

# Code Packaging and Deployment



## Overview



- Before you can configure EJBs in the Administrative Console, they must first be deployed.
- There are two ways to create a Deployed EJB JAR:
  - ▲ WebSphere Administrative Console
  - ▲ Visual Age for Java Enterprise Edition



- The deployed JAR file contains enterprise bean code, such as bean class, home and remote interfaces, deployed classes, serialized deployment descriptor (.ser file), and the manifest.
- The Administrative Console prompts for deployment if the EJBs being created have not yet been deployed.
- VA-J provides an option to export a deployed JAR. Note: VA-J 2.0 EJBs do not work with WebSphere 3.x.

## Creating an EJB JAR Manually



- Create Java files using an editor.
- Compile Java files using the **javac** compiler.
- Tasks that will be completed in lab:
  - ▲ Use the **jar** command to create a Java ARchive (JAR) of the class files.
  - ▲ Start **jetace** and load the JAR.
  - ▲ Create and define deployment descriptor for each EJB.
  - ▲ Save the deployable EJB JAR.



- The deployable EJB JAR you save will not have the deployed classes at this point.
- When configuring the EJBs in the WebSphere console, you have the option of creating stubs and proxies for workload management. The process of deploying the beans generates the deployed classes.

## Or, Use VisualAge for Java



VisualAge for Java (VA-J) does the same tasks in an integrated development environment. VA-J provides four options for exporting Enterprise JavaBeans:

- Export Deployed JAR
  - Export EJB JAR
  - Export EJB JAR for Component Broker
  - Export client JAR
- 
- 



- Export Deployed JAR
  - This method works best for use with WebSphere Advanced Edition.
  - Includes deployed classes.
- Export EJB JAR (often called "vanilla" jar--does not contain deployed classes)
  - The EJB JAR file contains the enterprise bean code, such as the enterprise bean class, home and remote interfaces, serialized deployment descriptor file (.ser), and manifest. Everything the deployed JAR had except the classes.
  - Contains the same components as the JAR file saved when using **jetace**.
- Export EJB JAR for Component Broker (CB): creates the same JAR as above, but launches the WebSphere Application Server Enterprise Edition/CB deployment tools to generate the deployed code. The JAR file contains meta-data.
- Export client JAR: Includes all the classes and interfaces used to access enterprise beans from a client application. The client JAR you export will be workload management enabled.

## Advantages of Using of VA-J



- Incremental compiling
- IBM VisualAge for Java Enterprise Edition provides a development and test environment for enterprise beans.
- VA-J supports WebSphere advanced features like finders, read-only methods, EJB associations, and EJB inheritance.
- VA-J has tools that help map entity beans, generate bean wrappers, generate deployment code for EJBs, and generate test clients.
- Refer to the VA-J Help for additional information on these topics.



- Incremental compiling: VA-J compiles the class after each change and flags errors
- Test environment provided: EJBs can be tested and debugged in an incremental fashion.
- Constat method indication: methods that do not result in EJB state changes can be defined as const methods to eliminate needless database syncs.

## Setting the Classpath



- For EJBs to deploy without error, the classpath must be set correctly. Appendix A describes where the classpath needs to be specified.



## Driver Classes and Data Sources



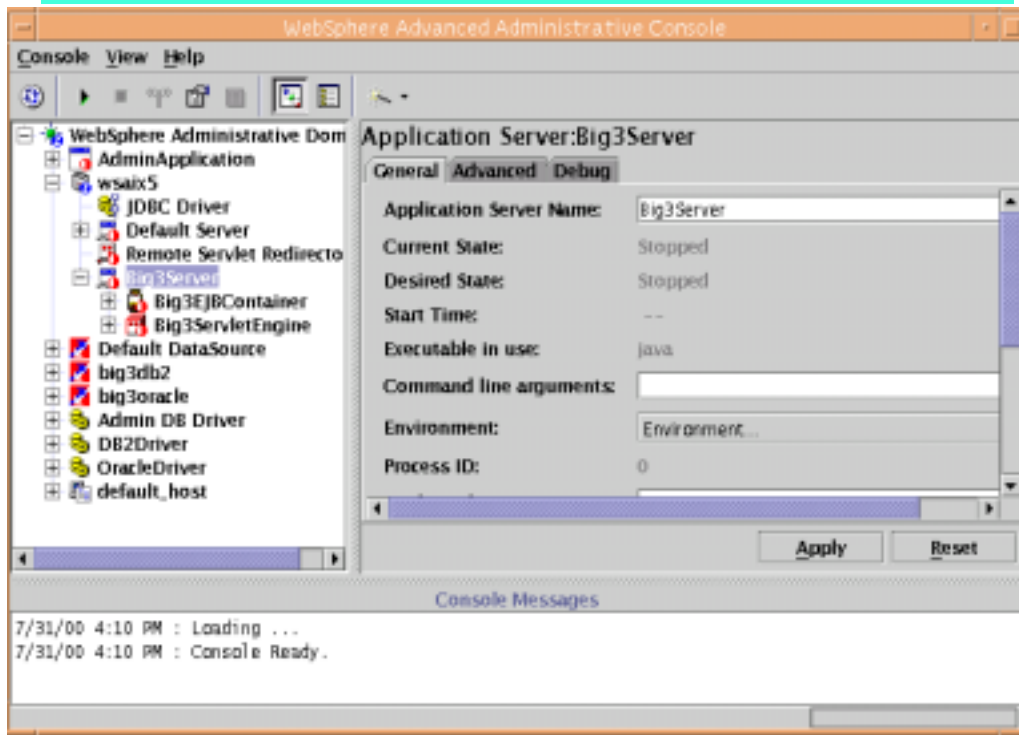
- See the online product release notes for the latest information on supported database drivers.
- For WebSphere Advanced Edition Version 3.5, the required database versions are:
  - ▲ DB2 6.1 with fixpack 4
  - ▲ Oracle Version 8.i Release 2
  - ▲ Sybase 12.0



