4

PowerTier Web Development Tools

This chapter describes the process of developing J2EE applications with Web components, and introduces the PowerTier tools you use at each stage of the development process.

This chapter contains the following sections:

Prerequisites

A basic understanding of the components that can make up a J2EE Web application, as introduced in Chapter 3, “Web Application Structure.”

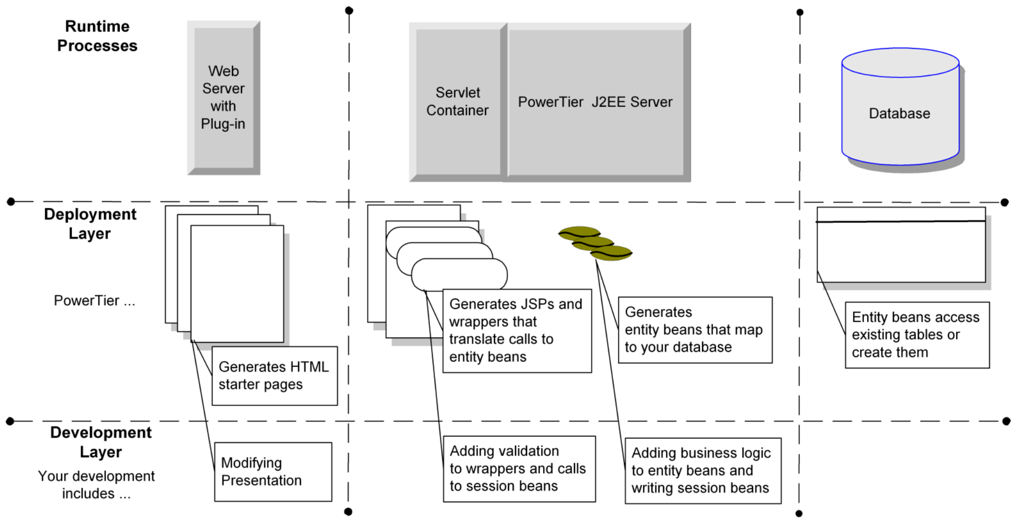
The Development Process

When you generate a PowerTier object model, you can use PowerPage to generate a Web layer that provides HTML access to your CMP entity beans. The generated Web files include the following:

- JavaServer Pages (JSPs) – Web files that contain HTML code that determines presentation, and Java code that creates dynamic content. For more information, see "JavaServer Pages" on page 32.
- HTML files that contain links to the JSPs.
- Wrapper classes – Java classes that run in the servlet container and translate requests and responses between the JSPs and EJBs.

Figure 3 shows the pieces of a Web-based application, and identifies what PowerTier generates for you and what you are responsible for creating or modifying.

Figure 3. Web-Based Development Responsibilities


Runtime Processes
Web Server with Plug-in
Servlet Container
PowerTier J2EE Server
Database
Deployment Layer
PowerTier ...
Generates HTML starter pages
Generates JSPs and wrappers that translate calls to entity beans
Generates entity beans that map to your database
Entity beans access existing tables or create them
Development Layer
Your development includes ...
Modifying Presentation
Adding validation to wrappers and calls to session beans
Adding business logic to entity beans and writing session beans

The development steps for the initial version of the Web layer are:

1. Obtain or create a PowerTier object model.
2. Generate project classes using **ps-gen** with PowerPage.

PowerPage produces the following directory structure, where Web components are highlighted in bold:

```
Project.per
generationDirectory\
    ps-makeejb.cfg
    ejb-jar.xml
    pt-jar.xml
    package\
        beanCode.java
        presentation\
            wrapperCode.java     →helper classes
    jsp\
        project\
            beanPages.htm        →static pages
            bean-JSPs.jsp        →JSPs
            images\
```

3. Configure the Web application (by editing XML files, adding additional files, etc.) as necessary.

4. Run the following command to build the EJB project files:

```
ps-makeejb -all
```

5. Run the following commands to create an open directory structure for the generated Web components:

```
cd generationDirectory
ps-makeweb –webApp web -misc jsp\project -classes presentation
```

This produces the following structure:

```
Project.per
generationDirectory\
    ps-makeejb.cfg
    ejb-jar.xml
    pt-jar.xml
    package\
        beanCode.java
    web\
        beanPages.htm            →static pages
        bean-JSPs.jsp            →JSPs
        images\
        WEB-INF\
            web.xml
            classes\
                wrapperCode.java →helper classes
            lib\
```

For more information about this structure, see "Web Application Structure" on page 40.

