16

The ps-makeejb Command

This chapter introduces the **ps-makeejb** command, which is central to the PowerTier development process. You use this command to build EJB source files into a deployable JAR file. The following describe the **ps-makeejb** command and how to use it:

- Understanding the ps-makeejb Command
- Using the ps-makeejb Command
- Developing a Complete Application
- Using the ps-makeejb Command for Iterative Subtasks

Understanding the ps-makeejb Command

The **ps-makeejb** command acts like the **make** command. A variety of options allow you to perform incremental tasks during iterative development. The **ps-makeejb** command:

- compiles Java source files
- generates container-adapter code (including the RMI stubs and skeletons that enable remote communication)
- generates starter deployment descriptors in XML format
- packages the deployment descriptors and compiled EJBs into JAR files
- cleans directories associated with a PowerTier project from the Persistence pantry

To ensure that you are using the correct environment variables, run **ps-makeejb** from a PowerTier command prompt. When you run **ps-makeejb**, you can use the **-verbose** option with any primary option to see more detailed information as the command runs. To see a list of available options, run the **ps-makeejb** command with no options specified. For more information, see the Syntax section of "ps-makeejb Command" in the *Class Reference*.

The following sections describe how the **ps-makeejb** command works:

- Specifying Source and Destination Directories
- Compiling and Building
- ps-makeejb and Deployment Descriptors
- ps-makeejb and JAR Files
- Clearing the Pantry
- Primary Options

Specifying Source and Destination Directories

The **ps-makeejb** command requires an implicit or specified source (to identify files on which to act) and a destination (to know where to place results). You can allow **ps-makeejb** to use default locations for the source and destination, or you can specify them on the command line.

Default Source

The **ps-makeejb** command looks for a **ps-makeejb.cfg** file in the directory from which it is invoked. The PowerTier code generator (**ps-gen**) creates a **ps-makeejb.cfg** text file when it generates CMP entity bean source files. The **ps-makeejb.cfg** file specifies the project and packages for which **ps-makeejb** builds JAR files or deployment descriptors.

You can add other package names to this text file so that **ps-makeejb** includes those packages in the build. If you are developing only BMP entity beans or session beans, you can create a **ps-makeejb.cfg** file to specify source parameters to the **ps-makeejb** command.

The first line of the **ps-makeejb.cfg file** specifies the project. The remaining lines specify package names. The **ps-makeejb.cfg** file has the following format:

```
PROJECT_NAME=projectName
PACKAGE_NAME=packageName
```

If you include BMP entity beans or session beans to existing packages generated for CMP entity beans, you do not need to change the **ps-makeejb.cfg** file. If you add subdirectories to existing packages, you do not need to change the **ps-makeejb.cfg** file either.

Default Destination

The default output destination depends on what you are building with **ps-makeejb**. Table 33 shows the default destinations of the results of this command.

Table 33. Default Destination for ps-makeejb Output

Result	Default Location
.class files	The Persistence pantry, in a directory hierarchy that mirrors the package structure.
generated container source files	A directory hierarchy that mirrors the package structure, starting from the current directory.
.jar files	The current working directory.
.xml files	The current working directory.
.ser file	The current working directory.

Compiling and Building

You can use the **ps-makeejb** command to build your EJBs from start to finish. The **-all** option creates default EJB-standard and PowerTier deployment descriptors (or updates the **pt-jar.xml** file if one already exists). It also produces both an EJB-compliant JAR file and a PowerTier deployment JAR file.

More specifically, **ps-makeejb** invokes the Java and RMI compilers to create **.class** files and container-adapter code. After compilation, **ps-makeejb** invokes another tool that combines enterprise bean **.class** files into a Java Archive (JAR) file with a **.jar** extension. If your project includes session beans or BMP entity beans, you place them in the generation-directory hierarchy, and **ps-makeejb** compiles, generates container-adapter code, and JARs them as well. For more information, see the chapters on BMP entity beans and session beans in the *PowerTier Server and EJB Development Guide*.

During the development cycle, as you make changes to your application, you do not need to rebuild the entire application every time. As you work, you can use the **ps-makeejb** command to rebuild the packages that correspond to the changes you have made.

ps-makeejb and Deployment Descriptors

The **ps-makeejb** command creates an EJB 1.1-compliant XML deployment descriptor for each project. It can also produce PowerTier deployment descriptors from third-party EJB deployment descriptors. In addition, you can use this command to modify and validate existing deployment descriptors, and to migrate EJB 1.0 **.des** files to EJB 1.1 XML descriptors.