For WebSphere Advanced Edition Version 3.02, the required database versions were:

- DB2 5.2 with fixpack 10 or 11 (shipped with product)
- DB2 6.1 with fixpack 2 (recommended for distributed transactions)
- Oracle Version 8.1.5 Driver Manager JDBC - Thin/100% Java for JDK 1.1.x. Both Oracle Version 8.0.5 and 8.1.5 are supported **but the driver must be 8.1.5.**
- In beta, Sybase. Be on the lookout for updates.

# Administrative Console



# Troubleshooting Tips



## Tools



- Tracing and Logging
- Object Level Trace and Debugging (OLT and OLD) facilities
- Administrative Intervention
  - ▲ Forcing a status change
  - ▲ Managing transactions
- Resource Analyzer



## WebSphere Tracing and Logging



- WebSphere includes built-in tracing and logging interfaces
- All components are instrumented
- Trace components are Java classes or packages
- Tracing is performed to a ring buffer in memory. Data is formatted with dumped to a file.
- Tracing can be dumped to **stdout**, **stderr** or a file. This can be costly due to formatting and I/O



## Messages versus Tracing



- Messages and tracing are generated by the system or by user code in response to system events
- Messages provide a high-level view of important events
  - ▲ Always collected; displayed on WebSphere Administrative Console
- Tracing is useful when more detailed information is required.
  - ▲ You specify trace settings



- When a message event occurs, the message displays in the Console Messages area located at the bottom of the Administrative Console display.
- You can also view messages by using the Serious Event Viewer. Click **Console-->Trace-->Event Viewer** to display the event viewer. The Event Viewer allows you to configure how many and what types of messages to display.
- When you require more information than what the messages provide, enable tracing. WebSphere creates a trace runtime instance for each server in the administrative domain. The trace runtime instance collects data from trace events.
- **Note:** Using trace can have a negative impact on performance. Use trace sparingly.

## Types of Messages



- **Audit:** a significant event that must be recorded
- **Warning:** a problem that must be fixed to prevent a more serious condition
- **Terminate:** a problem has terminated normally (returns 0)
- **Fatal:** a process has encountered a fatal error and has terminated abnormally (returns -1)



- Audit: general messages, console ready
- Warning, Terminate, and Fatal: show up as red text in the messages pane of the administrative console. Written out to log files (stderr, stdout)

## Tracing Events



- **Entry:** a process has entered a method
- **Exit:** a process has exited a method
- **Debug:** provides debugging information
- **Event:** a significant event has occurred, such as a state change



- You can look at the trace log files
- You can also:
  - specify a file
  - specify where to put the logs
  - specify where to put the stdout and stderr logs
- Using trace provides more detailed information about exceptions

## Trace Settings

- The syntax for a trace string follows:

```
<comp1>=<type>=[enabled|disabled],...:<comp2>=<type>=[enabled|disabled],...
```

- ▲ **comp**—the component name can be a class, package, or group of classes or packages
- ▲ **type**— type of tracing. Valid types include:
  - **debug**—provides debugging information
  - **entry/exit**— indicates that a process entered or exited a method
  - **event**— indicates that a significant event occurred
  - **all**— includes all types of tracing
- ▲ **[enabled|disabled]**—specifies whether tracing is enabled or disabled for the component.

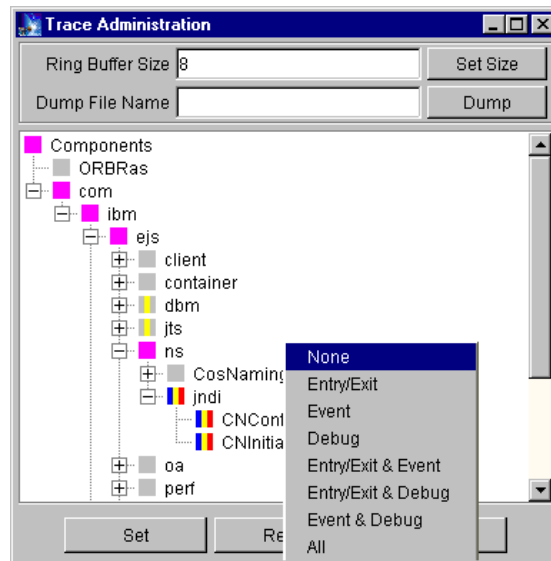
- Trace output destination can be one of the following: **stderr**, **stdout**, or a filename



- If you want trace written directly to a file, you should use **stdout** or **stderr**. This is more efficient and less likely to mask any problems that tracing to a specific file might, in rare cases, cause.
- If you specify a relative filename for a trace or log destination, the file is created in the working directory of the server. On NT, the default working directory for a server is **WINNT\SYSTEM32**.