6. Run the following command on the Web application files to create a WAR file and copy it to the Web pantry.

```
ps-makeweb -all -updatePantry
```

7. To deploy PowerTier Web components to the Web container, you must install your Web application to the Web server plug-in and at least one servlet container.

   For information, see "Installing Web Applications" on page 51.

The *QuickStart Tutorial* walks you through these preliminary steps, and describes the generated files. In just minutes, PowerTier creates a working application that you can test and use as a proof of concept. During iterative development, you can add validation logic to the wrapper classes. Should you need to regenerate code, PowerTier preserves any code you added to the wrapper classes.

You can also add business logic to the EJB layer in the form of custom entity bean methods or session beans. Using the generated JSPs and wrapper classes as an example, you can add calls to custom methods or session beans.

Development and deployment of a Web-based PowerTier application can include tasks for Web designers, client developers, bean developers, server developers, and system administrators. Your organizational structure and application requirements will determine who is responsible for which task.

Managing Changes to Generated JSPs and HTML

PowerPage provides code insertion points in the wrapper classes. When you regenerate your object model, PowerPage preserves any code in those insertion points. The generated HTML or JSP pages do not provide code insertion points, since they are intended only as examples. For efficiency, you may want to postpone changes to the HTML files and JSPs until the object model is in its final form. However, you must involve the Web designer from the start of development, to make sure that the application logic supports the desired functionality.

Compiling and Packaging EJB Components with ps-makeejb

When you use PowerTier PowerPage to generate JavaServer Pages, PowerPage creates a ***Class*.jsp** for each class defined in the object model. The ***Class*.jsp** allows an HTML client to access and update PowerTier CMP entity beans of type ***Class***. When the Web browser invokes the ***Class*.jsp** file, the JSP sends a request to the ***Class*Wrapper** bean, which runs in the JSP/servlet container.

You use the **ps-makeejb** command to compile the components in the EJB layer of your Web application, including wrapper classes generated by PowerPage. The **ps-makeejb** command:

- compiles Java source files
- generates container-adapter code (the RMI stubs and skeletons that enable remote communication)
- generates starter deployment descriptors
- packages the deployment descriptors and compiled EJBs into a JAR file for deployment

The helper classes generated by PowerPage have dependencies on the EJB bean classes in your object model. Before you can run **ps-makeweb** to construct and package the generated components into a Web application, you must run the **ps-makeejb** command. Running **ps-makeejb -all** places the compiled bean class files into the EJB pantry, which is on the system **CLASSPATH**. This ensures that all EJB bean class dependencies will be resolved when you compile your Web application.

For more information about the **ps-makeejb** command, see the *Tools Guide*.

Constructing Web Applications with ps-makeweb

The **ps-makeweb** command creates PowerTier Web applications. You can use **ps-makeweb** to extract the contents of a standard WAR file, create deployment descriptors, and package the results into a PowerTier WAR file for deployment. The **ps-makeweb** command provides options to:

- create a skeleton Web application, as defined in "Constructing a Web Application" on page 40
- construct a Web application from a list of files
- create and update XML deployment descriptors
- compile Web application source (**.java**) files
- package Web components into a PowerTier WAR file
- copy a PowerTier WAR file to the Web pantry

Web Application Structure

To deploy a Web application, you can use **ps-makeweb** to package your Web components in a WAR file or to place them in an “open” directory structure. If you start with a WAR file, then the Web administration tools **ps-webadm** and **ps-webgui** will expand the contents when you install the application to a servlet container. For more information, see “Expanding a WAR File to the Web Pantry” on page 52.