The **ps-makeejb** command creates default client and server deployment descriptors. Once you have generated the deployment descriptors, you must edit them to contain values appropriate to your application, especially if your application includes business logic in the form of custom methods, BMP entity beans, or session beans. In particular, **ps-makeejb** generates only sample entries for some of the PowerTier-specific sections, which you must edit.

Modifying deployment descriptors is a three-stage process:

1. Extract the descriptor files from the application's JAR file using **-editDD**.
2. Modify the extracted descriptor files either with a text editor or using **-createDD**.
3. Place the edited descriptor files into the application's JAR file, replacing previous versions, using **-updateDD**.

If you already have the XML files, you can skip the first stage. If you are an application deployer and have only the JAR files, you must extract the deployment descriptors before you can make any changes.

It is recommended that you run **ps-makeejb** with the **-validateDD** option after modifying deployment descriptors.

Sometimes, parts of your application may be developed separately from each other, or you acquire components from third parties. In this case, you must combine the parts to form a complete application. You can use **ps-makeejb** with the **-mergeDD** option to combine the deployment descriptors. Then use the **-ejbJar** and **-ptJar** options to create deployable JAR files.

Serializing Client Deployment Descriptors

Client deployment descriptors (**application-client.xml** and **pt-application-client.xml**) are not included in any JAR files, not even client JAR files. Instead, you should use the **-serializeClientDD** option of **ps-makeejb** to serialize client deployment descriptors. Using serialized versions of client deployment descriptors improves performance and provides a more efficient client program. Without using serialized deployment descriptors, a client program would need to include an XML parser.

It is recommended that you run **ps-makeejb** with the **-validateClientDD** option after modifying client deployment descriptors.

ps-makeejb and JAR Files

You can produce three types of JAR files using the **ps-makeejb** command: EJB-standard**,** PowerTier deployment, and client.

- An EJB-standard JAR contains the files mandated by the EJB specification: it includes bean and primary-key class implementations and deployment descriptors, and is ready for further development.
- A PowerTier deployment JAR file contains all the files that are in an EJB-standard JAR file, plus the RMI stubs and skeletons and container-adapter code and XML deployment descriptors, and is ready for deployment to a PowerTier server.
- The client-side JAR is a subset of the PowerTier deployment JAR file. It contains the Java classes and interfaces that client programs will invoke to access the beans. This includes the remote and home interfaces, the primary-key class, and client-side stubs. However, it does not contain the bean implementations, deployment descriptors, or server-side skeletons.

The **PROJECT_NAME** entry from the **ps-makeejb.cfg** file determines the name of the EJB JAR file and the PowerTier JAR file – for example **ATM.jar** and **ATM-pt.jar**.

To deploy your application, copy the generated PowerTier deployment JAR file to the pantry (which must be included in the **CLASSPATH**).

Clearing the Pantry

The Persistence pantry is the target directory for compiled EJB class files. This is where the PowerTier build process compiled the EJB source and the RMI stubs and skeletons (which allow remote invocation). The pantry is also the source directory for the components that go into the JAR file.

As you develop your application, you may find that the pantry contains old files with the same names as new files that you are creating. You can use **ps-makeejb** to remove the existing packages and files from the pantry to ensure a clean build. It is also a good idea to clean the pantry after changing your object model or debugging, before you rebuild your application.

The **ps-makeejb** command provides three options to remove old files and directories from the Persistence pantry: **-clean**, **-cleanAll**, and **-cleanCntr**. For more information, see "Using the ps-makeejb Command" on page 18.

When you build your application, if you get an error message that mentions "unimplemented" methods – and you know your source files are correct – use **ps-makeejb** to clean out your pantry and then try rebuilding.

Primary Options

The **ps-makeejb** command has a set of primary options that specify how you want to use the command. Table 34 lists these options and describes what each one does. You can use additional options with many of these primary options, to further control how the command operates.

Table 34. Primary Options for ps-makeejb

Option	Description
-all	Builds an entire project from start to finish. The **-all** option combines the processing of the **-createDD**, **-ejbJar**, and **-ptJar** options.
-clean [*packageName* ...]	Removes the subdirectories and files in the specified packages, from the pantry.
-cleanAll	Removes the entire contents of the pantry – all subdirectories and files.
-cleanCntr	Removes all of the compiled classes that represent container-specific code associated with the packages in the current project.
-clientJar *ptJar*	Builds a client-side JAR file from the PowerTier deployment JAR. The ***ptJar*** parameter must provide the absolute path to the PowerTier JAR file.
-createClientDD	Creates template J2EE and PowerTier client descriptor files.