## Trace Administration

- Select the server you want to trace.
  - ▲ To trace an Admin Server, select the **Node**
  - ▲ Select an **Application Server** to trace that server
- Select **Trace** from the context menu
- Enable trace events for classes or packages
- Dump the ring buffer to a file



- The trace administration window allows you to interactively control trace options for a running administrative server or for an application server.
- To trace an administrative server select the node, then select Trace from the context menu.
- To trace an application server select the server, then select Trace from the context menu.
- To trace a component, expand the component tree by clicking the square by each component name. Right click to select the trace setting for the component. The squares will change color to indicate which combination of trace settings are in effect. Click **Set** to apply the trace setting. The trace setting will be saved in the server properties so that it is in effect the next time the server is started.
- The trace facility keeps a ring buffer of trace messages. You can dump this buffer to a file by entering a file name and clicking the Dump button. You can also increase the size of the ring buffer if necessary.

## Tracing the Administrative Server

- If you have problems starting the admin server, you should look at its log file. You may need to enable tracing for more detail.
- Example problems:
  - ▲ Database connection problems
  - ▲ NT Workstation not running
- Log files
  - ▲ `<as_root>/logs/tracefile`
  - ▲ `<as_root>/logs/nanny.trace`
  - ▲ `<as_root>/logs/adminserver_native.log.was-oop.date_and_time`
- Configuring Admin Server
  - ▲ `<as_root>/bin/admin.config`

```
com.ibm.ejs.sm.adminServer.traceString=com.ibm.ejs.sm.*=all=enabled
com.ibm.ejs.sm.adminServer.traceOutput=astrace.txt
```
- Interactive trace in admin console
  - ▲ Select the node and select **Trace** from menu



## For More Information. . .



- See the following documents on the WebSphere site:  
<http://www.ibm.com/software/webservers/appserv/library.html>
  - ▲ Writing Enterprise JavaBeans
  - ▲ Online product release notes
  - ▲ Documentation Center -->"How do I" tab: "Develop enterprise beans" and "Deploy enterprise beans"
  
- VisualAge for Java Enterprise Edition Help tasks: "Using the EJB Development Environment"



- Note: From the Web site listed above, click "View additional documentation" to find white papers, redbooks and redpieces, and various other related documents.

## IBM WEBSHERE WORKSHOP - LAB EXERCISE

### Code Packaging Lab

#### **What This Exercise is About**

In this exercise, you will package a group of Enterprise Java Beans (EJBs) that make up the business logic of the Big3 application. The purpose of packaging the beans is to prepare them for deployment.

There are three levels of packaging that you will use to create JAR files. The first, also known as a “vanilla” JAR file, consists of the .class files and a manifest. The second, called a deployable JAR file, contains .class files, deployment descriptors (.ser), and the manifest. The third type, the deployed JAR, contains .class files, deployment descriptors (.ser), manifest, and stubs and skeletons.

In this lab exercise, you will use two different methods to package: manually using **jetace**, and by using VisualAge for Java Enterprise Edition (VA-J).

#### **User Requirement**

You need the recommended versions of DB2, WebSphere Advanced Edition, and VA-J installed on your system. You will also need the compiled .class files for the EJBs.

#### **What You Should Be Able to Do**

You should become familiar with the steps that are necessary to package EJBs. You will also be familiar with two ways you can package EJBs and choose the best method to use for your particular situation.

#### **Introduction**

There are two ways you can package EJBs for WebSphere. Both ways will be discussed in this lab. Manual packaging requires more steps and will be discussed first. It is strongly recommended that you use one of the available integrated development environments (IDEs) such as IBM VisualAge for Java (VA-J) to do the packaging. VA-J automatically generates a significant part of the enterprise bean code and contains integrated tools for packaging and testing the enterprise beans.

In the lab, we will use the Big3 example for both manual packaging and for packaging with VA-J.

## Exercise Instructions

To complete this lab you will be using the NT computer to do the packaging. You will then use FTP to put the deployable JAR file on the AIX server. Estimated time: 1 hour 15 minutes

The following lab consists of two parts. The necessary applications have already been installed on your lab computer. Simply follow the steps outlined below to complete the lab.

The following are the files that will be used for this code packaging lab. They have been installed on the NT lab computers in the **C:\big3** directory.

C:\big3	- Big3 Application Home Directory
\ejb	- Big3 compiled .class files
Claim.class	
ClaimBean.class	
ClaimBeanFinderHelper.class	
ClaimHome.class	
ClaimKey.class	
Policy.class	
PolicyBean.class	
PolicyBeanFinderHelper.class	
PolicyHome.class	
PolicyKey.class	
ProcessClaim.class	
ProcessClaimBean.class	
ProcessClaimHome.class	
\client	- Big3 JavaClient Directory
\html	- Big3 .html files
\servlets	- Big3 Servlets
\source	- Big3 .java files in .zip format
\util	- Big3 NamingContextHelper .class file*

\*the NamingContextHelper caches both Initial Contexts and Objects found in those Initial Contexts. Caching in this way can improve performance.

### Part One: Manual Packaging

- \_\_1. The first step to manual packaging is to create a Java Archive (JAR). It is important to preserve the file structure of the .class files for use with the Big3 application, so you need to execute the **jar** command from the root directory.

**NOTE:** The **jar** command is located in <JAVA\_HOME> directory. In the past, this directory was added to the classpath when WebSphere was installed. This is no longer the case. WebSphere is aware of <JAVA\_HOME> but doesn't change the path in case the user has a need to have an earlier version of the JDK installed. If you

know you don't need an earlier version of the JDK, you can add the path to the JDK to your environment variables. Change the data source to **jdbc:db2:big3**

- Click **Start --> Programs --> Command Prompt**
- At the DOS prompt, type `cd \big3` and press **Enter**
- In the **C:\big3** directory, execute the following batch file: **jarbig3.bat**, or type:  
`c:\WebSphere\AppServer\jdk\bin\java jar cvf0`  
`c:\big3\big3manual.jar big3\ejb\*.class big3\util\*.class` and press **Enter**. This entire command is typed on one line!