The following example shows a Web application in an open directory structure:

```
webAppDir\                → Web application context-root directory
    *.jsp                 → Web application files
    *.html                → static HTML pages
    images\*.gif          → image files
    *.class               → client-side applets, beans, classes
    WEB-INF\
        web.xml           → standard Web deployment descriptor
        ptwar.xml         → PowerTier Web deployment descriptor
        lib\*.jar         → libraries in JAR files
        classes\…\*.class → servlets and helper classes
```

The **WEB-INF** directory contains Java classes and configuration information for a Web application, including deployment descriptors. This directory is analogous to a JAR file's **META-INF** directory—it contains metainformation about the Web application’s contents. The **lib** subdirectory contains classes stored in JAR files. The **classes** subdirectory contains the class files for this Web application's servlets and support classes.

Constructing a Web Application

To create an open directory structure, **ps-makeweb** starts with a *skeleton Web application*—an empty Web application with the following directory structure:

```
webAppDir\
    WEB-INF\
        classes\
        lib\
        web.xml
```

In a skeleton Web application, the **classes** and **lib** subdirectories contain no files, and the **web.xml** deployment descriptor contains an empty **web-app** element. To create a skeleton Web application in a specific directory, use the following command:

```
ps-makeweb -webApp webAppDir
```

A skeleton application by itself is not all that useful. Therefore, you can also use **ps-makeweb** with the **-webApp** option to populate a Web application's open directory structure with files that reside elsewhere in your file system. To do this, you specify the files or directories that are part of that application.

For example, suppose you start with the following files and directories:

```
MyAppDir\
    lib.jar
    index.html
    help.html
    style.css
    dir1\
        images\
            hello.gif
            persistence.jpg
    dir2\
        classes\
            servlet1.class
            servlet2.class
            servlet3.class
    jsps\
        jsp1.jsp
        jsp2.jsp
        jsp3.jsp
    resources\
        web.xml
        ptwar.xml
        serverside_resource
```

You can use the **-webApp** option of **ps-makeweb** to place these files into the open directory structure of a Web application. The **-webApp** option works with the following additional options, to specify the types of files that contain your Web components:

-webinf	Specifies application resources to copy into the **WEB-INF** directory.
-misc	Specifies files and directories to copy into the application's root directory—the directory you specified with the **-webApp** option—by default, the current directory.
-classes	Specifies files and directories to copy to the Web application's **classes** directory. If **ps-makeweb** finds JAR files in the **classes** directory, it moves them to the **lib** directory.

You can use each of these options to specify both file and directory names. For each option, you can specify one or more files and directories. For example, you can combine these options to create a Web application as follows, starting in **MyAppDir**:

```
ps-makeweb -webApp webAppDir
           -webinf resources
           -misc jsps index.html help.html style.css dir1
           -classes lib.jar dir2
```

This command produces a Web application with the following open directory structure, where **webAppDir** is the context root:

```
webAppDir\
    jsp1.jsp
    jsp2.jsp
    jsp3.jsp
    index.html
    help.html
    style.css
    images\
        hello.gif
        persistence.jpg
    WEB-INF\
        web.xml
        ptwar.xml
        server_side_resource
        lib\
            lib.jar
        classes\
            servlet1.class
            servlet2.class
            servlet3.class
```

When you construct a Web application using the **-webApp** option, if you do not specify a **web.xml** file, then **ps-makeweb** creates one. To do this, **ps-makeweb** starts with the default **web.xml** deployment descriptor supplied with your PowerTier installation, and adds **servlet** and **servlet-mapping** entries, as described in "Updating Deployment Descriptors" on page 43.

ps-makeweb and Deployment Descriptors

Web applications use two deployment descriptors:

- **web.xml** – 2EE-standard Web application deployment descriptor
- **ptwar.xml** – PowerTier-specific Web deployment descriptor

The **ps-makeweb** command has two options that work with deployment descriptors: **-all** and **-updateDD**. By default, when you use these options, **ps-makeweb** performs strict validation of elements in XML files against the corresponding DTD (document type definition). To prevent this validation, use the **-noValidate** option in addition to **-all** or **-updateDD**. The following sections further describe the **-all** and **-updateDD** options.

Extracting and Merging Deployment Descriptors

When you extract Web components from an EAR or WAR file using the **ps-deploy** command, **ps-deploy** expands the WAR file and merges the default **web.xml** file with the existing application's **web.xml** deployment descriptor.