Table 34. Primary Options for ps-makeejb (Continued)

Option	Description
-createDD [-noRecompile]	Creates default **ejb-jar.xml** and **pt-jar.xml** files for the current project, or updates the **pt-jar.xml** file if it already exists. The **ps-gen** command creates a default **pt-jar.xml** file with basic information about your CMP entity beans. You can use the **-noRecompile** switch if you know your **.class** files are up-to-date.
-editDD *jar* [...]	Unjars the XML deployment descriptors from an EJB-standard or PowerTier deployment JAR file into the current directory. This enables you to edit the XML files from a JAR file.
-ejbJar [-noRecompile]	Builds an EJB-standard JAR file.
-genImpls *beanClassName* [...]	Generates container-adapter code from one or more specified ***beanClassName*** arguments.
-mergeDD *jarFile* [...]	This option accepts a list of JAR files (EJB or PT JARs) and produces combined EJB and PowerTier deployment descriptors in the files **ejb-jar.xml** and **pt-jar.xml**.
-ptJar *ejbJar*	Builds a PowerTier deployment JAR file from the specified EJB-standard JAR**.**
-serializeClientDD	Starts by validating the J2EE (required) and PowerTier (optional) client descriptors and reporting any validation errors. If no errors are detected, **ps-makeejb** serializes the resulting descriptor objects to the **application-client.ser** file.
-updateDD *jar* [...]	Adds the updated XML files to the specified ***jar*** file. Assumes the updated XML files are in the current directory, unless you use the **-xmlDir** option.
-validateDD [*ptcFilePath*]	Use this option to validate an **ejb-jar.xml** or a **pt-jar.xml** deployment descriptor or both. If you specify the server configuration file (***ptcFilePath***) as an argument, the **-validateDD** option will perform a more comprehensive validation of your configuration, including security.
-validateClientDD	Provides standalone validation of the **application-client.xml** and **pt-application-client.xml** files. By default, **ps-makeejb** looks for these files in the current directory. You can specify a different location for both files using the **-xmlDir** option.

Using the ps-makeejb Command

The following sections describe the **ps-makeejb** command and how to use it to perform various development tasks:

- Specifying Alternate Source and Destination Directories
- Compiling and Building
- Rebuilding Parts of an Application
- Clearing the Pantry
- ps-makeejb and Deployment Descriptors
- ps-makeejb and JAR Files

Specifying Alternate Source and Destination Directories

The **ps-makeejb** command compiles **.class** files into a package hierarchy under the directory defined by the **PERSISTENCE_PANTRY** environment variable. The files in the pantry become the source files when **ps-makeejb** creates JAR files.

The **-all**, **-createDD**, **-ejbJar**, **-mergeDD**, and **-clientJar** options of the **ps-makeejb** command look for a **ps-makeejb.cfg** file in the current directory to determine the source projects and packages on which to operate. You can use the following options to specify alternate source and destination locations:

-clientOutputDir *directoryPath*	Specifies the destination of serialized client deployment descriptors (**.ser** files).
-outputJarPath *jarFilePath*	Specifies the destination of the JAR file.
-packageName *packageName*	Specifies the packages on which to operate. If you specify more than one package, use the full path name for each package and separate the entries with spaces.
-projectName *projectName*	Specifies the project on which to operate.
-xmlDir *directoryPath*	Specifies the location of the XML deployment descriptors.

Building an Entire Application

You can use the **ps-makeejb** command to build your EJB components, from start to finish. The **ps-makeejb.cfg** file (generated by **ps-gen** or created with a text editor) specifies the source packages to build.

The **-all** option compiles the Java source files in the packages listed in the **ps-makeejb.cfg** file (or specified with the **-projectName** and **-packageName** options), creates or updates deployment descriptors in the current directory, and packages the results into both an EJB-compliant JAR file and a PowerTier deployment JAR file, also in the current directory. Use this option when you do not need to edit the deployment descriptor files before testing your application.

The **-all** option combines the processing of the **-createDD**, **-ejbJar**, and **-ptJar** options. For more information about these options, see "ps-makeejb and Deployment Descriptors" on page 306 and "ps-makeejb and JAR Files" on page 307.