The .class files will be archived into the **big3manual.jar** JAR file. The manifest that defines the EJB is also created.

Note: The Java Development Kit compiler **javac** is also required to compile the .java source files into .class files. To save time, this step has already been done for you. The proper directory structure has also been created for you.

- \_\_2. Once the **big3manual.jar** is created, you are ready to add the deployment descriptors to the EJBs using the WebSphere Application Server **jetace** program.

In an MS-DOS command window, change directories to `<as_root>\bin` and start **jetace** by typing **jetace** and pressing **Enter**.

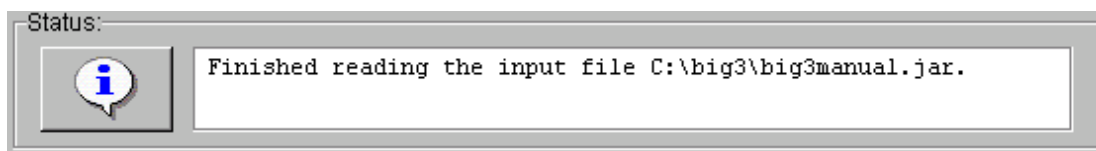
Note: The **jetace** tool can also be run from the command line to create an EJB JAR file. The syntax of this command follows, where `xmlFile` is the name of an XML file containing the enterprise bean deployment descriptors:

**jetace -f xmlFile**

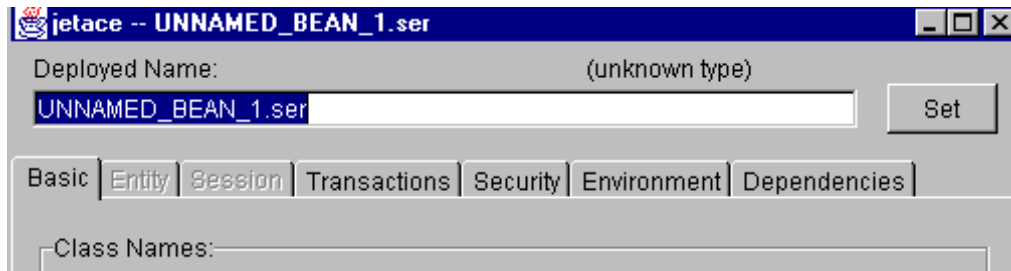
For more information on the syntax of the XML file required for this command, see *Writing Enterprise Beans in WebSphere*.

- \_\_3. Load the **big3manual.jar** file by selecting **File --> Load ...** And select the **C:\big3\big3manual.jar** file. Click **Open**.

The status message should indicate the JAR file loaded successfully.



- \_\_4. Create a deployment descriptor by clicking the **New** button. A default descriptor named **UNNAMED\_BEAN\_1.ser** will be created.



- \_\_5. Change the Deployed Name from **UNNAMED\_BEAN\_1.ser** to **big3/ejb/ProcessClaim.ser**. Click the **Set** button to change the name.

Note: If the background of a field appears as cyan, you haven't clicked the Set button to save your change.

- \_\_6. Make the following selection for the Class Names. Use the drop down combo box to access the list.

Enterprise Bean Class: **big3.ejb.ProcessClaimBean**  
Home Interface: **big3.ejb.ProcessClaimHome**  
Remote Interface: **big3.ejb.ProcessClaim**

- \_\_7. Change the JNDI HomeName to **ProcessClaim** and click the **Set** button.
- \_\_8. Select the **Session** tab and change the Session Timeout (seconds) to **0**. Click the **Set** button to make the change. And change the State Managed Attribute to **STATELESS\_SESSION**.

Note: Setting the Session Timeout to 0 indicates that it never times out. This is the default setting. Setting the session timeout to a value other than zero allows WebSphere to passivate session information after the specified time has passed.

- \_\_9. On the **Transactions** tab, change Transactions Attribute to **TX\_REQUIRED** and Isolation Level to **READ\_COMMITTED**.

Note: if you are using only DB2 as your database, you may wish to use **REPEATABLE\_READ** instead. **READ\_COMMITTED** is used here because you will be using both DB2 and Oracle as your data sources during this class. See the transactions information in the Programming Model Issues lecture for more information on isolation levels and database compatibility.

- \_\_10. On the **Security** tab, change the Run-As Mode to **SYSTEM\_IDENTITY**.
  
- \_\_11. Close the **big3/ejb/ProcessClaim.ser** properties screen.
  
