SYBASE®

User's Guide

PowerBuilder Application Server Plug-In

1.0

DOCUMENT ID: DC00401-01-0100-01

LAST REVISED: November 2006

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Subject

This book contains information about installing and configuring the PowerBuilder® Application Server Plug-In, and developing PowerBuilder components and clients.

Audience

This book is for anyone responsible for installing or configuring the server plug-in, or for creating and deploying components and clients.

How to use this book

Chapter 1, "Installing and Configuring the PowerBuilder Server Plug-In," contains instructions for installing the PowerBuilder Application Server Plug-In and configuring it for your application server.

Chapter 2, "Creating Embedded Installations," explains how to install the PowerBuilder Application Server Plug-In silently.

Chapter 3, "Developing PowerBuilder Components," describes how to develop and deploy PowerBuilder components.

Chapter 4, "Developing PowerBuilder Clients," contains information about developing PowerBuilder clients and using a proxy server.

Related documents

PowerBuilder documentation set The PowerBuilder documentation set is available on the Sybase Product Manuals Web site at http://infocenter.sybase.com/help/index.jsp.

Conventions

The formatting conventions used in this manual are:

Formatting example	To indicate
commands and methods	When used in descriptive text, this font indicates keywords such as: • Command or property names used in descriptive text • Method or class names used in descriptive text
variables or *files*	Italic font indicates: • Program variables, such as *myCounter* • Parts of input text that must be substituted, for example: `Server.log` • File names
File | Save	Menu names and menu items are displayed in plain text. The vertical bar shows you how to navigate menu selections. For example, File | Save indicates "select Save from the File menu."

Formatting example	To indicate
`configure`	Monospace font indicates: • Information that you enter in the user interface, on the command line, or as program text • Example program fragments • Example output fragments

Other sources of information

Use the Sybase Getting Started CD, the SyBooks™ CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.
- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://sybooks.sybase.com/nav/base.do.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ **Finding the latest information on product certifications**

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Select Products from the navigation bar on the left.

3 Select a product name from the product list and click Go.

4 Select the Certification Report filter, specify a time frame, and click Go.

5 Click a Certification Report title to display the report.

❖ **Creating a personalized view of the Sybase Web site (including support pages)**

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1 Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.

2 Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ **Finding the latest information on EBFs and software maintenance**

1 Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.

2 Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.

3 Select a product.

4 Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.

5 Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Accessibility features

PowerBuilder has been tested for compliance with U.S. government Section 508 Accessibility requirements. The online help for this product is also provided in Eclipse help formats, which you can navigate using a screen reader.

Note You may need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For additional information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

CHAPTER 1

Installing and Configuring the PowerBuilder Server Plug-In

This chapter describes how to install and configure the PowerBuilder application server plug-in, which runs in the following application servers:

- JBoss 4.0.4
- WebLogic 9.2
- WebSphere 6.1

The server plug-in is supported for PowerBuilder version 10.5.1 and later. If PBVM patches that address issues with your application server become available, apply them to the PBVM.

This version of the PowerBuilder Application Server Plug-In is available on Windows platforms only.

Throughout this book, the PowerBuilder Application Server Plug-In is also called the server plug-in.

Overview

The server plug-in provides:

- A deployment tool that wraps PowerBuilder NVOs as either Enterprise JavaBean (EJB) session beans or J2EE 1.4 Web services for deployment into a J2EE-compliant application server; for session beans, J2EE 1.3 and 1.4 are supported; for Web services, J2EE 1.4 or later is required

Note NVO is a generic term used to describe "custom class user objects," which inherit directly from the PowerBuilder system type NonVisualObject.

- A server runtime library that integrates a PowerBuilder Virtual Machine (PBVM) with an EJB container and an application server's transaction and connection managers
- A remote debugging component that permits debugging from the PowerBuilder IDE
- A proxy server, which allows PowerBuilder clients to call deployed PowerBuilder NVOs, without the need for a client-side JVM
- Sample deployment and configuration templates for getting started quickly and efficiently

Using the server plug-in, you can develop PowerBuilder components on the Windows platform. You can run the components on any platform where your application server supports a PBVM.

The deployment tool runs on your application server and emulates the PowerBuilder NVO deployment API.

JDK versions

The server plug-in installs version 1.4.2 and version 1.5.0 of the Java Development Kit (JDK) from Sun Microsystems. You can also configure the installation to use an existing version, as long as it is the required patch level.

Table 1-1: JDK versions and required patch levels

JDK version	Patch level
1.4.2	13
1.5.0	09

To verify the version and patch level of an existing installation, change to the *bin* directory of the JDK installation and run:

```
java -version
```

Installing the server plug-in

Install the server plug-in on the application server host, which need not be the same machine where the PowerBuilder IDE is running.:

1 Exit any programs that are running. If you do not, the Sybase installer may not be able to copy some files to the appropriate directories.

2 If you have downloaded PowerBuilder Application Server Plug-In, expand the installation software to a temporary location. Otherwise, insert the software CD into your CD drive.

3 Select Start | Run, and enter:

```
path\setup.exe
```

where *path* is the location of the installation software

The installer starts, and the installation window displays.

4 Click Next in the installation window.

Note Use Back and Next to step backward and forward through the installation process to modify specifications as necessary.

Select Cancel to halt the installation process.

---

5 Select your country from the dropdown list to display the license agreement. You must read and accept the terms of the license agreement for your country before you can install any Sybase products. Click Next.

6 Enter the full path to which the PowerBuilder Application Server Plug-In is to be installed.

If you want to modify the default locations of either the Sybase or Shared directory locations, select Show Advanced Locations and modify.

7 Select the type of installation, then click Next:

- Typical – installs the most common installation options.
- Full – installs everything.
- Custom – allows you to choose specific installation options.

  Select the features to install by placing a check mark next to the feature.

8 To use a JDK that is already installed, select Use the Following JDK, and enter the path to the JDK installation or use the Browse button to locate it. The JDK version must be one of those listed in Table 1-1 on page 2.

   If the installer detects an existing JDK of the appropriate version, it is displayed as the default location.

   If you do not select to use an existing JDK, the JDK is installed from the CD. If you choose a Typical install, JDK version 1.4.2 is installed. If you choose a Full install, both JDK versions 1.4.2 and 1.5 are installed. If you choose a Custom install, you can select which JDK versions to install.

9 The installer displays a summary of the features to be installed and the installation directory. Review these entries, then click Next to continue or Back to modify your entries.

   The installer begins copying files.

10 If the PowerBuilder Application Server Plug-In will obtain the licences it requires from a license server, select yes, and enter the host name and port number of the server.

11 To configure e-mail alerts, select yes, and enter:

   - SMTP Server Host Name
   - SMTP Server Port Number
   - Sender E-mail
   - Recipient E-mails
   - Message Severity for E-mail Alerts

12 The installer prompts you for an administrative password. Enter and confirm a password, following the guidelines described by the installer.

13 A summary screen informs you when the installation is complete.

   Select either of these options if you want to perform the corresponding tasks at this time. You can also perform these tasks at a later time:

   - View Readme – displays the readme file.

- Launch the Sybase Product Download Center (SPDC) Web Site – log in to the SPDC Web site to obtain a license for the PowerBuilder Application Server Plug-In. You need to know the product edition and license type for your particular installation. If you do not have this information, ask your system administrator. See the *FLEXnet Licensing End User Guide*, and the *Sybase Software Asset Management User's Guide* for more information.

14 Click Finish to exit the installer.

You can install multiple copies of the server plug-in on one machine, as long as each copy is in a separate installation directory.

Note PB_SERVER_HOME represents the server plug-in installation directory.

Configuring the server plug-in

To configure your system to run the server plug-in:

1 Open the *%PB_SERVER_HOME%\bin\set-java-home.bat* file, and verify that the Java environment variable—either DJC_JAVA_HOME_14 or DJC_JAVA_HOME_15—refers to the home directory of the JDK you plan to use.

   Verify that all the DJC_* variables are set to the correct JDK version; for example, for WebSphere:

   ```
   set DJC_JAVA_HOME_15=%WAS_HOME%\java
   set DJC_RT_DEFAULT=15
   set DJC_JDK_DEFAULT=15
   set DJC_JAVAC_TARGET=1.5
   ```

2 Set the administrative password for the server plug-in by running:

   ```
   "%PB_SERVER_HOME%"\bin\set-admin-password.bat
   ```

   The system prompts you to enter a password, which must contain at least six characters, and one of these must be a digit.

3 Configure the server plug-in for your application server, as described in the following sections.

When you configure the server plug-in, properties are added to *%PB_SERVER_HOME%\Repository*. If you configure the server plug-in for one application server, then want to configure the server plug-in for another application server, first remove the *%PB_SERVER_HOME%\Repository* directory. Otherwise, the repository will contain properties for both application servers, which causes problems.

