101000000000115032304 22e4554502e49424d2e4
34f4d000015b300000175000000b4d6d9c2d200000002c4d5e2d2e2d9e500b70216a75e04898e0000081
400000140c0a8008c020000000000000000000040d9d4c97a84924b848195a292854b83994b938581848
19797934b97999682854ba2824bd799968285e2c2c89694857af0f0f0f0f0f0f0f0f0f0f0f0f0f0f00000000
03d84924b848195a292854b83994b93858184819797934b97999682854ba2824bc5d1e2d9859496a385e
2a381a3859385a2a2d799968285e2c2c8969485000000000000000b9000000000000000010000000000000
000000000000000000000009db7057b6cd73998a70000086000000017c0a8008cb7057b6dd236b80800
00086000000017c0a8008cb7057b6dd89b062a0000086000000017c0a8008cc2d5c9c4000000001000000
01000000010000000db7057b6cd73998a70000086000000017c0a8008cb7057b6dd236b8080000086000
000017c0a8008cb7057b6dd89b062a0000086000000017c0a8008c6d6d88969485d686c8969485a20000
0000000000000000050000000000000000800000000049424d0049424d04000000007000500010200000494
24d0100000010d7f2f0c24bc5e3d74bc9c2d44bc3d6d449424d000000000000000000010000001c0000000
0100204170000000110020417100204170000000110020417
```

To understand a little more detail about how JNDI naming is implemented under WebSphere z/OS, let us breakdown the preceding LDAP entry into its constituent parts.

The following directory path, which is entered into the SMEUI by the deployer, maps the WithdrawSB EJB:

/PELPLEXT/legacyRoot/pok/sample/BankApp/Withdraw/sb/WithdrawSB

This path can be represented graphically by the following diagram:

```
PELPLEXT
   legacyRoot
       pok
          sample
                 BankApp
                           Withdraw
                                    sb
                                      WithdrawSB
```

PELPLEXT: This entry represents the SYSPLEX on which the WebSphere z/OS application server is running.

LegacyRoot: This entry represents the root anchor for the remaining segments of the LDAP entry.

pok/sample/BankApp/Withdraw/sb/WithdrawSB: This is the fully-qualified directory path which maps to the WithdrawSB EJB home. This segment typically maps to a package name in VisualAge® for Java or a source path in the WebSphere Studio Application Development (WSAD) tool.

The final significant segment of the LDAP entry for an EJB is the IOR. An IOR is a CORBA-defined construct that uniquely identifies an object in a namespace. When your code performs a lookup on the initial context, the WebSphere z/OS Naming service returns a reference to the target EJB's home. Once this reference to the EJB's home is returned, application code must convert it to a reference to an EJB home interface through a process called narrowing. With the reference to the EJB home, the application can create an instance of the EJB by driving the create ( ) or find by Primary Key ( ) methods on the EJB home.

How To Lookup An EJB

The following code sample shows the typical technique for establishing an initial context, performing a JNDI look up on the context, narrowing the home reference, and creating an instance of a WithdrawSB EJB:

```
// Get the initial context
InitialContext  ctx = new InitialContext();

// Look up the WithdrawSB EJB in the JNDI namespace
Object obj = ctx.lookup("java:comp/env/BankApp/WithdrawSB");

 // Narrow the obj reference to an EJB home reference
wsbRef = (pok.sample.BankApp.Withdraw.sb.WithdrawSB)
                    portableRemoteObject.narrow(obj,
        pok.sample.BankApp.Withdraw.sb.WithdrawSB.class);