The PowerTier installation process adds configuration information for the JSP engine to the default **web.xml** file. The default **ptwar.xml** file has all of the elements commented out. You must edit this file and add specific information about your application.

Updating Deployment Descriptors

You can use the **-updateDD** option of **ps-makeweb** to update **servlet** and **servlet-mapping** elements of the **web.xml** deployment descriptor. **ps-makeweb** updates these elements for any *precompiled* servlets in the **classes** directory or in JAR files in the **lib** directory of your Web application.

The **-updateDD** option uses the following syntax to identify servlet mappings:

```
packageName.className
```

For example, suppose you have a servlet class called **Hello.class** in the Java package **com.presistence**, in your Web application's **classes** directory. **ps-makeweb** adds the following entries to your **web.xml** file:

```
<servlet>
    <servlet-name> com.persistence.Hello </servlet-name>
    <servlet-class> com.persistence.Hello </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name> com.persistence.Hello </servlet-name>
    <url-pattern> com.persistence.Hello </url-pattern>
</servlet-mapping>
```

To simplify addressing, you can change the generated **url-pattern** element. So that you do not have to use the complete URL **http://*host*:*port*/*context*/com.persistence.Hello** to refer to the **Hello** servlet, you could change the **servlet-mapping** element in this example as follows:

```
<servlet-mapping>
    <servlet-name> com.persistence.Hello </servlet-name>
    <url-pattern> Hello </url-pattern>
</servlet-mapping>
```

Before you change **servlet-mapping** entries for pre-packaged servlet classes, be sure that all of your servlets have unique names. If servlets with the same name appear in different packages, the servlet container only recognizes the first servlet with each name.

The Web Pantry

You deploy Web components to the Web pantry—the location of expanded WAR files. Both the Web server plug-in and PowerTier servlet containers look for application components in this location. The **PERSISTENCE_WEB_PANTRY** environment variable specifies the location of the Web pantry. If this variable is not defined, the Web container uses the **web\apps** subdirectory of your PowerTier installation.

Note: By default, **PERSISTENCE_WEB_PANTRY** is not defined. For security reasons, it is important that you do not include the Web pantry in the **CLASSPATH**.

Compiling and Packaging Web Components

In addition to merging deployment descriptors, the **-all** option of **ps-makeweb** expands a standard WAR file to an open directory structure, compiles Java files in the **classes** directory, and packages the result in a PowerTier-specific WAR file with the name ***myWebApp*-pt.war**.

You should not use the **-all** option on a PowerTier WAR file. Instead, use **ps-deploy** to expand the WAR file first, modify any files that must be changed (such as **web.xml** and **ptwar.xml**), and then run **ps-makeweb -all** using the open directory structure to rebuild the PowerTier WAR file. The new file will retain the name ***myWebApp*-pt.war**.

If you modify the XML deployment descriptors after using the **-all** command, then you should run **ps-makeweb -all** again to recreate the PowerTier WAR file. If you also use the **-updatePantry** option, then **ps-makeweb** copies the PowerTier WAR file to the Web pantry in preparation for deployment to a servlet container.

When **ps-makeweb** copies a PowerTier WAR file to the pantry, it does not expand the contents of the archive. When you use **ps-webadm** or **ps-webgui** to install a Web application to a servlet container, the administration tools unjar the WAR file as part of the deployment process (for details see "Expanding a WAR File to the Web Pantry" on page 52). If the Web server is running when you deploy a Web application to a servlet container, you must restart the Web server before it recognizes the newly-installed application.

When you use the **-updatePantry** option of **ps-makeweb** to copy a WAR file to the Web pantry, the servlet container finds the file and expands it. In order for the Web server to see the changes to the application, you must use the administration tools to redeploy the WAR file to the Web server plug-in. The plug-in does not dynamically recognize changed WAR files. Until you use the administration tools to deploy the application to the plug-in, the static files (such as HTML and graphics) are not copied to the Web server's document root directory. You can redeploy a Web application without removing it first.

Extracting Enterprise Archive Files with ps-deploy

You use the **ps-deploy** command at the end of the development process when you have application components packaged in EAR, JAR, and WAR files that you need to modify. These components can come from other development groups, and can even come from other vendors. When you receive these files, you need to extract their contents so that you can customize the deployment descriptors and configuration files to reflect your runtime environment.