Note If a directory name contains a space, you must enclose the "-D" options in double quotes for both configuring the server plug-in and starting the application server; for example, to run JBoss:

```
bin\run.bat "-Dpb.server.home=%PB_SERVER_HOME%"
```

JBoss

To configure the server plug-in installed in a JBoss application server:

1 Shut down the JBoss application server.

2 Change to the *%PB_SERVER_HOME%\config* directory.

3 Verify the settings in the configuration scripts:

- *pb-server-jboss.xml* – general server plug-in, listener, security, and data source properties.
- *ejb-proxy-jboss.xml* – proxy server properties.

If required, edit then save the files.

4 Change to the *%PB_SERVER_HOME%\bin* directory.

5 Run the following command, where *jboss-home-dir* represents the JBoss installation directory:

```
configure pb-server-jboss -Djboss.home=jboss-home-dir
```

WebLogic

Note Sybase recommends that you use a Sun JVM with WebLogic; stability issues have been reported with the BEA jRockit JVM.

To configure the server plug-in installed in a WebLogic application server:

1 Shut down the application server if it is running.

2 Change to the *%PB_SERVER_HOME%\config* directory.

3 Verify the settings in the configuration scripts:

- *pb-server-weblogic.xml* – general server plug-in, listener, security, and data source properties.
- *ejb-proxy-weblogic.xml* – proxy server properties.

If required, edit, then save the files.

4 Change to the *%PB_SERVER_HOME%\bin* directory.

5 Run the following commands, where *wls-home-dir* represents the WebLogic installation directory:

```
configure pb-server-weblogic –Dwls.home=wls-home-dir
```

WebSphere

To configure the server plug-in installed in a WebSphere application server:

1 Shut down the WebSphere application server.

2 Change to the *%PB_SERVER_HOME%\config* directory.

3 Verify the settings in the configuration scripts:

- *pb-server-websphere.xml* – general server plug-in, listener, security, and data source properties.
- *ejb-proxy-websphere.xml* – proxy server properties.

If required, edit then save the files.

4 Change to the *%PB_SERVER_HOME%\bin* directory.

5 Run the following commands, where *was-home-dir* represents the WebSphere application server installation directory:

```
configure pb-server-websphere –Dwas.home=was-home-dir
```

Note If you edit any of the configuration scripts after you have configured the server plug-in, re-run the configure command.

Running the server plug-in

After you install and configure the server plug-in, start your application server:

1 WebLogic only: verify that *%PB_SERVER_HOME%\lib\pb-server-15.jar* is in either the CLASSPATH or a location that is shared by all the deployed applications in the JVM.

2 Start your application server.

The configuration task creates the script *start-<appServer>.bat*, which you use to start the application server with the plug-in. Consult your application server administrator to determine the best start-up options, and add them to the script if necessary.

The configuration task also creates *run-<appServer>.bat*, which defines the environment variables that are required to run the server plug-in. *start-<appServer>.bat* calls *run-<appServer>.bat*.

❖ **Starting a JBoss application server**

- Run:

```
%PB_SERVER_HOME%\bin\start-jboss.bat
```

❖ **Starting a WebLogic application server**

- Run:

```
%PB_SERVER_HOME%\bin\start-weblogic.bat
```

The first time you start the server:

a Use the WebLogic console to define a start-up class called com.sybase.pb.server.PbServerStart.

b Shut down, then restart the server.

❖ **Starting a WebSphere application server**

- Run:

```
%PB_SERVER_HOME%\bin\start-websphere.bat
```

The first time you start the server:

a Use the WebSphere administration console to enable the Start-up Beans service.

b Deploy the PowerBuilder start-up service: *%PB_SERVER_HOME%\deploy\websphere\pb-startup.jar*.

c Shut down, then restart the server.

CHAPTER 2

Creating Embedded Installations

If you are packaging the PowerBuilder Application Server Plug-In with your own software, you may want to use a script to create a silent installation, so your end users can install the PowerBuilder Application Server Plug-In without interacting with the server plug-in installer.

Configuring and running the silent installer

The silent installer is a Java program. The installation CD contains a sample batch file to run the install with the correct JRE and CLASSPATH settings. An additional text file specifies the installation type and options. These instructions assume that you will include the PowerBuilder Application Server Plug-In install files and customized installer scripts with the install media for your own software.

❖ **Configuring the silent installer**

1. Create a directory for your install image.
2. Copy the following files from the PowerBuilder Application Server Plug-In installation CD to the install image directory:
   - *PBASP100.jar*.
   - *readme.htm*. This file contains a link to the online documentation.
   - The *JRE_1_5* subdirectory and its contents. Sybase recommends that you use this JRE to run the install. Other JRE versions may not work as well.
   - The *Modules\LicensePanel* directory is required, as are the files in the *Modules* folder.

3 Create configuration and run files as described in "Configuration and run files" on page 10. Copy any required additional files from the PowerBuilder Application Server Plug-In CD to your install image, as described in that section.

4 To support the silent uninstallation process, create the files described in "Silent uninstaller files" on page 15.

❖ **Testing and running the silent installer**

1 Before running a silent installation:

   a Exit any programs that are running. If you do not, the installer may not be able to copy some files to the appropriate directories.

   b Verify that there is enough space in your product directories; 450MB are required.

   c If your home directory contains an InstallShield *vpd.properties* file, make a backup copy. If you run the installer with a different user ID, check for this file in the home directory of that user ID and back it up if it exists.

2 Test the silent installer using the run script that you created at the command line or in your own product's installation script. Running a silent installation takes 5 – 10 minutes, depending on the speed of your computer.

3 After each trial run, check for errors, and clean up your machine as described in "Troubleshooting and cleanup of the silent installer" on page 16.

Configuration and run files

In a silent installation, users cannot input information or choices. You must supply all required information in a configuration file or on the command line that runs the silent installer. The PowerBuilder Application Server Plug-In installation script contains a sample configuration file, *SilentInstall.txt*. The script *SilentInstall.bat* runs the install with this configuration. Start with copies of these files and modify them to suit your installation.

Place your configuration file and run script in the root directory of your install image. Edit the run script to refer to the file name you are using for your configuration file.

Edit the configuration file to customize the install as described below.

Installation location

To specify the installation location, set `-W setPBInstallLocWindow.value`.

The default value is `C:\Program Files\Sybase\PBAppServer1`.

License agreement

For the silent installation to run, you must change the value of `-V AgreeToSybaseLicense` from false to true, indicating that you have read and agreed to the software license agreement. You can view license text by running the interactive install or on the Sybase Web site at http://www.sybase.com/softwarelicenses.

JDK installation parameters

You can configure the PowerBuilder Application Server Plug-In to use the JDKs listed in Table 1-1 on page 2. For each JDK version, you can either install the JDK or use an existing installation.

The parameters in Table 2-1 allow you to configure the PowerBuilder Application Server Plug-In installation to use JDK installations that are already in place.

Table 2-1: Silent installer existing JDK parameters

Parameter	Specifies
-V EASJDKUseExisting_JDK14_CheckBox	Whether to use an existing JDK 1.4 installation. To use an existing installation, set this parameter to true and specify the location as the value of the next parameter. Also, set the value of JDK14.active to false in the feature selection section.
-V EAS_JDK14_Install_Location	If you are using an existing JDK 1.4 installation, the location where it is installed. Verify the version and patch level of the specified JDK as described in "JDK versions" on page 2.
-V EASJDKUseExisting_JDK15_CheckBox	Whether to use an existing JDK 1.5 installation. To use an existing installation, set this parameter to true and specify the location as the value of the next parameter. Also, set the value of JDK15.active to false in the feature selection section.
-V EAS_JDK15_Install_Location	If you are using an existing JDK 1.5 installation, the location where it is installed. Verify the version and patch level of the specified JDK as described in "JDK versions" on page 2.

To install a JDK from your install image, enable the feature parameter for that JDK and include the required files in your image, as listed in Table 2-2.

Table 2-2: Parameters to install JDKs

Parameter	Specifies	Comments
-P JDKs.active	Whether to install any JDKs from the install image	If not set to true, the next two parameters are ignored
-P JDK14.active	Whether to install JDK 1.4	
-P JDK15.active	Whether to install JDK 1.5	

Administrative password

To enable starting the application server, set the administrative password:

```
"set JVM_ARG=%JVM_ARG% -Deas.password=adminPassword"
```

Sybase Software Asset Management License input parameters

If licenses are to be obtained from the License Server, you must define the license server parameters. Set the parameters in Table 2-3 to define the license server.

Table 2-3: License server parameters

Parameter	Set the value to
-V Variable_LicServerYes	Set to true to use a license server
-V Variable_LicServerHostname	The license server host
-V Varaiable_LicServerPortNum	The license server port

Sybase Software Asset Management e-mail alerts

To configure e-mail alerts, set:

-V Variable_RBEmailAlertsYes=true
-V Variable_RBEmailAlertsNo=false

Uncomment and set the -V Variable_CBSySAMEmailSeverity variable to one of these values:

WARNING
INFORMATIONAL
ERROR

Additional SySAM variables that you can set include:

- -V Variable_TFSySAMEmailHost
- -V Variable_TFSySAMEmailPort
- -V Variable_TFSySAMEmailSender

- -V Variable_TFSySAMEmailRecipient
- -V Variable_CBSySAMEmailSeverity

See the *Sybase Software Asset Management User's Guide* for additional information.