// Create an instance of the WithdrawSB EJB
wsb = wsbRef.create();
Or
Wsb = wsbRef.findByPrimaryKey(<key>);
```

Note: Application development tools such as IBM's VisualAge for Java and the WebSphere Studio Application Development tool provide wizards for generating access beans that contain code for performing the initial context look up and instantiation of EJBs. By using these wizards, you do not have to code the JNDI look up operations yourself. Instead, all you have to do is create an instance of the access bean and all the JNDI naming resolution is done in the access bean.

Locating Datasources Under WebSphere z/OS

WebSphere z/OS does not store datasource entries in LDAP; rather to ensure faster access to database resources, these entries are stored exclusively in a JNDI cache. Therefore, datasource lookup strings for J2EE applications on WebSphere z/OS must begin with the **java:comp/env** string. For example, lookup string for the datasource that might support the banking application referred to earlier could be "java:comp/env/jdbc/BankAppDataSource."

Mapping EJB Resource Reference Names To J2EE Datasources

During application assembly, you specify datasource reference names or lookup names for EJBs on the Resources tab of the AAT. In the example we are using, the Reference name for the banking application would be **jdbc/BankAppDataSource.**

When you deploy this application into a WebSphere z/OS server, you must define a J2EE resource and J2EE resource instance that represent a datasource that your EJBs access. Then, when you import the EAR file for your application into the SMEUI, you map the JNDI name of the datasource that each EJB accesses to the J2EE resource specified on the SMEUI. This is done through the Reference and Resource Resolution panel displayed by the SMEUI during the deployment process. When the SMEUI creates the container for the server, the datasource name, in this case jdbc/BankAppDataSource, is registered in the JNDI namespace under the java:comp/env context.

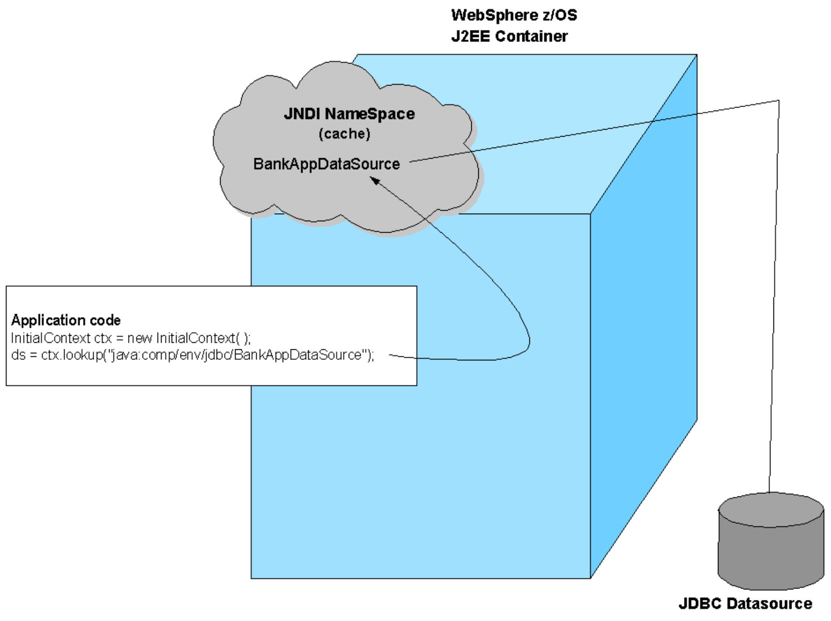
Finally, to perform a lookup on this datasource during runtime, provide the following code in your servlets or access beans:

```
// Get the initial context
InitialContext ctx = new InitialContext ();

// Look up the datasource in the JNDI namespace
DataSource ds = ctx.lookup("java:comp/env/BankAppDataSource");
```

Instead of hard coding the datasource lookup string in the code itself, you can specify the string as a variable in a properties file that you can read during servlet initialization or access bean construction. This allows you to change the datasource name if necessary without regenerating the jar file that contains the access code. Several Java classes such the Properties and Resource Bundle classes provide methods for managing and accessing properties files.

Figure 2 shows the interaction between a lookup in application code on a JDBC resource and how that lookup resolves to a reference to the actual database mapped by that reference.