The **ps-deploy** command extracts the contents of one or more specified EAR, JAR, or WAR files to an output directory structure and creates default deployment descriptors, or updates existing ones. **ps-deploy** creates a separate directory for each module described in the application deployment descriptor (**application.xml**) of an EAR file.

Unless you specify otherwise (using the **-extractOnly** option), **ps-deploy** creates deployment descriptors and PowerTier configuration files (or updates existing ones) with default entries for each extracted component. Based on the type of module, **ps-deploy** calls **ps-makeejb** or **ps-makeweb** to create these descriptors, as follows:

- For EJB components extracted from JAR files, **ps-deploy** calls **ps-makeejb -createDD**.

- For Java client modules extracted from JAR files, **ps-deploy** calls **ps-makeejb -createClientDD**.
- For Web components extracted from WAR files, **ps-deploy** calls **ps-makeweb -all**.

Output Directory Structure

When **ps-deploy** extracts EAR, JAR, or WAR files, it derives the names of the output directories from the names of the original archive files. For example, if you have a JAR file called **MyApp\atm\ATM.jar**, then **ps-deploy** creates a project directory called **ATM**. Unless you specify otherwise (using the **-outputDir** option), the root of the extracted files is the current directory. For an example of the generated directory structure, see the *Reference Guide*.

Repackaging Components for Deployment

Once you have extracted the contents of EAR, JAR, and WAR files and updated the necessary configuration files, you can use the **ps-makeejb** and **ps-makeweb** commands to repackage application components for deployment. Because the PowerTier server does not recognize EAR files in this release, you must package EJB components and Java client programs into JAR files, and package Web components into WAR files. If you want to create EAR files from your application components, you must use Sun's command-line tools.

If you have modified the generated deployment descriptors and configuration files, Persistence recommends that you use the validation options of **ps-makeejb** and **ps-makeweb** before repackaging your components.

To repackage your components, use the following commands:

- For EJB components extracted from JAR files, use either:
  - **ps-makeejb -ejbJar** and **ps-makeejb -ptJar**, or
  - **ps-makeejb -all**
- For Java client modules extracted from JAR files, use **ps-makeejb -serializeClientDD**.
- For Web components extracted from WAR files, use **ps-makeweb -all**.

Administering Web Applications with ps-webadm

You use **ps-webadm** to administer Web components within a single PowerTier installation. You can create servlet containers in any directory, but **ps-webadm** can only fully administer containers in the **web\se** directory tree in your PowerTier installation.

If your application includes servlet containers that are not installed in the **web\se** directory tree or that run collocated with the J2EE server, you can only use **ps-webadm** to stop these containers, install or remove Web applications to or from them, and delete them.

The **ps-webadm** command lets you install a Web application and control the life cycle of standalone and collocated servlet containers. **ps-webadm** runs in two modes:

- Directly from the command line, where you specify all of the necessary options.
- As an interactive tool with an ASCII character-based interface.

  To use the ASCII interface, run **ps-webadm** with no options (except **-?**, **-help**, or **-externalJre**).

Figure 4 shows the main menu of **ps-webadm**'s ASCII interface.

Figure 4. The Main ps-webadm Menu

```
***********************************************
Welcome to the PowerTier Servlet Engine Manager:
***********************************************


The Web server is not currently running.

1) Administer the Plugin (Available for the Apache Web Server only)
2) Administer a PT Servlet Container
3) Administer a Web Application
4) Quit

Selection:
```

The **ps-webgui** command provides parallel functionality, using a graphical user interface rather than an ASCII interface, with the following differences:

- Only **ps-webadm** can:
  - Start, stop, restart, or show the status of servlet containers not installed in the **web\se** directory tree.
  - List installed Web applications or listen ports for servlet containers not installed in the **web\se** directory tree.
- Only **ps-webgui** can:
  - Show properties files of servlet containers in the **web\se** directory tree.

- Show log files of servlet containers in the **web****se** directory tree.

For more information, see "Administering Web Applications with ps-webgui" on page 54.