To use the **-all** option, follow these steps:

1. Open a PowerTier command prompt, by clicking **Start→Programs→PowerTier for J2EE for J2EE→Tools→PowerTier Command Prompt**.
2. From the PowerTier command prompt, change directories to the location of your **ps-makeejb.cfg** file. For example:
   ```
   cd %PERSISTENCE_HOME%\atm\ejb
   ```
3. Invoke **ps-makeejb** to compile your beans and build a JAR file ready to deploy to the PowerTier server:
   ```
   ps-makeejb -all
   ```
4. To deploy your application, copy the **atm-pt.jar** file to the pantry. For example:
   ```
   cp ATM-pt.jar %PERSISTENCE_PANTRY%
   ```

Passing Arguments to the Java and RMI Compilers

When you use **ps-makeejb** to compile project files, you can specify arguments to pass directly to the Java compiler or the RMI compiler. For example:

```
ps-makeejb -javaCompilerArgs "-g -verbose" -all
ps-makeejb -rmiCompilerArgs "-nostub -noskel" -all
```

You can use the **-javaCompilerArgs** option with **-all**, **-createDD**, **-ejbJar**, and **-ptJar**. You can use the **-rmiCompilerArgs** option with **-all** and **-ptJar**. When you use either the **-javaCompilerArgs** or the **-rmiCompilerArgs** option to pass arguments to a compiler, separate the arguments with spaces, and enclose the entire list in double quotation marks.

Rebuilding Parts of an Application

During the development cycle, as you make changes to your application, you do not need to rebuild the entire application every time. As you work, you can use the **ps-makeejb** command to rebuild the packages that correspond to the changes you have made.

Rebuilding parts of an application can include any of the following tasks:

- Building Specific Packages
- Adding Business Logic
- Preventing Recompilation
- Retaining Existing Source Files

Building Specific Packages

To recompile and rebuild specific packages, use the **-all** option, with the **-projectName** and **-packageName** options, to specify which packages to rebuild. For example:

```
ps-makeejb -all -projectName HRapp -packageName HR.resources HR.departments
```

Specifying these options on the command line overrides the values in the **ps-makeejb.cfg** file.

When you specify packages in this way, **ps-makeejb** creates a new project with only the packages you list and their sub-packages. Any packages you do not list are excluded from the JAR file.

Adding Business Logic

The **-all** option is useful if your project contains no custom code. If you are writing business logic – in the form of BMP entity beans, session beans, or custom methods in CMP entity beans – you should not use this option. Instead, you should run the **ps-makeejb -createDD** command to create deployment descriptors, and then use the **-ejbJar**, **-ptJar**, and **-clientJar** options to build the remaining pieces of your project.

For more information, see “Creating Deployment Descriptors” on page 315 and “ps-makeejb and JAR Files” on page 307.

Preventing Recompilation

If you have not changed your source files since the last time you compiled them, and you only want to rebuild the deployment descriptors and JAR files, you can use the **-noRecompile** option to prevent recompiling. If you use this option, ensure that the directory – most likely the pantry containing your compiled **.class** files – is in your **CLASSPATH**.

You can use the **-noRecompile** option with the **-all**, **-createDD**, and **-ejbJar** options.

Retaining Existing Source Files

By default, **ps-makeejb** deletes generated source files when you rebuild your project. If you want to avoid this, use the **-keepgenerated** option. This option retains all source files generated by **ps-makeejb**, rather than deleting or regenerating them. You can also use the **-keepgenerated** option with **-all** and **-ptJar**.

Clearing the Pantry

The **ps-makeejb** command provides three options to remove old files and directories from the Persistence pantry to ensure a clean build: **-clean**, **-cleanAll**, and **-cleanCntr**. The following sections describe these options.

Removing Compiled Code

Use the **-clean** option to remove the package subdirectories, and compiled **.class** files, associated with the current project from the pantry. You can also list specific packages to clear, rather than all of the files for the entire project.

To remove compiled code from the pantry:

1. From a PowerTier command prompt, specify one or more packages to remove from the pantry. For example:
   ```
   ps-makeejb -clean HR.departments HR.scheduling
   ```
2. At the prompt, answer yes (`Y`) to confirm that you want to remove the packages. You can use the **-force** option to disable the confirmation prompt.

Clearing the Entire Pantry

Use the **-cleanAll** option to remove all subdirectories and compiled files from the pantry. To remove everything from the **PERSISTENCE_PANTRY**:

1. From a PowerTier command prompt, enter the following command:
   `ps-makeejb -cleanAll`
2. At the prompt, answer yes (`Y`) to confirm that you want to remove the files.
   You can use the **-force** option to disable the confirmation prompt.

Removing Container-Adapter Code

Use the **-cleanCntr** option to remove all of the compiled classes that represent container-specific code associated with the packages in the current project. This option removes only the container-adapter code (the RMI/IIOP stubs and skeletons).