Feature selection parameters

These parameters specify which optional features to install. Table 2-4 lists the parameters that select which features are installed. Each parameter requires a value. Specify true to install the feature or false to not install the feature.

Some features have a parent-child relationship (shown by indentation in the sample installation script). To install child features, you must enable both the parent feature and the child feature.

Some features require additional files to be added to your installation image, as listed in Table 2-4. If you enable these features, add the required files to your image by copying them from the PowerBuilder Application Server Plug-In installation CD. Paths within your install image must match those listed in Table 2-4.

Table 2-4: Feature selection parameters

Parameter	Feature	Additional requirements
-P Server.active	Parent feature for several core server and client install features.	
-P CorePluginFiles.active	Files required to run the server plug-in.	Requires parent feature -P Server.active. Requires file: *\Modules\PBASP-100_ThirdPartyLegal.pdf*
-P RuntimeLibraries.active	Parent feature for client runtime libraries. No runtime libraries are installed unless this parameter is set to true.	Requires parent feature -P Server.active.
-P Standard.active	Standard is compatible with JDK 1.4 and JDK 1.5.	Requires parent feature -P RuntimeLibraries.active.
-P Optimized.active	Optimized is compatible with JDK 1.5 only.	Requires parent feature -P RuntimeLibraries.active.
-P Extras.active	Parent feature for extra features.	Requires parent feature -P Server.active.
-P jConnect605.active	Installs the jConnect DB Scripts.	Requires parent feature -P Extras.active.

Parameter	Feature	Additional requirements
-P JDKs.active	Parent feature for JDK installation.	
-P JDK14.active	Installs JDK 1.4.	Requires parent feature -P JDKs.active. Requires file: *\Modules\eas-jdk-14.jar*
-P JDK15.active	Installs JDK 1.5.	Requires parent feature -P JDKs.active. Requires file: *\Modules\eas-jdk-15.jar*
-P SybaseSYSAM.active	Parent feature for Sybase Software Asset Management (SySAM)	
-P NetworkLicenseServer.active	Installs the SySAM network license server.	Requires parent feature -P SybaseSYSAM.active Requires file: *\Modules\sysam.jar*
-P LicenseUtils.active	Installs the SySAM license utilities.	Requires parent feature -P SybaseSYSAM.active Requires file: *\Modules\sysam-utils.jar*
-P ToolsSupport.active	Parent option for the tools support option. If this parameter is set to false, the tools support options are ignored.	
-P PowerBuilderv1051.active	Installs the PowerBuilder version 10.5.1 virtual machine.	Requires parent feature -P ToolsSupport.active. Requires file: *\Modules\pbvm1051.jar*

Specifying parameters on the command line

You may want to configure some install settings dynamically at install time. For example, you may want to set the PowerBuilder Application Server Plug-In installation directory to a location selected by the end user of your own installer. To do this, you can remove settings from the configuration file and specify them as command line arguments to the silent installer.

For example, if your silent installation script is *SilentInstall.bat*, this command installs the PowerBuilder Application Server Plug-In to *C:\Program Files\Sybase\PBserverPlugin*:

```
SilentInstall.bat -W
"setInstallLocWindow.value=C:\Program Files\Sybase\PBserverPlugin"
```

You must also remove the equivalent settings from the silent install configuration file.

Silent uninstaller files

You can configure the silent installer to support silent uninstallation. This creates a script that your users can run to silently remove the installation from their system. The silent uninstaller requires:

- The *PBASPuninstall.jar* and *uninstall.dat* files that are created when users run the installer. The JAR file contains the Java uninstallation program, and the *.dat* file contains data about installed features. These files are installed in the *_uninstall* subdirectory of your PowerBuilder Application Server Plug-In installation.
- A JRE installation of the same version as found on the PowerBuilder Application Server Plug-In install CD.
- The files *SilentUninstall.txt* and *SilentUninstall.bat*. If these are present in the root directory of your silent installer, they are copied to the *_uninstall* directory when users run the install. You must prepare these files as described below before you release your silent installer to your users.

SilentUninstall.txt

A sample of this file is on the PowerBuilder Application Server Plug-In installation CD. However, *SilentUninstall.txt* does not run unless you edit the copy placed in your install image. This file configures the features to uninstall, using syntax similar to the options described in “Feature selection parameters” on page 13. To remove everything, set all the feature options to true. For a partial uninstallation, change the feature options to false for those features that should not be removed.

Note Some feature options in *SilentUninstall.txt* have a parent-child relationship, indicated by indentation in the sample file. To uninstall a parent feature, the parent feature and all child features must be set to true in *SilentUninstall.txt*.

SilentUninstall.bat

Users will run this file to remove the installation from their systems. A sample of this file is on the PowerBuilder Application Server Plug-In installation CD. Place a copy in your install image, and verify the following, keeping in mind that the file will be run in the PowerBuilder Application Server Plug-In *_uninstall* directory:

- The CLASSPATH includes *PBASPuninstall.jar* (located in the same directory).
- The java command line specifies the path to a java executable of the same version as supplied on the PowerBuilder Application Server Plug-In software CD. You can run the uninstall with the JRE that is installed in the *_jvm* subdirectory of the installation, for example:

```
..\_jvm\bin\java -classpath %CLASSPATH% run %* -options
SilentUninstall.txt
```

The uninstaller does not remove all files. Files created after the installer was run are not deleted. This includes log files, property and resource files updated at runtime, and any application files that you have created in the PowerBuilder Application Server Plug-In directory. After uninstalling, you must remove these files manually.

Troubleshooting and cleanup of the silent installer

After a trial run of your silent install, check for errors, verify the installation, and clean up the machine before trying another run.

Check for installer errors on the console and in the installer log file in the specified install location. If you see *ZipException* errors, make sure you have included all required files in the install image.

When testing your installation results, start a server in the installation and verify that the expected features are licensed by checking the licensed features listed in the server log file. If not, verify that you have configured the license parameters described in “Sybase Software Asset Management License input parameters” on page 12.

Test any other features that you are installing, such as the Web Console, Web Services, and so forth.

❖ **Cleaning up the machine**

Before re-running a silent installation, uninstall the previous installation using the silent uninstaller—see "Silent uninstaller files" on page 15.

If your installer was not configured correctly, or you abort the installation before it completes, the uninstall process may fail. In that case, clean the previous installation from your machine as follows:

1 Delete the PowerBuilder Application Server Plug-In installation directory and subdirectories.

2 If you made a backup copy of the *vpd.properties* file, restore it. Otherwise, delete the *vpd.properties* file that was generated during the installation.

3 Make sure the DJC_HOME environment variable is not set in the shell where you re-run the install.

CHAPTER 3

Developing PowerBuilder Components

This chapter describes how to build and deploy PowerBuilder components using the PowerBuilder Integrated Development Environment (IDE) version 10.5.1.

Developing PowerBuilder components

PowerBuilder provides built-in system objects you can use in client/server, multitier, and Web applications. You can also build both visual and nonvisual user-defined objects. One type of nonvisual user-defined object, the custom class user object, inherits directly from the PowerBuilder NonVisualObject system class. Objects of this type, often called simply NVOs, can be deployed to J2EE-compliant application servers as EJB components and EJB 2.1 Web services. The server plug-in must be installed and configured on the application server.

Application server component wizards

Develop and deploy PowerBuilder NVOs as application server components using application server component wizards and projects in the PowerBuilder development environment.

There are three application server component wizards, which are available from the Target, PB Object, and Project pages in the New dialog box.

Target

From the Target page, the Application Server Component wizard creates a new PowerBuilder application server target, a PowerBuilder library (PBL), and a PowerBuilder custom class user object (NVO). It also creates a project from which you deploy the component to the server.

In the wizard, specify the properties of the application server component, including the profile of the server to which you want to deploy, a package name, transaction properties, and whether the components should be deployed as a Web service. For details, see "Specifying component properties" on page 20.

When you complete the wizard, open the NVO in the User Object painter, add methods to your component's interface, and write scripts for events. The object has two pairs of events: Activate and Deactivate, and Constructor and Destructor. You typically use the Activate and Deactivate events to control instance pooling. For more information, see "Controlling the state of a pooled instance" on page 24.

PB Object

To add additional objects to the same target, and optionally, to the same PBL, package, and project, right-click the application server target in the System Tree, select New, and complete the Application Server Component wizard from the PB Object page of the New dialog.

Project

From the Project page, the Application Server Component wizard creates a new project to which you can add any NVOs in the current target's library list. You might want to use this wizard to deploy a set of components to more than one application server.

Specifying component properties

Most of the properties listed in this section can be specified in any of the wizards or in the Project painter. Table 3-1 lists the properties in the order they appear in the Application Server Component wizard that you launch from the Target page.