You can use **ps-webadm** to perform the following functions. Except where noted, you can do each of these using either the command line or the ASCII interface:

- Servlet Container Administration Tasks
  - Create a servlet container that can run standalone or collocated with a PowerTier J2EE server.
  - Start, stop, and restart the Apache, IIS, or iPlanet Web server and standalone servlet containers.
  - Delete a servlet container (standalone or collocated).
- Web Application Installation Tasks
  - Deploy a Web application to the Web server plug-in.
  - Deploy a Web application to an existing servlet container.
  - Deploy a Web application to the Web server plug-in and to all servlet containers in the **web****se** directory tree.
  - Uninstall a Web application from one or more servlet containers and the Web server plug-in.
  - Uninstall a Web application from the Web server plug-in and all servlet containers in the **web****se** directory tree.
- Servlet Container Monitoring Tasks

  The following functions are only available using the ASCII interface, not from the command line.
  - List the Web applications installed for a Web server and one or more servlet containers.
  - List the "listen" port for one or more servlet containers.

ps-webadm.properties File

ps-webadm and **ps-webgui** both use a file called **ps-webadm.properties** to store parameters about the administration tools themselves. You can find this file in the **config** directory of your PowerTier installation. This file contains information including:

- The location of your log file, and whether you have enabled logging.
- Your Web server's document root directory.
- The location of your default Web deployment descriptor files.
- How often **ps-webgui** should refresh its internal components table.

Creating Servlet Containers

To create a servlet container, you use the **-createSe** option of **ps-webadm**. You must specify the name of the new container and its port number. Each servlet container on a single node must have a unique name (even if the containers reside in different directories) and a unique port number.

When you use this option, **ps-webadm** does the following:

1. Creates a directory for the new servlet container with the container name you specified. You can run this container standalone or in-process with a PowerTier J2EE server.

   If you specified a path, **ps-webadm** creates the servlet container in that directory. If not, **ps-webadm** creates the servlet container in the default directory, **web\se**, in your PowerTier installation.
2. Copies the default **ps-se.properties.in** and **ps-startup.in** files (from the **web\templates** directory of your PowerTier installation) to the new container's directory.
3. Renames **ps-se.properties.in** to ***containerName*.properties** and adds properties specific to the new container (such as the name and port number) to the properties and startup files.
4. Configures a context for this container, called /**servlet**.

Once your servlet container has been created, you need to configure the Web server plug-in with information about the new container. To do this, you add a **MillSE** record to the **ps-pi.conf** file.

If you plan to run your new container collocated with a PowerTier J2EE server, you must add a **JSPServletEngine** element to the server's **.ptc** file.

Note: The command-line option **-createSe** does not configure servlet containers for auto-start. To create a servlet container to use auto-start, you must use **ps-webadm**'s ASCII interface or **ps-webgui**'s graphical interface.

Starting, Stopping, and Restarting the Web Container

The **-start**, **-stop**, and **-restart** options of **ps-webadm** let you change the state of standalone servlet containers and the Web server plug-in. You can use these options to change the state of different parts of the Web container, as follows:

- the Web server plug-in
- one or more individual servlet containers
- all servlet containers in the **web\se** directory tree
- all servlet containers and the Web server plug-in together

To start or stop a collocated servlet container, use the Command Center to start or stop the associated PowerTier J2EE server.

Deleting Servlet Containers

To delete a servlet container, you use the **-deleteSe** option of **ps-webadm**. When you use this option, **ps-webadm** deletes the specified servlet container and the contents of the directory where it is located.

Warning: **ps-webadm** does not ask for confirmation before deleting a servlet container. Use the **-deleteSe** option with caution.

When you delete a servlet container, you must also:

- Reconfigure or remove any PowerTier server that uses the container.

  To do this, you remove the **JSPServletEngine** element from the server's **.ptc** file and restart the server.
- Reconfigure the Web server plug-in, if needed, so that it does not attempt to route requests to a nonexistent servlet container.

  To do this, you comment out or remove the **MillSE** record associated with the deleted servlet container from the **ps-pi.conf** file, and then restart the Web server.

Note: The command-line option **-deleteSe** does not delete a servlet containers that has been configured for auto-start. To delete a servlet container that uses auto-start, you must use **ps-webadm**'s ASCII interface or **ps-webgui**'s graphical interface.