To clean the container-adapter code from all the packages specified in the current **ps-makeejb.cfg** file:

1. From a PowerTier command prompt, enter the following command:
   `ps-makeejb -cleanCntr`
   You can also list the names of specific packages whose container-adapter code to remove.
2. At the prompt, answer yes (`Y`) to confirm that you want to remove the files.
   You can use the **-force** option to disable the confirmation prompt.

Working with Deployment Descriptors

The **ps-makeejb** command creates default EJB 1.1-compliant and PowerTier-specific deployment descriptors for each project. In addition, it provides a number of options that enable you to modify and validate existing PowerTier-specific and client-side deployment descriptors, and to migrate EJB 1.0 **.des** files to EJB 1.1 descriptors. The following sections describe these options:

- Creating Deployment Descriptors
- Modifying Deployment Descriptors
- Merging Beans from Different Projects
- Validating Deployment Descriptors
- Serializing Client Deployment Descriptors

When **ps-makeejb** makes changes to existing XML deployment descriptors, it creates backup copies of the descriptor files in the same directory as the originals. These backup files are named **ejb-jar_old.xml** and **pt-jar_old.xml**.

Creating Deployment Descriptors

The **ps-makeejb** command has options to create default client and server deployment descriptors. You must edit the generated XML files to provide specific information about your application.

Creating Server-Side Deployment Descriptors

The **-createDD** option creates default EJB-standard and PowerTier-specific deployment descriptors for the current project (or updates them if either one already exists). This option reads the **ps-makeejb.cfg** file in the current directory, and uses the **PACKAGE_NAME** entries to determine which beans will be described in the deployment descriptors. The resulting **ejb-jar.xml** and **pt-jar.xml** files contain entries only for beans that are part of the specified packages and their sub-packages.

If you are developing BMP entity beans or session beans or deploying a JAR at production time, the **ps-makeejb.cfg** file may be unavailable. In this case, you should specify the source using the **-packageName** and **-projectName** options, or create a **ps-makeejb.cfg** file containing the project and package names.

To create deployment descriptors with default values for your application:

1. If your project contains CMP entity beans, enter the following command to generate code:
   ```
   ps-gen SampleApp.per
   ```
   This command generates a partial **pt-jar.xml** file with **ps-entity** elements for your CMP entity beans.
2. Change to the directory that contains your project file. For example:
   ```
   cd %PERSISTENCE_HOME%\SampleApp\ejb
   ```
3. Run **ps-makeejb** to generate default deployment descriptors for the specified package. For example:
   ```
   ps-makeejb -createDD -packageName com.persistence.example -projectName
   SampleApp
   ```
   This command generates the **ejb-jar.xml** file, and updates the **pt-jar.xml** file (if one exists), with entries for each enterprise bean in the **com.persistence.example** package.

4. Open the generated file **ejb-jar.xml** and change the value of the **trans-attribute** element for each of your custom methods if necessary. For more information about transaction attributes, see the chapters on transactions in the *PowerTier Server and EJB Development Guide*.

The **-createDD** option also compiles the **.class** files in the Persistence pantry. If you have not changed your source files since the last time you compiled them, you can use the **-noRecompile** option to prevent recompiling. If you use this option, ensure that the directory containing your compiled **.class** files is in your **CLASSPATH**.

The **ps-makeejb** command assumes that your enterprise beans use one of the naming conventions in Table 35when it generates entries in the EJB-standard deployment descriptor.

Table 35. Naming Conventions

Source File Type	Default Name (option 1)	Default Name (option 2)
remote interface	***Class*.java**	***Class*Remote.java**
bean class	***Class*Bean.java**	***Class*.java**
home interface	***Class*Home.java**	***Class*Home.java**

If the custom enterprise beans in your project do not follow either of these naming conventions, you just specify the names using the **ejb-class**, **remote**, and **home** elements in the EJB-standard deployment descriptor. After modifying the deployment descriptor, run **ps-makeejb -createDD** again.

Specifying Compiler Arguments

When you use the **-createDD** option, you can pass arguments directly to the Java compiler. For more information, see “Passing Arguments to the Java and RMI Compilers” on page 311.

Additional Deployment Descriptor Generation Options

You can use the following options to modify how **ps-makeejb** creates the default deployment descriptors:

-groupTxGen Groups all methods of the same bean that have the same transaction attribute type into one **container-transaction** element. This provides a more compact **ejb-jar.xml** file. (By default, **ps-makeejb** generates one **container-transaction** element per method, to simplify editing the **trans-attribute** element for each method.)