Note You must define an application server profile before you use a wizard. For more information, see "Creating an application server profile" on page 22.

Table 3-1: Component properties

Property	Description
Application name, library, and target	By default, the name you use for the application is used for the library and target and as part of the object and component names.
Library search path	The path the target searches for NVOs. Click Browse to add additional libraries to the path.
PowerBuilder object name and description	The name of the NVO in the PowerBuilder library.
Application server component name	The name of the EJB component generated from the NVO. *Do not use hyphens* in the name of the component or any of its methods.
Application server profile	A predefined profile that specifies the host, port, user name, and password for an application server.
Package name	The name used to generate the default Java package name and JNDI name for the component.
Java package name	The name of the Java package. For an NVO package called "xyz," the default Java package name is xyz.ejb. Use this property if you want to specify a different name, such as com.mycompany.bank. You must specify a name to deploy the component as an EJB 2.1 Web service.
Package comment	An optional comment that is associated with the package.
Role name list	One or more security role names, entered one on each line. In the Target and PB Object wizards, the role names apply to the component generated by the wizard. In the Project wizard, the role names apply to all the components selected in the wizard. You can associate role names with individual components on the Components page of the Properties dialog in the Project painter. If required, use your application server to map these logical role names to physical roles.
Instance pooling options	Whether an instance of a component is always pooled after each client use, or controlled by the CanBePooled event. See "Controlling the state of a pooled instance" on page 24.
Component timeout	How long a component can remain idle before being deactivated. The default, 0, specifies that the component is never automatically deactivated.
Transaction support options	Whether the component supports transactions. See "Transaction support options" on page 22.
Stateless session bean	Whether the component is in stateless mode.
Expose user events as methods	Whether to include user-defined events in the component interface.
Expose public instance variables	Whether to generate get and set methods for public instance variables.
Use of unsupported datatypes generates an error	Whether to generate an error at build time if the component uses unsupported datatypes such as system datatypes and the Any datatype. If you choose not to generate errors, any functions or variables that use unsupported datatypes are not available in the component interface.

Property	Description
Perform full rebuild	Whether to perform a full rebuild before deploying to the server to ensure that all objects are synchronized.
Collapse class hierarchy	Whether the methods of ancestor objects are included in the component interface.
Debugging options	Whether the component can be debugged remotely and rebuilt from the User Object painter. These options are for use in test environments only. See "Testing and debugging components" on page 36.
Expose component as EJB 2.1 Web service	Whether to expose the component on the server as an EJB 2.1 Web service. The application server must support J2EE 1.4, and you must specify a Java package name. EJB 2.1 Web services do not support the ResultSet return type. You must use array or structure types instead. Method names cannot be overloaded.
Project name and description	The name and optional description of the project used to generate and deploy the component.
Dynamic library options	Whether to consolidate all the libraries in the library list into a single PowerBuilder dynamic library (PBD) file. To ensure that DataWindow™ objects are included in the PBD, select Include Unreferenced Objects.

❖ **Creating an application server profile**

An application server profile is a named set of parameters stored in your system registry that defines a connection to a particular application server host. Before you use a wizard to create a component, create a profile for the server where the component will be deployed.

1 Select Tools | Application Server Profile.

   The Application Server Profiles dialog displays, listing your configured application server profiles.

2 Select Add.

   The Edit Application Server Profile dialog displays.

3 Enter the profile name, the TCP host name for the server, the IIOP port number on the server, the login name admin@system, and the password you specified for the server plug-in.

4 (Optional) Select Test to verify the connection.

5 Click OK to save your changes and close the dialog box.

   The Application Server Profiles dialog displays, with the new profile name listed. The application server profile values are saved in the Registry in *HKEY_CURRENT_USER/Software/Sybase/PowerBuilder/10.5/JaguarServerProfiles*.

Transaction support options

Each component has a transaction attribute that indicates how the component participates in transactions. Table 3-2 lists the options.

Table 3-2: Transaction attribute options

Transaction type	Description
Not Supported	The component never executes as part of a transaction. If the component is activated by another component that is executing within a transaction, the new instance's work is performed outside the existing transaction.
Supports Transaction	The component can execute in the context of a transaction, but a transaction is not required to execute the component's methods. If the component is instantiated directly by a client, the server does not begin a transaction. If component A is instantiated by component B and component B is executing within a transaction, component A executes in the same transaction.
Requires Transaction	The component always executes in a transaction. When the component is instantiated directly by a client, a new transaction begins. If component A is activated by component B and B is executing within a transaction, A executes within the same transaction; if B is not executing in a transaction, A executes in a new transaction.
Requires New Transaction	Whenever the component is instantiated, a new transaction begins. If component A is activated by component B, and B is executing within a transaction, then A begins a new transaction that is unaffected by the outcome of B's transaction; if B is not executing in a transaction, A executes in a new transaction.
Mandatory	Methods can be invoked only by a client that has an outstanding transaction. Calling this component when there is no outstanding transaction generates a runtime error.
Never	Methods cannot be invoked when there is an outstanding transaction. Calling this component when there is an outstanding transaction generates a runtime error.

The PowerBuilder TransactionServer class supports the following methods:

- CreateInstance – (for NVO intercomponent calls) use the two-argument form, and specify the full JNDI name of the target component:

```
TransactionServer ts
getContextService("TransactionServer", ts)

// generate and use proxies
pbtest_MyComp comp
ts.createInstance(comp, "pbtest/MyComp")
```

```
// call methods on comp
```

If the target NVO is not in the same EJB-JAR as the calling NVO, your application server's class loader may not work correctly; you may get a ClassCastException. Consult your application server vendor for help with class-loader issues across EJB-JAR boundaries.

- DisableCommit – prevents the current transaction from being committed, because the component's work has not been completed. The instance remains active after the current method returns.
- EnableCommit – do not deactivate the component after the current method invocation; allow the current transaction to be committed if the component instance is deactivated.
- IsTransactionAborted – determines whether the current transaction has been aborted.
- SetAbort – specifies that the component cannot complete its work for the current transaction and that the transaction should be rolled back. The component instance is deactivated when the method returns.
- SetComplete – indicates that the component has completed its work in the current transaction and that, as far as it is concerned, the transaction can be committed and the component instance can be deactivated.

Note If you are using neither a proxy server nor the TransactionServer.CreateInstance method for NVO intercomponent calls, remove "-djcProxy" from the ejbSourceOptions property value in *pb-server-<targetServerName>.xml*, and re-run configure. This prevents generating unused code.

Controlling the state of a pooled instance

When you create an application server component that supports instance pooling, that component might need to reset its state after each client has finished using the pooled instance.

To allow you to control the state of a component, the application server triggers one or more of the events shown in Table 3-3 during the life cycle of the component.

Table 3-3: Component-state events

Event	PBM code
Activate	PBM_COMPONENT_ACTIVATE
CanBePooled	PBM_COMPONENT_CANBEPOOLED
Deactivate	PBM_COMPONENT_DEACTIVATE

When the component's pooling option is set to Supported, you might need to script the Activate and Deactivate events to reset the state of the pooled component. This is necessary if the component maintains state in an instance, shared, or global variable.

When the component's pooling option is set to Not Supported, you can optionally script the CanBePooled event to specify whether a particular component instance should be pooled. If you script the CanBePooled event, you may also need to script the Activate and Deactivate events to reset the state of the pooled component. If you do not script the CanBePooled event, the component instance is not pooled.

The Application Server Component wizards that you launch from the Target and PB Object pages automatically add the Activate and Deactivate events to the NVOs they generate. If you want to script the CanBePooled event, add this event yourself. If you do this, map the event to the correct PBM code.

Constructor and Destructor are fired once When instance pooling is in effect, a component's Constructor and Destructor events are each fired only once. The Constructor and Destructor events are not fired each time a new client uses the component instance. Therefore, to reset the state of a component instance that is pooled, add logic to the Activate and Deactivate events, instead of to the Constructor and Destructor events.

PowerBuilder to EJB datatype mapping

Table 3-4 lists the PowerBuilder to EJB datatype mappings, which are valid for datatypes passed by value, in and return parameter modes.

The PowerBuilder Application Server Plug-In does not support IDL inout and out parameter modes, because JAX-RPC holder classes are not portable in EJB remote interfaces.