Installing Web Applications

Installing Web components to the Web container gives your application access to EJBs in the J2EE server, via the servlet containers. The Web server then makes these EJBs available to Web-based clients on local hosts or over the Internet.

You can use both Web container administration tools—**ps-webadm** and **ps-webgui**—to install Web applications. If you use the command-line option **-install** with **ps-webadm**, use the **-pi** option to install an application to the Web server plug-in and the **-se** option to install an application to one or more servlet containers. In either case, you specify a context for the application, which becomes the URI.

Note: You can install a Web application to the Web server plug-in and one or more servlet containers at the same time, by using the **-pi** and **-se** options on the same command line.

Installing to the Web Server Plug-In

When you install a Web application to the Web server plug-in, the administration tool does the following:

1. Copies the PowerTier WAR file to the Web pantry (if it is not already there) and expands it (if it is not already expanded). For more information, see "Expanding a WAR File to the Web Pantry" on page 52.
2. Copies all Web components that are not in the **WEB-INF** or **META-INF** directory to the Web server's document root directory.
3. Adds **MillMount** directives to the plug-in's configuration file (**ps-pi.conf**) for the specified Web application.

Installing to a Servlet Container

When you install a Web application to a servlet container, the administration tool does the following:

1. Copies the PowerTier WAR file to the Web pantry (if it is not already there) and expands it (if it is not already expanded). For more information, see "Expanding a WAR File to the Web Pantry" on page 52.
2. Modifies the servlet container's properties file (***containerName*.properties**) to support this Web application.
3. Modifies the **MILLSE** directive in the Web server plug-in's configuration file (**ps-pi.conf**) corresponding to the servlet container where you installed the application, to inform the plug-in about the installed application.

Expanding a WAR File to the Web Pantry

When the administration tools expand a WAR file to the Web pantry, they perform the following steps:

1. Remove the initial slash (/) from the context name.
2. If the rest of the context name contains slashes, replace each one with the **context-delimiter** character defined in **ps-webadm.properties**. (See "Ensuring Unique Context Names" on page 52.)

   This new name is referred to as the *normalized context name*. For example, if you specify a context of /**abc**/**xyz**, the normalized name is **abc#xyz**.
3. Create a directory in the Web pantry using the normalized context name; for example, **PERSISTENCE_HOME**\web\apps\abc#xyz.
4. Expand the WAR file into this new directory.
5. If the Web server plug-in supports this context, copy static files from the expanded WAR file in the Web pantry to the Web server's document root directory.

   The default value of the document root directory is **PERSISTENCE_HOME**\apache\htdocs. You can change this using the **document-root** property in the **ps-webadm.properties** file.

Ensuring Unique Context Names

The **ps-webadm.properties** file contains a property called **context-delimiter**. This property defines a string delimiter, with a default value of **#**. **ps-webadm** uses this delimiter to collapse the context root specified with the **-install** option into a single directory. This helps guarantee that later on when the generated PowerTier WAR file is expanded in the Web pantry, contexts such as /**abc** and /**abc**/**xyz** will not be expanded into the same directory (which would be a huge security risk and a violation of the servlet specification, since /**abc** would be able to access /**abc**/**xyz**'s files).

For example, suppose you run **ps-webadm** with the following options (and **context-delimiter** is either not specified or set to **#**):

```
ps-webadm –installAll webapp-pt.war –contextroot /abc/xyz
```

ps-webadm adds the following line to the **ps-pi.conf** file:

```
MillMount /abc/xyz abc#xyz
MillSE containerName host port /abc/xyz  #for each servlet container
```

ps-webadm also adds the following lines to the ***containerName*.properties** file for each of the servlet containers installed in the **web****se** directory tree:

```
contexts=/abc/xyz
/abc/xyz.war=PERSISTENCE_HOME/web/apps/webapp-pt.war
/abc/xyz.path=abc#xyz
```

The following names are invalid context names:

- **privatefs** – The **privatefs** directory is created by the first servlet container to run in your environment. All servlet containers then use this location as a file system to maintain all HTTP session objects.
- **sharedlib** – Contains the shared library files for the Servlet Engine, which can be dynamically reloaded from this location.
- **defaultapp** – This is a reserved context name that implements the default context defined in the Servlet 2.2 specification.

ps-webadm displays an error message if you try to install a Web application using one of these names as the context.