-maxGen Generates fully-expended **method-permission** entries in the **pt-jar.xml** file. By default, these entries are commented out.

For more information about the security entries in a PowerTier deployment descriptor, see "pt-jar.xml File" in the *Class Reference*.

Creating Client-Side Deployment Descriptors

The **-createClientDD** option creates template J2EE and PowerTier client descriptor files. After you generate the default files, you must populate them with relevant information about your application.

To create client-side deployment descriptors:

1. From a PowerTier command prompt, change to the directory containing your **ps-makeejb.cfg** file. For example:
   ```
   cd %PERSISTENCE_HOME%\atm\ejb
   ```
2. Enter the following command to create default client-side deployment descriptors:
   ```
   ps-makeejb -createClientDD
   ```
3. Use the options in the following table to override the default destination, or the default names, for the generated files:

-xmlDir By default, **ps-makeejb** creates the client descriptor files in the current working directory. Use the **-xmlDir** option specify a different location.

-applClientXml The default J2EE application client descriptor is named **application-client.xml**. Use the **-applClientXml** option to specify a different name for your EJB client descriptor.

-ptApplClientXml The default name of the PowerTier-specific application client descriptor is **pt-application-client.xml**. Use the **-ptApplClientXml** option to specify a different name for your PowerTier client descriptor.

Modifying Deployment Descriptors

To modify deployment descriptors, you must extract the XML descriptor files from the application's JAR file so that you can edit them. After you make your changes, you must replace the edited files in the application's JAR file.

Use the **-editDD** option to unjar the contents of an EJB-standard or PowerTier deployment JAR file into the current directory. This enables you to edit the XML files in the JAR file.

After editing the descriptors, use the **-updateDD** option to update the XML files in the original JAR file. This option looks for XML files in the current directory, or in the directory specified by the **-xmlDir** option. Then it creates a JAR file with the newly-updated XML files. This JAR file can be an EJB-standard or PowerTier-specific JAR file.

To update deployment descriptors in a PowerTier JAR file:

1. Change directories to the location of the **ps-makeejb.cfg** file. For example:
   ```
   cd %PERSISTENCE_HOME%\ATM\ejb
   ```
2. Run **ps-makeejb** to add the modified deployment descriptor to the PowerTier JAR file. For example:
   ```
   ps-makeejb -updateDD ATM-pt.jar
   ```
   You can use the **-xmlDir** option to specify a different directory for the extracted XML files.

Note: You may modify the EJB-standard and PowerTier-specific deployment descriptors after the container-adapter code has been built and packaged in the PowerTier-specific JAR file. However, if you modify these descriptors after packaging, you can change only runtime information, such as the security elements or the JNDI bean environment elements. If you change information that affects the container-adapter code, such as transaction attributes, you must run the **ps-makeejb -ptJar** command again.

3. Copy the new PowerTier JAR file (in this case, **ATM-pt.jar**) to the pantry and overwrite the JAR file from the basic application. For example, from the **ATM\ejb** directory:

   ```
   cp ATM-pt.jar %PERSISTENCE_HOME%\pantry\ATM-pt.jar
   ```

When you use **-updateDD**, **ps-makeejb** reports an error if it cannot find an **ejb-jar.xml** file, or if you specify a PowerTier JAR file and it cannot find a **pt-jar.xml** file . However, if you specify an EJB JAR file and no **pt-jar.xml** file is found, no error will be reported.

Merging Beans from Different Projects

If parts of your application have been developed separately from each other, you must combine the parts to form a complete application. You can use **ps-makeejb** with the **-mergeDD** option to combine the separate JAR files and deployment descriptors into a single deployable JAR file.

The **-mergeDD** option accepts a list of JAR files (EJB-standard or PowerTier-specific JARs) and produces **ejb-jar.xml** and **pt-jar.xml** files that contain entries combined from the original deployment descriptors.

To merge individual JAR files:

1. Place the JAR files to combine into a single directory (or specify absolute or relative path names when you enter the command). If the JAR files are in the current directory, enter the following command:

   ```
   ps-makeejb -mergeDD jarFile1 jarFile2 jarFile3
   ```

   ps-makeejb unjars the specified JAR files to the pantry, and places the resulting merged XML files either in the current directory or in the location specified by the **-xmlDir** option.
2. Use **ps-makeejb -ejbJar** and **ps-makeejb -ptJar** to build EJB-standard and PowerTier JAR files containing the newly created XML descriptors.
3. To deploy or test your application, copy the resulting ***app*-pt.jar** file to the pantry.