```
InitialContext ctx = new InitialContext( );
ds = ctx.lookup("java:comp/env/jdbc/BankAppDataSource");
```

Figure 2. Datasource lookup in the JNDI namespace.

Debugging Common JNDI Naming Problems

When you deploy a J2EE to a WebSphere z/OS container for the first time, especially if you migrate an application from distributed WebSphere, you may encounter some problems related registering or accessing J2EE objects in the JNDI namespace. This section addresses some of the most common problems and what you can do to resolve them.

Naming Registration Failure and NoClassDefFoundError

A Naming registration failure manifests itself as a message displayed on the OS/390® or z/OS console. The first time a server region is started after you deploy an application through the SMEUI, the names of all EJBs and servlets are registered in the JNDI namespace. If an exception is thrown during the Naming registration phase, this message is posted to the console.

To debug this problem:

1. Under SDSF, locate the active server region job.
2. Expand the job into its constituent parts with the '?' line command.
3. Select the SYSPRINT file and locate the bottom of the trace.
4. Search backward using the word 'exception' as a search argument.
5. Once you locate an exception trace entry, search for a Java stack trace that might identify the source of the naming registration problem. Typically, this problem occurs when the

Naming server cannot locate a class referenced by an object such as an EJB in the runtime environment. During JNDI registration, the container requires that all classes referenced by objects be available. The topmost entry in the Java exception stack trace identifies the class that the container cannot locate. Usually, a NoClassDefFoundError exception is posted in the stack trace.

6. When you have identified the missing class, locate the .jar file that contains that class in the HFS directory and add the path to that .jar file to the CLASSPATH record in the current.env file.

NameNotFoundException

The exception:

"Name <name> not found in context "java:comp/env": NameNotFoundException" frequently occurs when a server application fails to locate a reference (<name>) in the JNDI namespace. The code that populates the java: namespace relies on exception triggered recursion during the loading process. The first time EJBs are referenced, much of the java:comp namespace structure does not exist for the EJB being activated. The container detects this fact through NameNotFoundExceptions. For these cases, the container's code that populates the namespace correctly builds the missing portions before trying to bind in the java: information.

However, this exception can also occur for failed datasource lookups during runtime, the typical message is:
Name jdbc not found in context "java:comp/env": NameNotFoundException

In this case, the datasource name specified in the code on the lookup method call, in a properties file, or environment variable does not contain the java:comp/env prefix. To correct this problem, ensure that the datasource name includes this prefix. For example, if your datasource name is **jdbc/BankAppDataSource**, it must be defined as **java:comp/env/ jdbc/BankAppDataSource.**

No Such Object

Another exception thrown frequently by the javax.naming class is
javax.naming.NameNotFoundException: LDAP: Error code 32 - No Such Object
Remaining name 'ibm-wsnName=<*name*>,ibm-wsnName=jdbc'

This error occurs when you do not pass the "java:comp/env" prefix on the lookup( ) method when you lookup a datasource name. To remedy this problem, add the "java:comp/env" prefix to the parameter string passed to the lookup( ) method. To ensure platform neutrality, specify the datasource look up string in a properties or environment file. This way, you can have one version of the datasource name on distributed and another on z/OS. The z/OS version must contain the java:comp/env prefix.

Class violates loader constraints

While this condition is not directly related to JNDI naming, it frequently occurs after you correct a NameNotFoundException or NoClassDefFoundError. In this case, the container has successfully located the reference in the JNDI namespace but it is unable to load the class associated with the reference. The following exception is thrown and is displayed at the top of the java exception stack in the server region's SYSPRINT log:
java.lang.LinkageError: Class <classpath/classname> violates classloader constraints.

To solve this problem, try changing the classloader mode in the server region's jvm.properties file. WebSphere z/OS uses classloader mode 1 (compatibility mode). To change the mode,

1. Edit the server region's jvm.properties file which can be found in HFS directory WebSphere390/CB390/controlinfo/envfile/<sysplex-name>/<server-name>/.
2. Add the following line to the file:
   com.ibm.ws390.server.classloadermode=*n* where *n* is value 0, 1, 2, or 3.

Note: For detailed information about WebSphere z/OS classloader modes, see the documentation for APAR PQ53684.

Summary

This technical paper focused on JNDI naming concepts that programmers, application assemblers, server application deployers should keep in mind when designing and deploying J2EE applications into WebSphere z/OS servers. These concepts include:

- *EJB names are specified in the deployment descriptors and these names are registered in a JNDI namespace during J2EE application deployment.*
- *Datasource names can be hard coded, specified in properties files, or defined with environmental variables. For WebSphere z/OS, datasource names must be prefixed with the string "java:comp/env".*
- *Client code, such as servlets and access beans, must obtain a reference to a server's initial context and then drive a lookup for an EJB or datasource on the initial context.*
- *To debug common JNDI naming problems on WebSphere z/OS, analyze the Java exception stack traces that are posted in the server region's SYSPRINT log.*

References

Enterprise JavaBeans Specification, v1.1
Sun Microsystems, Inc.

WebSphere Application Server V4.0 and V4.0.1 for z/OS and OS/390
Configuring Web Applications
Donald C. Bagwell
IBM Washington Systems Center

Enterprise JavaBeans for z/OS and OS/390 WebSphere Application Server
IBM Redbook SG24-6283