Removing Web Applications

Use the **-remove** and **-removeAll** options to remove a Web application that you previously deployed to a servlet container or the Web server plug-in. To remove a Web application, you specify its context. You can also specify from which parts of the Web container you want to remove the application:

- the Web server plug-in
- one or more standalone or collocated servlet containers
- all servlet containers in the **web****se** directory tree
- all servlet containers and the Web server plug-in together

When you remove a Web application, the Web administration tools delete all files associated with that application from the directories associated with the Web container parts you specify.

To Reviewers:

Monitoring Servlet Containers

You must use **ps-webadm**'s ASCII interface to monitor running servlet containers. The following options are not available from the command line.

- List the installed Web applications for the Web server and one or more servlet containers.

  The list of installed servlet containers shows the following information for each one:

  - Name (and port number)
  - Status (running or stopped)
  - Web Applications
- List the "listen" port for one or more servlet containers.

Administering Web Applications with ps-webgui

The **ps-webgui** command does the same things as **ps-webadm**—using a graphical user interface instead—with the following differences:

- Only **ps-webgui** can:
  - Show properties files of servlet containers in the **web\se** directory tree.
  - Show log files of servlet containers in the **web\se** directory tree.

- Only **ps-webadm** can:
  - Start, stop, restart, or show the status of servlet containers not installed in the **web\se** directory tree.
  - List installed Web applications or listen ports for servlet containers not installed in the **web\se** directory tree.

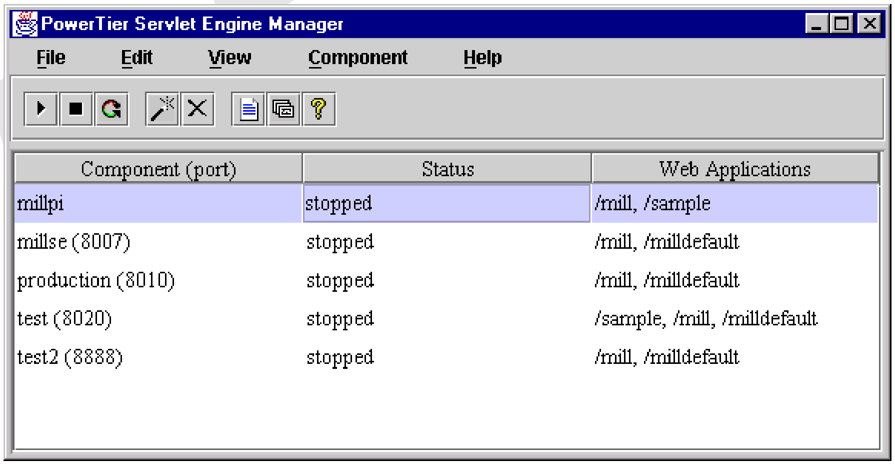
You can create servlet containers in any directory, but **ps-webgui** can only fully administer containers in the **web\se** directory tree in your PowerTier installation. For more information, see "Administering Web Applications with ps-webadm" on page 47.

ps-webgui uses the file **ps-webadm.properties**—the same properties file **ps-webadm** uses. For more information, see "ps-webadm.properties File" on page 48.

The **ps-webgui** command lets you install a Web application and control the life cycle of standalone and collocated servlet containers. Figure 5 shows the main window of **ps-webgui**. In this window, you see a list of servlet containers. For each container, the list shows:

- Name (and port number)
- Status (running or stopped)
- Web Applications

Figure 5. The ps-webgui Main Window



Component (port)	Status	Web Applications
millpi	stopped	/mill, /sample
millse (8007)	stopped	/mill, /milldefault
production (8010)	stopped	/mill, /milldefault
test (8020)	stopped	/sample, /mill, /milldefault
test2 (8888)	stopped	/mill, /milldefault

Related Information

See the following sources for related information:

Subject	Location
PowerTier Servlet Engine features	Chapter 2, “Introducing the Web Container”
Web components in a J2EE application	Chapter 3, “Web Application Structure”