Validating Deployment Descriptors

You can use **ps-makeejb** to validate both server-side deployment descriptors (**ejb-jar.xml** and **pt-jar.xml**) and client-side deployment descriptors (**application-client.xml** and **pt-application-client.xml**).

Server-Side Deployment Descriptors

Use the **-validateDD** option to validate either an **ejb-jar.xml** or a **pt-jar.xml** deployment descriptor or both. You should run this option on any XML file whose text you have edited directly. When you use the **-validateDD** option, do not use any options with it except **-xmlDir**. For example:

```
ps-makeejb -validateDD -xmlDir %PERSISTENCE_HOME%\atm\XML-descriptors
```

When you use **-validateDD**, you can specify your server configuration (**.ptc**) file as an optional argument. This enables **ps-makeejb** to perform a more comprehensive validation of your configuration. It does so by validating the XML files against the **.ptc** file, and against the user-to-role and user-account configuration files that are described in the **.ptc** file.

For a detailed list of what **ps-makeejb** validates, see the validation appendix in the *PowerTier Tools User Guide*.

Client-Side Deployment Descriptors

Use the **-validateClientDD** option to perform standalone validation of your **application-client.xml** and **pt-application-client.xml** files. By default, **ps-makeejb** looks for these files in the current directory. You can specify a different location for both files using the **-xmlDir** option. For example:

```
ps-makeejb -validateClientDD -xmlDir \myProject\atm\XML-descriptors
```

By default, the J2EE application client descriptor is **application-client.xml**. You can specify a different name by using the **-applClientXml** option.

By default, the name of the PowerTier-specific application client descriptor is **pt-application-client.xml**. You can specify a different name by using the **-ptApplClientXml** option.

If **ps-makeejb** cannot find any J2EE client descriptors, it reports an error. The PowerTier-specific descriptor is optional. If you specify a PowerTier descriptor using the **-ptApplClientXml** option and **ps-makeejb** cannot find the file, it reports an error. As it validates the J2EE and optional PowerTier descriptor, **ps-makeejb** reports any errors.

Serializing Client Deployment Descriptors

Client deployment descriptors are not included in any JAR files. Instead, you should use the **-validateClientDD** option of **ps-makeejb** to validate and the **-serializeClientDD** option to serialize client deployment descriptors. Using serialized client deployment descriptors eliminates the need to include an XML parser in your client environment.

You must use the **ps.client.descriptor** property value to specify the location of the resulting **.ser** file when deploying your application client. Application developers are responsible for managing the client deployment descriptors: **ps-makeejb** does not include an equivalent to the **-editDD** option for client-side deployment descriptors.

The **-serializeClientDD** option starts by validating the J2EE (required) and PowerTier (optional) client descriptors and reporting any validation errors. If no errors are detected, **ps-makeejb** serializes the resulting descriptor objects to the **application-client.ser** file.

You can use the **-applClientXml** and **-ptApplClientXml** options to specify client deployment descriptors that do not use the default names. Those defaults are **application-client.xml** and **pt-application-client.xml**, respectively.

The default name for the serialized output file is the name of the J2EE application client XML file, with the extension **.ser**. By default, **ps-makeejb** creates this file in the current directory. You can use the **-applClientSer** option to specify another name, and the **-clientOutputDir** option to specify another location.

To create a serialized client deployment descriptor to deploy with your client program:

1. From a PowerTier command prompt, enter the following command:
   ```
   ps-makeejb -createClientDD
   ```
   This command creates default client deployment descriptors called **application-client.xml** and **pt-application-client.xml**. For more information about these files, see the corresponding pages in the "Configuration Files" part of the *Class Reference*.
2. Using a text editor, make any necessary changes to the deployment descriptor files.
3. Enter the following command to validate the modified deployment descriptors:
   ```
   ps-makeejb -validateClientDD
   ```
   The **ps-makeejb** command reports any errors it encounters.
4. Enter the following command to serialize the client deployment descriptor files:
   ```
   ps-makeejb -serializeClientDD
   ```
   This command produces a serialized deployment descriptor that combines the contents of both **application-client.xml** and **pt-application-client.xml** into the file **application-client.ser**.
5. Enter the following command to deploy the serialized client deployment descriptors:
   ```
   java -D ps.client.descriptor=myApp\myClient\application-client.ser
   ```

Creating JAR Files

You can produce three types of JAR files using the **ps-makeejb** command: EJB-standard**,** PowerTier deployment, and client.