- \_\_12. Click **New** again, and continue to create the following deployment descriptors using the following information (Hint: Don't forget to click the **Set** button when you change settings! If the background color in the field is cyan, the data has not been saved. Clicking the **Set** button saves your entry.)

big3/ejb/Claim.ser

Enterprise Bean Class: **big3.ejb.ClaimBean**  
Home Interface: **big3.ejb.ClaimHome**  
Remote Interface: **big3.ejb.Claim**  
JNDI Home Name: **Claim**  
Primary Key Class: **big3.ejb.ClaimKey**  
Container-Managed Fields: **amount, claimNo, state**  
Transaction Attribute: **TX\_MANDATORY**  
Isolation Level: **READ\_COMMITTED**  
Run-As Mode: **SYSTEM\_IDENTITY**

big3/ejb/Policy.ser

Enterprise Bean Class: **big3.ejb.PolicyBean**  
Home Interface: **big3.ejb.PolicyHome**  
Remote Interface: **big3.ejb.Policy**  
JNDI Home Name: **Policy**  
Primary Key Class: **big3.ejb.PolicyKey**  
Container-Managed Fields: **amount, policyNo, premium**  
Transaction Attribute: **TX\_MANDATORY**  
Isolation Level: **READ\_COMMITTED**  
Run-As Mode: **SYSTEM\_IDENTITY**

Note: It is important to fill in the JNDI name. Failure to fill in the JNDI name will cause the error, "Unable to open the given jar file." when you attempt to deploy the application in WebSphere.

- \_\_13. Select **File --> Save As...** and save the JAR file as **big3mandeployable.jar** in the **C:\big3** directory.

- \_\_14. Exit the **jetace** program by selecting **File --> Exit**. You should have two JAR files in the **C:\big3** directory: **big3manual.jar** - (contains just the .class and manifest files) and **big3mandeployable.jar** - (contains the .class, manifest, and deployment descriptor (.ser) files).

Note: In WebSphere Application Server Advanced Edition you need to choose the **Deploy and Enable WLM** option when deploying the .jar file to enable the application for workload management.

## **Part Two: Packaging Using VisualAge for Java**

Using VisualAge for Java makes the packaging process easier. The following lab will show you the steps of defining each bean using VisualAge for Java (VA-J). We could import the **big3mandeployable.jar** (the one with the deployment descriptors) directly into VA-J, but we should show you the steps to define each bean within VA-J.

Before you can develop enterprise beans in VA-J, you must set up the EJB development environment. You need to perform this step only once. This procedure directs VisualAge for Java to import all of the classes and interfaces required to develop standard enterprise beans.

Select **File --> Quick Start** and highlight **Features** and select **Add Feature**. Click **OK**. Highlight **IBM EJB Development Environment 3.5** and click **OK**. The IBM EJB Development Environment 3.5 will be imported and an EJB tab will be created for you.

**To save time, this step has been done for you on the lab computer.**

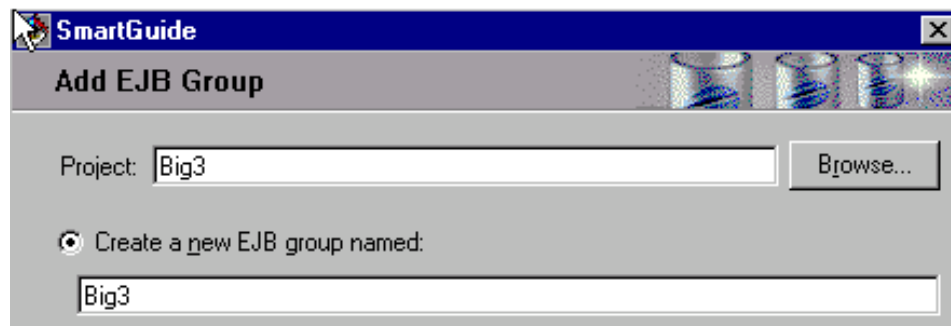
- \_\_1. Start VA-J by double clicking the **IBM VisualAge for Java** icon that is located on your desktop. Click **Close** on the Welcome to VisualAge screen.

If this is your first time setting up VA-J for packaging files for use in WebSphere, you need to add the **db2java.zip** to the Workspace class path to fix problems with generating code within VA-J. To do this, select **Window --> Options**. Select **Resources** from the tree view. Then in the **Workspace class path** field, add the full path name of **db2java.zip**.

**To save time, this step has been done for you on the lab computer.**

- \_\_2. You now can import the Big3 source file into VA-J. Select **File --> Import....**

- \_\_3. Select **Jar file** on the SmartGuide screen and click **Next**.
- \_\_4. Click the **Browse** button for the Filename field and select **C:\big3\source\source.zip**  
*Hint: you may want to click the drop down for File Types to select all files or zipped files.*
- \_\_5. Leave a check mark next to **.java** for “What type of files do you want to import?” The other check boxes can be empty for the Big3 application.
- \_\_6. Enter **Big3** for the Project name.
- \_\_7. Click the **Finish** button and answer **Yes** to the create project. VA-J will now import the Big3 source code and create a JAR file. Click **OK**.
- \_\_8. After the import, click the **Projects** tab and note the **Big3** project was created. Click the **Packages** tab and note that the **big3.client**, **big3.ejb**, **big3.servlet**, and **big3.util** packages were created.
- \_\_9. Create the EJBs. The steps are similar to those used with the **jetace** tool.
  - From the **EJB** menu, select **Add --> EJB Group...**
  - For the project name, click the **Browse** button and pick the **Big3** project. Click **OK**.
  - Enter **Big3** for the *Create a new EJB Group named:* field.
- \_\_10. Click **Finish**.



- \_\_11. Right-click the **Big3** group that was created and select **Add --> Enterprise Bean....**
- \_\_12. Select **Create a new enterprise bean** and enter **ProcessClaim** as the Bean name.

- \_\_13. Use **Session bean** as the Bean type. Select **Use an existing bean class**. Click the Browse button next to the Package: field. Double click **big3.ejb**. Click the Browse button next to the Class: field. Double click **ProcessClaimBean**. Double clicking the class name prepends the package name in the form. Compare your display with the following information:

**Note:** If the fields did not fill in properly, be sure to click the **Browse** button in the class field. Select the type name if it is not highlighted. Click **OK**. The program will fill in **big3.ejb**. in the class field for you **if** you selected the big3.ejb package first.