Table 3-4: PowerBuilder to EJB datatype mappings

PowerBuilder type	EJB parameter type
Blob	byte[]
Boolean	boolean
Byte See “Byte datatype” on page 27.	byte
Char	char – see “Character datatypes” on page 28.
Date	java.util.Calendar
DateTime	java.util.Calendar
Decimal	java.math.BigDecimal
Double	double
Integer	short For Java client components that communicate with PowerBuilder server components, the numerical range that this datatype supports is -32768 – 32767.
Long	int For Java client components that communicate with PowerBuilder server components, the numerical range that this datatype supports is -2147483648 – 2147483647.
LongLong	long
Real	float
String	String
Time	java.util.Calendar
MyModule_MyArray[] or MyArray[] (return type only)	MyModule.ejb.MyElement[]
MyModule_MyException or MyException	MyModule.ejb.MyException
MyModule_MyComp or MyComp	MyModule.ejb.MyComp
MyModule_MyStruct or MyStruct	MyModule.ejb.MyStruct
MyModule_MyUnion or MyUnion	MyModule.ejb.MyUnion
MyModule_MyElement[] or MyElement[]	MyModule.ejb.MyElement[]
MyModule_MySequence or MySequence (return type only)	MyModule.ejb.MyElement[]
MyModule_MyElement[n] or MyElement[n]	MyModule.ejb.MyElement[]

PowerBuilder type	EJB parameter type
ResultSet	java.sql.ResultSet
ResultSets	java.sql.ResultSet[]
XDT_BooleanValue	java.lang.Boolean See "XDT datatypes" on page 29.
XDT_CharValue	java.lang.Character See "Character datatypes" on page 28.
XDT_ByteValue	java.lang.Byte
XDT_ShortValue	java.lang.Short
XDT_IntValue	java.lang.Int
XDT_LongValue	java.lang.Long
XDT_FloatValue	java.lang.Float
XDT_DoubleValue	java.lang.Double
XDT_DecimalValue	java.math.BigDecimal
XDT_IntegerValue	java.math.BigInteger
XDT_DateValue	java.util.Calendar
XDT_TimeValue	java.util.Calendar
XDT_DateTimeValue	java.util.Calendar

Sybase suggests that you use the PowerBuilder DataStore system object with the ResultSet return type, especially for NVOs running in an application server. For improved performance, use NVO instance variables, and create the DataStore and assign the DataObject in your NVO constructor.

Byte datatype

PowerBuilder version 10.5 introduced a Byte datatype. To use the PowerBuilder Char datatype for backward compatibility, run the following command (once) before deployment:

```
configure idl-octet-to-pb-char
```

To switch back to using the PowerBuilder Byte datatype, run the following command (once) before deployment:

```
configure idl-octet-to-pb-byte
```

Camel case option

You can modify the default mapping of CORBA IDL identifiers to EJB identifiers to use Java naming conventions. This is called the "camel case" deployment option. When using this option, IDL operation and parameter names, such abc_xyz, map to abcXyz. IDL interfaces, sequence, structure, and union type names, such as abc_xyz, map to AbcXyz. These mappings are not applied to exception and structure field names. By default, the camel case option is disabled.

To enable the camel case option:

```
configure camel-case-on
```

To disable the camel case option:

```
configure camel-case-off
```

If you plan to expose components as Web services, enable the camel case option; otherwise, you may have problems with the JAX-RPC identifier mapping rules. See Chapter 20, "Appendix: Mapping of XML Names" in the JAX-RPC 1.1 specification, which you can download from Java Technology and XML Downloads at http://java.sun.com/xml/downloads/jaxrpc.html#jaxrpcdocs11.

Character datatypes

Only characters in the ISO 8859-1 character set can be used for in and return parameter modes. To propagate other characters, use the String datatype.

The char and java.lang.Character datatypes have no defined XML schema mappings for EJB Web services, so you cannot use these as a parameter types or structure field types if you intend to expose a component as a Web service. Use the String datatype instead.

DataStore system object

Sybase recommends that you use the PowerBuilder DataStore system object with the ResultSet return type, especially for NVOs running in an application server. For improved performance, use NVO instance variables, and create the DataStore and assign the DataObject in your NVO constructor.

ResultSet datatype

If you intend to expose a component as a Web service, do not use the ResultSet datatype, because java.sql.ResultSet is not portable in EJB Web service endpoint interfaces. You can use arrays (IDL sequences) of structures instead, Java arrays or PowerBuilder variable-sized arrays. The EJB return type java.sql.ResultSet maps to a complex XML schema element that contains result set data and the schema for the result set. The content of the nested XML element is mapped according to the SQL/XML ANSI standard; for example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
   targetNamespace="jdbc.wst.sybase.com">
  <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="DataReturn">
    <sequence>
       <element name="XML" nillable="true" type="xsd:string" />
       <element name="updateCount" type="xsd:int" />
       <element name="DTD" nillable="true" type="xsd:string" />
       <element name="schema" nillable="true" type="xsd:string" />
    </sequence>
   </complexType>
</schema>
```

Using IDL parameter modes inout and out with TabularResults::ResultSet is not supported for components that are exposed as Web services. You may find that using arrays (IDL sequences) of structures instead of ResultSets simplifies the coding of Web service client applications, and this technique is portable across all application servers. When writing PowerBuilder NVO methods, which do not permit the use of arrays as method return types, define a row structure to represent a result row, and a table structure containing an array of row structures to represent a ResultSet.

XDT datatypes

To obtain the PowerBuilder XDT_* datatypes to use as PowerBuilder structure field types or component parameter types, use the EAServer Proxy wizard or the Application Server Proxy wizard in the PowerBuilder IDE to generate proxies for the XDT package. Each of the XDT_* datatypes contains a value field and an isNull field. You must set isNull to true if you want to indicate null values.

Data access

From PowerBuilder NVOs, you can access data using either JDBC data sources or Sybase native data sources.

❖ **Accessing JDBC data sources in NVOs**

1 To set up a JDBC data source in an NVO, use this PowerScript™ code:

```
sqlca.dbms = "JDBC"
sqlca.dbparm = "CacheName='DefaultDS'"

connect;    // check error code
...         // use embedded SQL or DataStore

disconnect; // check error code
```

2 Using your application server facilities, define a JDBC data source and assign a JNDI name to it.

3 Edit *%PB_SERVER_HOME%\config\pb-server-<serverName>.xml*, and map the value of the PowerBuilder CacheName to an application server data source JNDI name.

4 Re-run the configure command—see “Configuring the server plug-in” on page 5.

Native data sources

The server plug-in supports five native data source types: Sybase, Oracle, Oracle Unicode, ODBC, and ODBC Unicode. These data source types create their connections using C/C++ code. The connections are managed using Java objects, which provide a JDBC API.

❖ **Accessing Sybase native data sources in NVOs**

1 The following PowerScript code sets the DBMS to Sybase native, and the cache name to the Sybase_JCM cache:

```
sqlca.dbms = "SYJ"
sqlca.dbparm = "CacheName='Sybase_JCM'"
...
```

To use a cache other than "Sybase_JCM," set CacheName to the value of a data source that is defined in *%PB_SERVER_HOME%\config\pb-server-<serverName>.xml*.

2 In your application server interface, set the driver class and database URL data source properties, replacing *dbName*, *userName*, and *password* with the appropriate values.

Data source type	Driver class	Database connection URL
Sybase_JCM	com.sybase.jaguar.jcm.sybase.SybaseDriver	jdbc:sybase:jcm:sybase:databaseName=*db-name*;user=*userName*; password=*password*
Oracle_JCM	com.sybase.jaguar.jcm.oracle.OracleDriver	jdbc:sybase:jcm:oracle:databaseName=*db-name*;user=*userName*; password=*password*
Oracle_Unicode	com.sybase.jaguar.jcm.oracle.OracleuDriver	jdbc:sybase:jcm:oracle:databaseName=*db-name*;user=*userName*; password=*password*
Odbc_JCM	com.sybase.jaguar.jcm.odbc.OdbcDriver	jdbc:sybase:jcm:odbc:databaseName=*db-name*;user=*userName*; password=*password*
Odbc_Unicode	com.sybase.jaguar.jcm.odbc.OdbcuDriver	jdbc:sybase:jcm:odbc:databaseName=*db-name*;user=*userName*; password=*password*

3 Sybase_JCM data source type. The server plug-in uses different names for Open Client™ libraries. Copy the libraries from the Open Client *dll* directory to *%PB_SERVER_HOME%\lib*, changing the library names as appropriate—the Open Client library names begin with "lib" and the PowerBuilder library names begin with "libj."

Open Client libraries	PowerBuilder libraries
libcs.dll	*libjcs.dll*
libct.dll	*libjct.dll*

Before deploying components

This section describes tasks to perform before you deploy PowerBuilder components to your application server.

JBoss

Before you deploy PowerBuilder components to a JBoss application server:

1 Verify that automatic deployment is enabled—see the JBoss online documentation at http://wiki.jboss.org/wiki/Wiki.jsp?page=ConfiguringTheDeploymentScannerInConfjbossSystem.xml.

2 Set the ScanEnabled attribute to true.

Sybase tests have shown that automatic redeployment does not always work correctly—you may need to restart JBoss to pick up your changes.

WebLogic

Before you deploy PowerBuilder components to a WebLogic application server:

1 Customize the WebLogic Ant deployment options in *%PB_SERVER_HOME%\config\wls-ejb-deploy.xml*. See your WebLogic documentation for details.

2 In *%PB_SERVER_HOME%\bin\wls-ejb-deploy.bat*, verify the settings for:

- username
- password
- port

- servername

WebSphere

Before you deploy PowerBuilder components to a WebSphere application server, customize the WebSphere Ant deployment options in the following files, located in *%PB_SERVER_HOME%\config*:

- *ws-ejb-deploy.xml*
- *ws-install-app.xml*

See your WebSphere documentation for details.

Sybase tests have found that deploying an EJB-JAR to WebSphere can be slower than deploying to JBoss or WebLogic. Check your application-server or server plug-in log file to verify the status of deployment in progress.

Deploying components to an application server

The deployment tool provided with the server plug-in wraps PowerBuilder NVOs as standard EJB session beans and generates target-specific deployment descriptors to bind JNDI names and JDBC data source resource references automatically.

❖ **Deploying PowerBuilder components**

1 Open the PowerBuilder Application Server Component project in the Project painter.

2 Select Edit | Properties from the menu, or click Properties on the PainterBar.

3 On the Server Host page in the Properties dialog, verify that the properties you specified in the Application Server Profiles dialog are correct. See "Creating an application server profile" on page 22.

Note You can override the host name and port number that the server uses for its deployment listener by changing the iiopListeners property in *%PB_SERVER_HOME%\config\pb-server-<targetServerName>.xml*. If you change this property, re-run the configure command.

4 In the Standard Options group on the Components page, verify that you have selected Stateless Session Bean if you want the component to be stateless.

5 If your scripts reference DataWindow objects dynamically, on the Libraries page, select the Include Unreferenced Objects in Consolidated PBD to make the DataWindow definitions available to the component.

6 On the Advanced page, optionally specify the names and values of custom EJB properties.

7 Click OK to apply your changes.

8 Select Design | Deploy Project from the menu or click the Deploy button on the PainterBar.

❖ **Validating deployment**

To validate that your NVOs deployed successfully, you can run the test program pb-server-test. pb-server-test must communicate directly with the application server, not with a proxy server.

1 Verify that the application server with the server plug-in is running.

2 Use the PowerBuilder IDE to define a component called "MyComp" (case sensitive) in a package called "pbtest."

3 Add a few business methods to the component; perhaps methods that access the database.

4 To generate basic performance metrics, create a method called "perftest"; for example, as in the following pseudocode:

```
integer perftest(integer a, integer b)
{
   return a+b;
}
```

5 In *pb-server-<targetServerName>.xml* (for example, *pb-server-weblogic.xml*), verify that these properties are set correctly:

```
<property name="test.username" value="weblogic"/>
<property name="test.password" value="weblogic"/>
```

Note If you change the XML configuration file, re-run configure—see "Configuring the server plug-in" on page 5.

6 For WebSphere only, use the IBM JVM, instead of the Sun JVM:

a Set JAVA_HOME to *%WAS_HOME%\java*.

b Add *%JAVA_HOME%\bin* to the path.

7 Deploy the package to your application server.

8 Change to *%PB_SERVER_HOME%\bin*, and run:

```
pb-server-test
```

The test program tries to call all the methods in your component, using fake parameter values; for example, 1, 2, "S1," "S2." If you created a "perftest" method, the test program calls it repeatedly. If the program runs successfully, you see something similar to the following:

```
Looking up home interface using JNDI name: pbtest/MyComp
Obtaining EJB meta data...
Resolving home interface class...
Home interface type is: ejb.components.pbtest.MyCompHome
Narrowing home interface...
Looking up home create() method...
Calling home create method...
Remote interface type is: ejb.components.pbtest.MyComp
Business method signatures:
perftest(integer, integer)
mymethod(java.lang.String, java.lang.String)
Calling business methods:
perftest(1, 2) -> 3
... checking application server remote call performance
#remote calls/sec = 1189
#remote calls/sec = 1641
#remote calls/sec = 1650
#remote calls/sec = 1739
#remote calls/sec = 1749
#remote calls/sec = 1748
#remote calls/sec = 1748
#remote calls/sec = 1749
#remote calls/sec = 1768
#remote calls/sec = 1768
mymethod(S1, S2) -> S1S2
Test passed.
```

Note If you see an error such as `LoadLibrary Failed: pbjag100.dll`, verify that the PowerBuilder directory that contains *pbvmXXX.dll* is in the system PATH.

Source code for the test program is located in *%DJC_HOME%\src\java\com\sybase\pb\server\PbServerTest.java*.

Generated code

The base directory for generated files is *%PB_SERVER_HOME%\genfiles\java*, which includes these subdirectories:

- *applications*
- *classes*
- *ejbjars*
- *src*
- *stubs*

Typically, you can delete generated files after deployment, but this causes redeployment to be slower. If you are using a proxy server, do not delete the generated files; the proxy server uses some of them at runtime.

Naming conventions

If you enable the camel case option (see "Camel case option" on page 27), PowerScript identifiers that contain underscores are mapped to Java names using lowerCamelCase for NVO methods; for example, "my_simple_method" maps to "mySimpleMethod."

A similar mapping is used for structure names, but the first letter is capitalized; for example, "my_structure" maps to "MyStructure."

Component names are not changed from the names you enter in the Project painter. Sybase recommends that you use the Java class naming conventions; for example, "MyComp."

An NVO implementation class can use any name.

Repository files

The base directory for repository files is *%PB_SERVER_HOME%\Repository*, which includes these subdirectories:

- *IDL* – interface definitions.
- *Component* – component properties and PowerBuilder dynamic libraries (PBDs).
- *Instance* – server and data source properties.

- *Package* – package properties.

The repository files are used during deployment and at runtime.

Clusters

If your application server is running in a cluster, and each server in the cluster has its own copy of the server plug-in directory, you must either:

- Deploy all the components to all the servers, or
- Copy the contents of the *Repository* directory to all the servers in the cluster, and use your application server to distribute the deployed EJB-JAR files across the cluster.

Testing and debugging components

This section describes three techniques you can use to test your components:

- Live editing
- Remote debugging
- Writing messages to the server log

Live editing

To test or debug a component, you can use a feature of PowerBuilder called live editing, which allows you to build the project automatically from the User Object painter. When live editing is enabled, PowerBuilder builds the project for an application server component each time you save the corresponding user object. The generator does not deploy PBDs to the application server, but instead tells the application server how to access the PBLs that contain the required object definitions.

Configuring a server for live editing

To configure an application server for live editing:

1 Open *%PB_SERVER_HOME%\config\pb-server-<targetServerName>.xml*, and set the ejbDeployIfUnchanged property to false.

2 Re-run configure.

Enabling live editing for an NVO

To enable live editing for an NVO:

1 Create a project that includes the user object for which you want to generate an application server component.

You can use an existing PBL that allows for deployment to an application server, or alternately, you can create a new project and use it only for testing.

2 Optionally modify the live editing library list for the project.

When you are testing a component with a server that resides on a remote machine, you must tell the server where to find the PBLs. To do this, modify the library list on the Advanced page of the Properties dialog in the Project painter.

The library list you specify must contain fully qualified paths that use Universal Naming Convention (UNC) names. UNC names take the form: *\\servername\sharename\path\file*.

By default, the live editing library list is based on the application library list. You need not modify the live editing library list if your server is local.

3 In the User Object painter, on the General page in the Properties view, specify the project that will generate the component.

The project name you specify must meet these requirements:

- It must be an application server component project.
- It must include the user object that you currently have open in the User Object painter.
- The library list for the project must match the current application library list.

Generating the component in the painter

To generate an application server component from the User Object painter, select File | Save. PowerBuilder builds the component just as it would at deployment time, except that it does *not* generate PBDs for the component.

If the project build results in errors, PowerBuilder displays the error messages in the Output window.

If instance pooling is enabled for the user object, the generator disables pooling for the current build. Pooling is not supported with live editing because PowerBuilder cannot save the user object if the PBL that contains the user object is locked by the application server.

Remote debugging

When you are building a PowerBuilder NVO as an application server component, you can use the PowerBuilder debugger to debug the application server component. You can debug the component whether you use the live editing feature in the User Object painter or deploy the component to the server from the Project painter.

Getting ready to debug a component

Before you begin debugging a remote component, check that your configuration meets these requirements:

- You are using the same version of the application and PBLs as were used to develop the deployed component. To debug several deployed components in the same session, they must all have been built using the same versions of the PBLs, the same application name, and the same library list.
- The Supports Remote Debugging check box on the Components page in the Project painter is selected. You can also set the debugging option by checking the Supports Remote Debugging check box in the Project wizard.
- You have a client application that exercises the methods and properties in the deployed components. This can be a compiled executable built with any compatible development tool, or a PowerBuilder application running in another PowerBuilder session.

Starting the debugger

Open the target that contains the deployed components. Click Start Remote Debugging in the PainterBar, and complete the wizard. You can select only components that were generated in PowerBuilder with remote debugging support turned on. Remote debugging support is a security setting that does not add any debugging information to the component. To prevent users from stepping into and examining your code, turn remote debugging support on when you are testing a component, then, turn it off when you deploy the component to a user's site.

Set breakpoints as you would when debugging a local application, then start the client application that invokes the remote components (if it is not already running).

About states

The Instances view shows the state of each component instance:

- **Idle** The component is idle or in the instance pool.
- **Running** The component is currently executing code.
- **Stopped** The component is stopped at a breakpoint waiting for a debugger action.

When a component instance is destroyed, it is removed from the Instances view.

Multiple instances

Multiple component instances can be stopped at the same time, but actions you take in the debugger act only on the first instance that hits a breakpoint. This instance is indicated by a yellow arrow in the Instances view. The current instance changes to the next instance in the queue when the method completes or when you click Continue.

You can also change context from one instance to another by double-clicking the new instance in the Instances view. You might want to do this if you step over a call to another component instance and the Instances view shows that the called instance stopped.

Writing messages to the server log

To record errors generated by PowerBuilder objects running in an application server to *%PB_SERVER_HOME%\logs\pb-server.log*, create an instance of the ErrorLogging service context object and invoke its log method. For example:

```
ErrorLogging errlog
getContextService("ErrorLogging", errlog)
errlog.log("Write this string to log")
```

You can use the ErrorLogging service to provide detailed information about the context of a system or runtime error on the server. This information is useful to system administrators and developers in resolving problems.

While you are developing components, you can use the ErrorLogging service to trace the execution of your component. For example, you can write a message to the log when you enter and exit functions. The message can identify the name of the component, the name of the function, and whether it is entering or exiting the function.

Troubleshooting

To troubleshoot runtime problems, check:

- The PowerBuilder application server plug-in log *%PB_SERVER_HOME%\logs\pb-server.log*
- The application server log files

- The application server console window, if available

CHAPTER 4

Developing PowerBuilder Clients

This chapter describes how to build PowerBuilder clients and how to use a proxy server to allow PowerBuilder clients to connect to NVOs running in an EJB server.

Developing a PowerBuilder client

A PowerBuilder application can act as a client to an application server component. To access a method associated with a component on the server, the PowerBuilder client must connect to the server, instantiate the component, and invoke the component method.

❖ **Building and deploying an application server client**

1. Use the Connection Object wizard to create a standard class user object that inherits from the Connection object. You can use this object in a script to establish a connection.

   If you use the Template Application wizard to create the client application, you can create the Connection object in that wizard.

2. Use the Application Server Proxy wizard from the Project page of the New dialog to create a project for building proxy objects that you can use to communicate with a proxy server running on the application server. Then generate the proxy objects.

3. Create the windows, menus, and scripts required to implement the user interface.

4. Write the code required to create the application server component instance and call one or more component methods from the client.

5. Test and debug the client.

6 Deploy the application.

Creating a Connection object

When you select Application Server as the connection type in the Connection Object wizard, PowerBuilder creates a standard class user object that inherits from the Connection object. You supply the connection-object properties in the wizard and specify whether connection information will be in the registry, an *INI* file, or a script. The Connection Object wizard gets information about the server to which you want to connect from the application server profiles. See "Creating an application server profile" on page 22.

The Constructor event of the new Connection object calls the function of_getconnectioninfo, which gets the stored connection information from the source you specified.

Once you have completed the Connection Object wizard, execute these PowerScript statements:

1 Use the Create statement to instantiate the connection object.

2 Invoke the ConnectToServer function to establish a connection to the server.

3 Check for errors.

You need not set properties for the connection object, but you can modify them in the of_getconnectioninfo function. You can also set options for the connection object in its Constructor event.

Example The following script instantiates the myconnect instance of the n_myclient_connect object that is created by the wizard, invokes the ConnectToServer function to establish a connection to the server, and checks for errors:

```
long ll_rc
myconnect = create n_myclient_connect
ll_rc = myconnect.ConnectToServer()
IF ll_rc <> 0 THEN
    MessageBox("Connection failed", ll_rc)
END IF
```

Establishing multiple connections

You can establish multiple connections in a single client application. To connect a client to two different servers, run the Connection Object wizard again to create a new user object with different connection properties.

Generating application server proxy objects

To access an application server component, communicate through an application server proxy object in the client application.

Use the Application Server Proxy wizard on the Project page of the New dialog to create projects for building application server proxy objects. It allows you to connect to an application server and select the components you want to be able to access from the client. Once you have created the project, use the Project painter to modify your project settings and build the proxy library.

When you generate a proxy for an application server component that was not created in PowerBuilder, the names of any methods that use a PowerBuilder reserved word are changed. The proxy generator automatically adds an underscore ("_") prefix to these methods. For example, if the component has a method named destroy, the method name in the proxy will be _destroy.

Many application server components throw exceptions that you can handle in your client application. To use the proxy you are generating with a client application that does not handle exceptions, or to not declare the exceptions in the client you are building, exclude exceptions from the generated proxy, either in the wizard or in the Project painter.

Accessing components

For clients—JavaServer Pages (JSPs), servlets, or other EJBs—running in the same application server process, you can use either EJB references or direct JNDI lookups to access components.

When you deploy PowerBuilder components, if the package name is "MyPackage" and the component name is "MyComp":

- The generated EJB home interface is MyPackage.ejb.MyCompHome.
- The generated EJB remote interface is MyPackage.ejb.MyComp.
- The JNDI name is "MyPackage/MyComp."

The PowerBuilder EJBConnection class allows you to call EJBs running in an application server—see the EJBConnection class description in the PowerBuilder documentation at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.pb_10.5.apptech/html/apptech/CCJBGAEBA.htm.

You can find configuration information; for example, the initial context factory, provider URL, and required class path settings, in *%PB_SERVER_HOME%\config\pb-server-<targetServerName>.xml*.

Proxy servers

The PowerBuilder application server proxy enables PowerBuilder clients to communicate with EJB session beans, using a Connection object. The EJB session beans must be running in one of the application servers defined in "Installing and Configuring the PowerBuilder Server Plug-In" on page 1.

The PowerBuilder Application Server Plug-In includes two editions of the proxy server:

- Client – runs as a separate process on the same machine as your PowerBuilder client application.
- Server – runs as a separate process on the same machine as your application server.

Installing proxy servers

The Server Edition of the proxy server is installed when you install the PowerBuilder Application Server Plug-In.

❖ **Installing a Client Edition proxy server**

- To install the Client Edition of the proxy server, run:

  ```
  %PB_SERVER_HOME%\bin\configure ejb-proxy-client
  ```

  A standalone client proxy server is installed in *%PB_SERVER_HOME%\..\EJBProxy*.

Configuring proxy servers

To configure a proxy server, the target application server must be installed on the same machine as the proxy server. For both the Client and Server Editions, verify the Java environment variables, then run the configure command for your application server. The Client Edition may require additional configuration:

1 Open *set-java-home.bat*, and verify that the Java environment variable—either DJC_JAVA_HOME_14 or DJC_JAVA_HOME_15—refers to the home directory of the JDK you plan to use. The location of *set-java-home.bat* depends on the proxy server edition:

- Client Edition – *%PB_SERVER_HOME%\..\EBJProxy\bin*
- Server Edition – *%PB_SERVER_HOME%\bin*

Use a JDK version that your application-server vendor recommends for creating standalone EJB client applications.

To use the proxy server with IBM WebSphere, use an IBM JDK.

2 Run the configure command, specifying the application server installation directory. You can also specify the options defined in Table 4-1.

Table 4-1: Configuration options

Property name	Default value	Syntax example
proxy.name	ejb-proxy	`-Dproxy.name=my-proxy`
proxy.host	*host name*	`-Dproxy.host=my-host` **Note** Can be either a host name or an IP address. Ignored in the Client Edition, which always listens on the local loopback address 127.0.0.1.
proxy.port	2000	`-Dproxy.port=1000`
jboss.host	*host name*	`-Djboss.host=my-host`
jboss.port	1099	`-Djboss.port=2099`
wls.host	*host name*	`-Dwls.host=my-host`
wls.port	7001	`-Dwls.port=7501`
was.host	*host name*	`-Dwas.host=my-host`
was.port	2809	`-Dwas.port=3809`

For each supported application server, the configure command calls an Ant configuration script, which you can customize. The Ant scripts are located in *%PB_SERVER_HOME%\config*:

- *ejb-proxy-jboss.xml*
- *ejb-proxy-weblogic.xml*
- *ejb-proxy-websphere.xml*

3 Client Edition only. If developers will be connecting to the client proxy server using the PowerBuilder IDE:

   a Set the administrative password. Change to the *bin* subdirectory of the client proxy server installation, and run:

```
set-admin-password.bat
```

   The system prompts you to enter a password, which must contain at least six characters, and one of those characters must be a digit.

   b Ping the client proxy port (by default, 2000).

   If only client applications will be connecting to the client proxy server, these steps are not required.

❖ **Configuring a proxy server on JBoss**

- Change to the *%PB_SERVER_HOME%\bin* directory, and run:

```
configure.bat ejb-proxy-jboss "-Djboss.home=jboss-home-dir"
["-D<property-name>=<property-value>"]
```

where:

- *jboss-home-dir* is the JBoss installation directory
- *property-name* is one of the optional properties listed in Table 4-1
- *property-value* is the property value

For example, to set the name of the proxy server to "myProxy":