The **PROJECT_NAME** entry from the **ps-makeejb.cfg** file determines the name of the EJB JAR file and the PowerTier JAR file – for example **ATM.jar** and **ATM-pt.jar**. By default, the client JAR for the same project is **ATM-client.jar**. You can specify a different name for your client JAR file in the **ejb-client-jar** element of the EJB deployment descriptor.

When **ps-makeejb** makes changes to existing JAR files, it creates backup copies of the descriptor files in the same directory as the originals. These backup files are named ***j*arFileName_old.jar** and ***jarFileName*-pt_old.jar**.

Creating an EJB-Standard JAR File

The **-ejbJar** option builds an EJB-standard JAR file. By default, the **ps-makeejb** command creates the JAR in the current directory. To specify a different directory, use the **-outputJarPath** option.

Assuming that your Java classes are up-to-date, and your deployment descriptors are correct and complete, follow these steps to build your application's JAR file:

1. From a PowerTier command prompt, change to the directory containing your **ps-makeejb.cfg** file. For example:
   ```
   cd %PERSISTENCE_HOME%\atm\ejb
   ```
2. Enter the following command to build the EJB-standard JAR file **ATM.jar**:
   ```
   ps-makeejb -ejbJar
   ```

Note: This EJB-standard JAR file and its contents fully comply with the EJB specification, although the beans it contains will only work in the PowerTier container.

If you are developing BMP entity beans or session beans, or deploying at production time, there may be no **ps-makeejb.cfg** file available. In this case, you should specify the inputs using the **-packageName** and **-projectName** options.

If you have not changed your source files since the last time you compiled them, you can use the **-noRecompile** option to prevent recompiling. If you use this option, ensure that the directory containing your compiled **.class** files is in your **CLASSPATH**. By default this is the directory defined by the **PERSISTENCE_PANTRY** environment variable.

Specifying Compiler Arguments

When you use the **-ejbJar** option, you can pass arguments directly to the Java compiler. For more information, see "Passing Arguments to the Java and RMI Compilers" on page 311.

Creating a PowerTier Deployment JAR File

The **-ptJar** option builds a PowerTier deployment JAR file from an EJB-standard JAR file. You must specify the absolute path to the source JAR file. When you build the PowerTier JAR file, the PowerTier tools generate the container-adapter code (the RMI stubs and skeletons) for your custom methods, and compile your additions together with the rest of the project.

To create a PowerTier deployment JAR file:

1. From a PowerTier command prompt, change to the directory containing your **ps-makeejb.cfg** file. For example:
   ```
   cd %PERSISTENCE_HOME%\atm\ejb
   ```
2. Enter the following command to build the PowerTier JAR file **ATM-pt.jar**:
   ```
   ps-makeejb -ptJar ATM.jar
   ```
3. To test your application, copy the PowerTier JAR file to the pantry. For example:
   ```
   cp ATM-pt.jar %PERSISTENCE_PANTRY%
   ```

Adding Session Beans or BMP Beans

When you add session beans or BMP beans to your application, you must generate container-adapter and RMI stubs and skeletons and add them to the rest of the project in your PowerTier JAR file. To do this, run the **ps-makeejb** command with the **-ptJar** option.

When your project includes BMP entity beans or session beans, you must add them before you generate deployment descriptors. You must edit the default deployment descriptors and include the class files for these beans in your JAR file. For more information, see the chapters on BMP entity beans and session beans in the *PowerTier Server and EJB Development Guide*.

Specifying Compiler Arguments

When you use the **-ptJar** option, you can pass arguments to both the Java and the RMI compilers. For more information, see "Passing Arguments to the Java and RMI Compilers" on page 311

Creating a Client JAR File

The **-clientJar** option builds a client-side JAR file from the PowerTier deployment JAR. The client-side JAR file contains the class files that clients use to access the enterprise beans specified in **ejb-jar.xml**.

To build a client-side JAR file:

1. From a PowerTier command prompt, change to the directory containing your **ps-makeejb.cfg** file. For example:
   ```
   cd %PERSISTENCE_HOME%\atm\ejb
   ```
2. Enter the following command to build the client JAR file **ATM-client.jar**:
   ```
   ps-makeejb -clientJar ATM-pt.jar
   ```
   The ***ptJar*** parameter must provide the absolute path.

This option uses the **PACKAGE_NAME** entries from the **ps-makeejb.cfg** file to determine which beans from the ***ptJar*** file to include in the client JAR file. Only beans that are part of the specified packages and their sub-packages are included. You can also use the **-packageName** and **-projectName** options to specify which packages to include.