- \_\_14. Click **Next**.

- \_\_15. Fill in the following (accept the defaults).

- \_\_16. Click the **Finish** button.

- \_\_17. Right-click the **Big3** group and select **Add --> Enterprise Bean**. Use the following information to define the bean:

Bean name: **Policy**

Bean type: **Entity bean with container-managed persistence (CMP) fields.**

Create bean: **Use an existing bean class**

Project: **Big3**

Package: **big3.ejb**

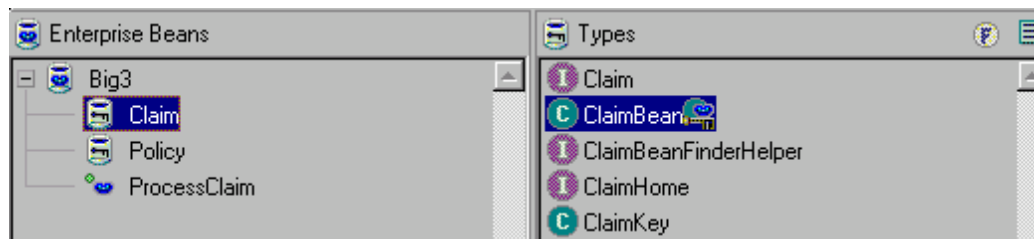
Class: **big3.ejb.PolicyBean**  
Home interface: **big3.ejb.PolicyHome**  
Remote interface: **big3.ejb.Policy**  
Key class: **big3.ejb.PolicyKey**  
Leave checked **Create finder helper interface to support finder methods**

\_\_18. Click **Finish**.

\_\_19. Right-click the **Big3** group and select **Add --> Enterprise Bean**. Use the following information to define the bean:

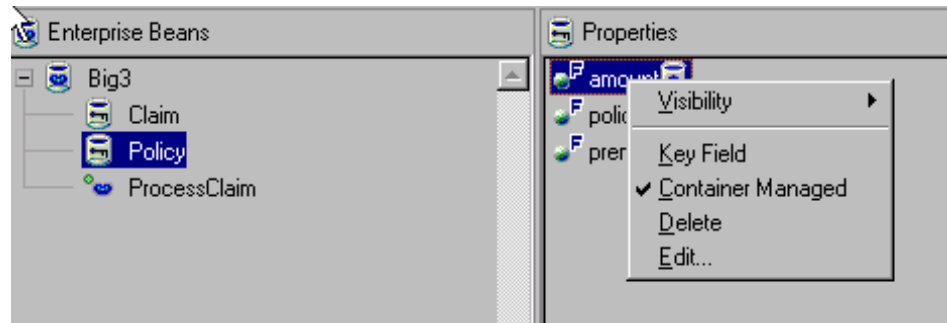
Bean name: **Claim**  
Bean type: **Entity bean with container-managed persistence (CMP) fields**.  
Create bean: **Use an existing bean class**  
Project: **Big3**  
Package: **big3.ejb**  
Class: **big3.ejb.ClaimBean**  
Home interface: **big3.ejb.ClaimHome**  
Remote interface: **big3.ejb.Claim**  
Key class: **big3.ejb.ClaimKey**  
Leave checked **Create finder helper interface to support finder methods**

\_\_20. Click **Finish**. You now have the three beans.



\_\_21. Now you want to make sure all the fields in the entity beans are container managed. In the **Types** column, click the **F** on the upper right corner of the container tool bar. Select each field, right click, and select **Container Managed**. Do this for each field on both

entity beans.



- \_\_22. Under the Big3 group in the Enterprise Bean column, right click the Claim bean. Select **Properties** and the JNDI name (Home interface), Transaction Attribute, Isolation Level, and Run-As Mode.

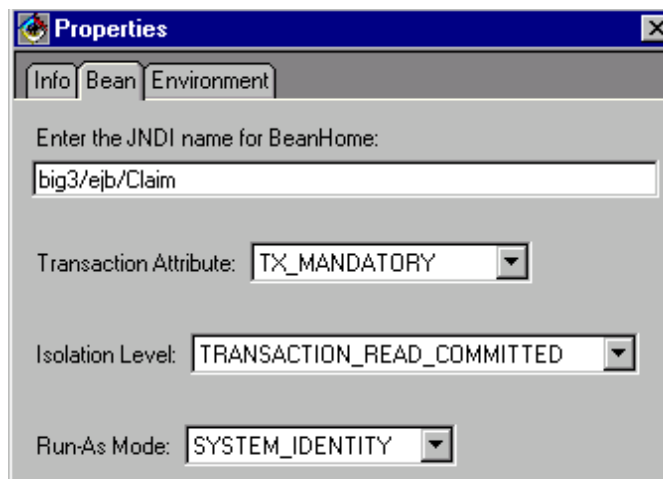
JNDI Home Name: **big3/ejb/Claim**

Transaction Attribute: **TRANSACTION\_TX\_MANDATORY**

Isolation Level: **READ\_COMMITTED**

Run-As Mode: **SYSTEM\_IDENTITY**

Your display should look like this:



- \_\_23. Update the properties on the Policy and the ProcessClaim bean adding the JNDI name (Home interface), Transaction Attribute, Isolation Level, and Run-As Mode.