```
configure.bat ejb-proxy-jboss "-Djboss.home=jboss-home-dir"
"-Dproxy-name=myProxy" "-Dproxy.port=1000"
```

Note If the application server installation directory path contains spaces, enclose the `-D` options in double quotes; otherwise, quotes are not required.

❖ **Configuring a proxy server on WebLogic**

- Change to the *%PB_SERVER_HOME%\bin* directory, and run:

```
configure.bat ejb-proxy-weblogic "-Dwls.home=wls-home-dir"
["-D<property-name>=<property-value>"]
```

where *wls-home-dir* is the WebLogic installation directory, and *property-name* and *property-value* are optional property name/value pairs.

❖ **Configuring a proxy server on WebSphere**

- Change to the *%PB_SERVER_HOME%\bin* directory, and run:

```
configure.bat ejb-proxy-websphere "-Dwas.home=was-home-dir"
["-D<property-name>=<property-value>"]
```

where *was-home-dir* is the WebSphere installation directory, and *property-name* and *property-value* are optional property name/value pairs.

Client Edition proxy servers

After you configure a Client Edition proxy server:

1 Copy the client proxy server installation directory *%PB_SERVER_HOME%\..\EJBProxy* (including its subdirectories) to a client machine. The client machine does not need an application server installation.

You can also copy the client proxy server installation to a server machine that is not running the PowerBuilder Application Server Plug-In.

2 On each machine where the client proxy server is installed:

a Install JDK version 1.4 or 1.5.

Note WebSphere requires an IBM JDK.

b Edit *EJBProxy\bin\set-java-home.bat*, and verify that all the DJC_* variables are set to the correct JDK version.

3 Optionally, you can delete the application server installation from the machine on which you configured the Client Edition proxy server.

Starting and stopping proxy servers

The Server Edition of the proxy server starts and stops automatically when you start or stop the application server. To prevent the proxy server from starting automatically, specify the following option when you configure the PowerBuilder Application Server Plug-In:

```
"-Dproxy.name=none"
```

See “Configuring the server plug-in” on page 5.

The *.bat* files that start and stop a proxy server are located in the *bin* subdirectory of the proxy server installation.

❖ **Starting a proxy server in the current window**

- For a proxy server that uses the default name, run:

```
run-server.bat ejb-proxy
```

Or, if you configured the proxy server using a nondefault value for the proxy.name property, run:

```
run-server.bat proxy-name
```

where *proxy-name* is the name of the proxy server.

The run-server command prints the location of the server log file. Check the server log for debugging information. The default log is *ejb-proxy.log*, located in the *logs* subdirectory of the proxy server installation.

❖ **Starting a proxy server in a new window**

- For a proxy server that uses the default name, run:

```
start-server.bat ejb-proxy
```

Or, if you configured the proxy server using a nondefault value for the proxy.name property, run:

```
start-server.bat proxy-name
```

❖ **Starting a proxy server in the background**

- For a proxy server that uses the default name, run:

```
start-server.bat -bg ejb-proxy
```

Or, if you configured the proxy server using a nondefault value for the proxy.name property, run:

```
start-server.bat -bg proxy-name
```

❖ **Starting a proxy server from a PowerBuilder client**

- In a PowerBuilder client, you can start a proxy server using the PowerScript Run function. For example, add the following code to an application's Open event:

```
Run("C:\ejb-proxy\bin\start-server -bg ejb-proxy",
Minimized!)
```

Sybase recommends that you start the proxy server in the background to impede stopping the proxy server while the client application is running.

❖ **Stopping a proxy server that is running in a command window**

- In the window where the proxy server is running, enter Ctrl+C.

❖ **Stopping a proxy server that is running in the background**

1 Open a command window.

2 If you did not specify a name when you started the proxy server, enter:

```
stop-server.bat -local ejb-proxy
```

If you did specify a name when you started the proxy server, enter:

```
stop-server.bat -local proxy-name
```

❖ **Stopping a proxy server from a PowerBuilder client**

- In a PowerBuilder client, you can stop a proxy server using the PowerScript Run function. For example, add the following code to an application's Close event:

```
Run("C:\ejb-proxy\bin\stop-server -local",
Minimized!)
```

Enabling PowerBuilder clients to communicate with EJBs

To enable PowerBuilder clients to communicate with EJBs using a proxy server:

1 Generate a PowerScript proxy class for each EJB session bean that the client application calls. Session beans can use only the datatypes defined in Table 4-2 on page 51.

2 Code your PowerBuilder client application to communicate with the proxy server.

❖ **Generating PowerScript proxy classes for EJB session beans**

For each EJB session bean, generate proxy classes:

1 Deploy the EJB module to the proxy server.

Note If you are using the Server Edition of the proxy server and your client application calls only PowerBuilder NVOs (wrapped as EJBs), you can skip this step. When you deploy an NVO, this step is performed automatically.

To deploy an EJB module, where *name-prefix* is a JNDI name prefix that the application server uses to look up the remote home interfaces in your EJB module. *deploy.bat* is located in the *bin* subdirectory of your proxy server installation:

```
deploy.bat bank.jar -nc:name-prefix
```

For example, if an EJB called Teller has a JNDI lookup name of "com.acme.bank/Teller," the name prefix must be "com.acme.bank". If you do not specify the `-nc:name-prefix` option, an empty name prefix is used.

2 Use the Application Server Proxy wizard to generate PowerScript proxy classes for the EJB module—see "Generating application server proxy objects" on page 43.

Note A proxy class that is generated using the EJB Client Proxy wizard cannot access the EJB. To rectify, delete the proxy project and the PBL, then generate the proxy class using the Application Server Proxy wizard.

❖ **Coding PowerBuilder clients to communicate with proxy servers**

1 In the client application, add the code to establish a connection to the proxy server; for example:

```
Connection conn
conn = create Connection
conn.driver = "jaguar"
conn.userID = "myUserName"
conn.password = myPassword
conn.location = "iiop://my-host:2000";

int status
status = conn.connectToServer()
if status <> 0 then
// report error
end if
```

The proxy server passes the values of the connection userID and password properties to your application server. See your application server's documentation for instructions on configuring user name/password authentication.

Note If you are using the Client Edition of the proxy server, Sybase recommends that you use a retry loop for establishing a connection, in case the proxy server's start-up procedure has not completed when you try to connect.

2 Obtain a reference to call the session bean; for example:

```
bank_Teller teller
```

```
status = con.createInstance(teller, "bank/Teller")
if status <> 0 then
// report error
end if
```

The name prefix used in the createInstance call is the EJB module name (without the *.jar* suffix) followed by a "/"; in this example, "bank/". If the EJB module name contains periods or hyphens, they are replaced with underscores. The name prefix used in the createInstance call need not match the JNDI *name-prefix* that you specified when you deployed the module.

3 Call business methods on the session bean; for example:

```
teller.deposit("MyAccount", 1000.0)
```

4 For stateful session beans only: remove the session bean. For example:

```
teller.remove()
```

Table 4-2: Java datatypes allowed in proxy classes

Java datatype	PowerBuilder datatype
Array	PowerScript Array **Note** PowerBuilder methods cannot return an Array datatype. PowerBuilder maps a Java Array return type to a PowerBuilder Structure that contains an array.
boolean	Boolean
byte	Byte
byte[]	Blob
char	Char **Note** You can use only characters in the ISO 8859-1 character set. If other characters must be propagated via the proxy server, you must use the String datatype.
double	Double
float	Real
int	Long
java.lang.Boolean	XDT_BooleanValue
java.lang.Byte	XDT_ByteValue
java.lang.Character	XDT_CharValue See the note for the char datatype.
java.lang.Double	XDT_DoubleValue

Java datatype	PowerBuilder datatype
java.lang.Float	XDT_FloatValue
java.lang.Integer	XDT_IntValue
java.lang.Long	XDT_LongValue
java.lang.Short	XDT_ShortValue
java.math.BigDecimal	XDT_DecimalValue
java.math.BigInteger	XDT_IntegerValue
java.sql.Date	XDT_DateValue
java.sql.ResultSet	ResultSet
java.sql.RowSet	XDT_DataTable
java.sql.Time	XDT_DateValue
java.sql.Timestamp	XDT_DateValue
java.util.Calendar	XDT_DateTimeValue
java.util.Date	XDT_DateTimeValue
long	LongLong
short	Int
Serializable class	PowerScript Structure **Note** For any serializable Java class, each non-final non-transient field (private, protected, or public) is mapped to a PowerScript Structure field. Class methods are not mapped.

Collection types

Collection types, such as java.util.list, are not supported in this release. Use Java arrays to ensure full interoperability.

Troubleshooting a proxy server

To find runtime problems with a proxy server, check *%PB_SERVER_HOME%\logs\pb-server-proxy.log*.

Index

E

F

G

I

J

L

N

O

P

R

S

T

U

V

W

X