Policy

JNDI Home Name: **big3/ejb/Policy**

Transaction Attribute: **TRANSACTION\_TX\_MANDATORY**

Isolation Level: **READ\_COMMITTED**

Run-As Mode: **SYSTEM\_IDENTITY**

### ProcessClaim

JNDI Home Name: **big3/ejb/ProcessClaim**

Transaction Attribute: **TRANSACTION\_TX\_REQUIRED**

Isolation Level: **READ\_COMMITTED**

Run-As Mode: **SYSTEM\_IDENTITY**

- \_\_24. Next add the Schema settings. Right-click the **Big3** group and select **Add --> Schema and Map from EJB Group**. VA-J will add the Schema information. You will notice the icons change slightly for the Claim and Policy beans.



Note: The schema mapping is a set of definitions for all attributes matching all the columns in your database table, view, or SQL statement. VA-J needs to have this mapping because some of the beans will be accessing databases.

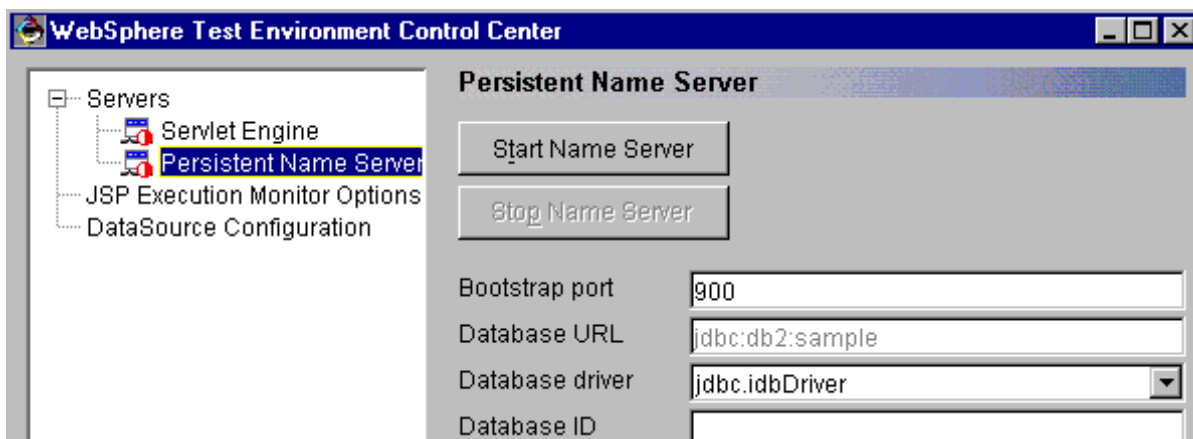
- \_\_25. To generate deployment code, right-click the **Big3** group and select **Generate --> Deployed Code**. VA-J will generate the Deployed Code.

When you generate deployed code, the enterprise beans are analyzed to ensure that the rules specified in the Sun Microsystems EJB specifications are met, as well as the rules specific to the EJB server. If any errors are found, error messages are issued to the Console log. Depending on the severity of the error, code generation may not begin.

For each enterprise bean that resides in a standard enterprise bean input file, the code-generation tool generates the home and EJBObject (remote) implementations and implementation classes for the home and remote interfaces, as well as the JDBC persister and finder classes for CMP beans. It also generates the Java ORB, stubs, and tie classes required for RMI access over IIOP, as well as stubs for the home and remote interfaces.

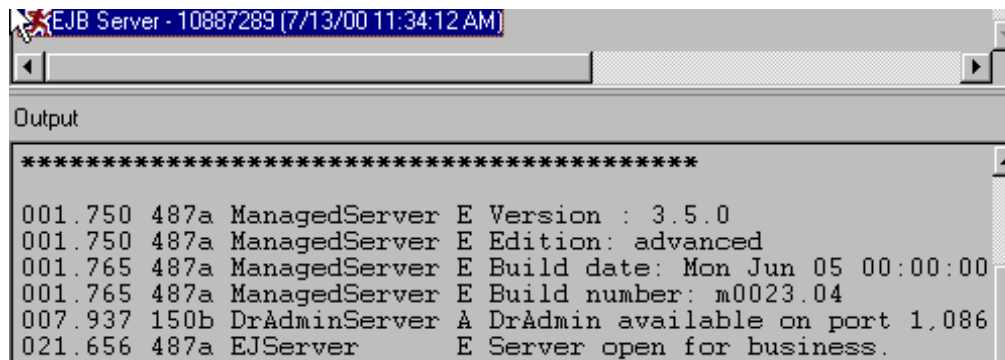
- \_\_26. When you actually deploy the JAR using VA-J, the program creates the client stubs that are necessary for WLM. They were included when the deployed code is exported. In addition, you can export a smaller-sized client JAR file that is workload management enabled. Right click the **Big3** group and select **Export --> Client JAR**.

- \_\_27. For the JAR file, click the **Browse** button. Type the following: **C:\big3\big3client.jar**. Leave check marks next to **beans** and **.class**.
- \_\_28. Select *Compress the contents of the JAR file*, if desired.
- \_\_29. Click **Finish**. VA-J will create the **big3client.jar** file.
- \_\_30. Now you can test your EJBs from within VA-J. Right click the **Big3** group and select **Add to...-->Server Configuration**.
- \_\_31. In the Server Configuration window, right click **EJB Server-->Properties**.
  - Change the data source to **jdbc:db2:big3**
  - Enter the username: **db2admin**
  - Enter the password: provided by instructors
  - Click **OK**
- \_\_32. Right click **EJB Server** and select **Create database table**.
- \_\_33. From the menu bar at the top, select **Workspace-->Tools-->WebSphere Test Environment**. When the test environment control center window opens, select **Persistent Name Server** and start the name server. A console window will open. Watch the messages until you see the message, "E Server open for business" before proceeding.



Note: The Persistent Name Server is using the default configuration where instantDB is the repository.

- \_\_34. Right click **EJB Server** and start the EJB server. A console window will open. Watch the messages until you see the message, “E Server open for business” before proceeding.



The screenshot shows a console window titled "EJB Server - 10887289 [7/13/00 11:34:12 AM]". The output area contains the following text:

```

*****
001.750 487a ManagedServer E Version : 3.5.0
001.750 487a ManagedServer E Edition: advanced
001.765 487a ManagedServer E Build date: Mon Jun 05 00:00:00
001.765 487a ManagedServer E Build number: m0023.04
007.937 150b DrAdminServer A DrAdmin available on port 1,086
021.656 487a EJServer      E Server open for business.

```

- \_\_35. Open an MS-DOS command window. Change directories to c:\big3\client and execute the following command:

```
runclient <policy_number> <claim_number> <amount> <NT_machine_name>
```

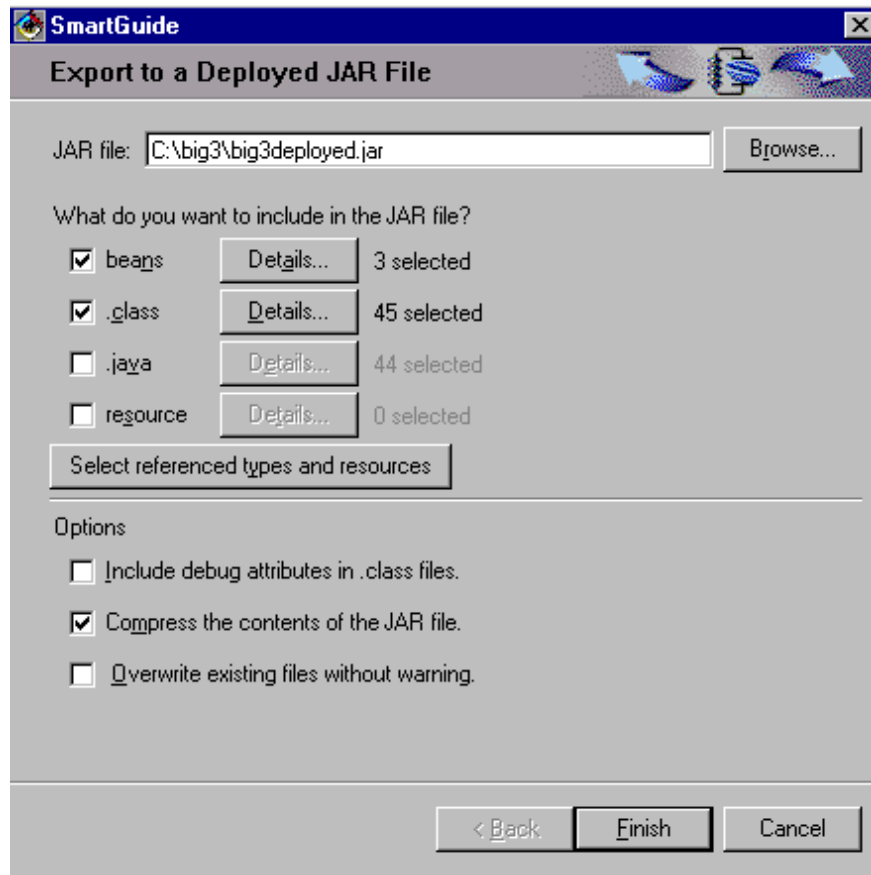
For example, if you were on a machine named MY\_NT, the command would be:

```
runclient 123 456 99 MY_NT
```

If you've configured your beans correctly, you should get no errors in the VA-J console.

- \_\_36. Right-click the **Big3** group and select **Export --> Deployed Jar....**
- \_\_37. Click **Browse** and use the name: **C:\big3\big3deployed.jar**. Make sure the **beans** and **.class** check boxes are selected.
- \_\_38. Click **Select referenced types and resources** to add the NamingContextHelper .class file to your deployed JAR file. Alternatively, you can select the **Details** button to the right of the .class check box. Scroll down until you find the check box for NamingContextHelper and select that file.

\_\_39. If you wish, check **Compress the contents of the JAR file**.



\_\_40. Click **Finish**.

The **big3deployed.jar** can now be deployed into WebSphere Application Server.

\_\_41. Finally, use FTP to put **C:\big3\big3deployed.jar** to the **<as\_root>/deployedEJBs** directory on the AIX computer. Use **bin** to send the file as binary. The instructor can assist if you are unfamiliar with FTP commands.

Note: To find **<as\_root>** on the AIX machine, execute the following command: **find / -name startupServer.sh** Make a note of the path before **/bin/startupServer.sh**. This directory will be referred to as **<as\_root>** for the remainder of the course.

## **What you did in this exercise**

This lab showed you how use two methods of packaging EJBs. First you created a “vanilla” JAR file using the **jar** command. Then you created a deployable JAR file manually using **jetace** (big3mandeployable.jar). Finally, you used VisualAge for Java to create a deployed JAR file (big3deployed.jar) and a client JAR file (big3client.jar), and used the VA-J test environment to test your JAR file before exporting it.