

SYBASE®

Deploying Applications and Components to .NET

PowerBuilder®

11.0

DOCUMENT ID: DC00586-01-1100-01

LAST REVISED: May 2007

Copyright © 1991-2007 by Sybase, Inc. All rights reserved.

This publication pertains to Sybase software and to any subsequent release until otherwise indicated in new editions or technical notes. Information in this document is subject to change without notice. The software described herein is furnished under a license agreement, and it may be used or copied only in accordance with the terms of that agreement.

To order additional documents, U.S. and Canadian customers should call Customer Fulfillment at (800) 685-8225, fax (617) 229-9845.

Customers in other countries with a U.S. license agreement may contact Customer Fulfillment via the above fax number. All other international customers should contact their Sybase subsidiary or local distributor. Upgrades are provided only at regularly scheduled software release dates. No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of Sybase, Inc.

Sybase trademarks can be viewed at the [Sybase trademarks page](http://www.sybase.com/detail?id=1011207) at <http://www.sybase.com/detail?id=1011207>. Sybase and the marks listed are trademarks of Sybase, Inc. ® indicates registration in the United States of America.

Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Unicode and the Unicode Logo are registered trademarks of Unicode, Inc.

All other company and product names used herein may be trademarks or registered trademarks of their respective companies.

Use, duplication, or disclosure by the government is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFARS 52.227-7013 for the DOD and as set forth in FAR 52.227-19(a)-(d) for civilian agencies.

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

Contents

About This Book ix

PART 1 CHOOSING A .NET TARGET

CHAPTER 1	Overview and Configuration of .NET Targets.....	3
	Choosing a .NET application target	3
	How .NET deployment works	5
	Configuring ASP.NET for a .NET Web project.....	6
	Installing IIS.....	6
	Selecting the default ASP.NET version.....	7
	Viewing and modifying global properties in IIS Manager	8
	Directory structure on the server	9
	Setting up a SQL Anywhere database connection.....	10
	Setting up IE Web Controls on the server	12
	Checklist for deployment to production servers	12

PART 2 WEB FORMS TARGETS

CHAPTER 2	Moving PowerBuilder Applications to the Web	17
	About PowerBuilder Web Forms applications.....	17
	Creating a PowerBuilder .NET Web Forms target	18
	Deploying and running a .NET Web Forms project.....	22
	Sharing data across sessions	26

CHAPTER 3	Client-Side Events and Default Event Handlers.....	29
	About client-side programming.....	29
	Default event handlers	31
	Client-side support for the Web DataWindow control.....	33
	Alphabetical list of Web DataWindow client-side events.....	35
	ButtonClicked	36
	ButtonClicking	37

	Clicked	38
	DoubleClick	39
	ItemChanged.....	40
	ItemError	41
	ItemFocusChanged.....	42
	RButtonDown	43
	RowFocusChanged.....	44
	RowFocusChanging.....	45
CHAPTER 4	User Management and Registry Operations in Web Forms	47
	Creating permanent user accounts	47
	Managing permanent user accounts.....	51
	Using the registry functions.....	53
CHAPTER 5	Print, File, Mail Profile, and Theme Managers	55
	Using the Web Forms Print Manager	55
	Print Manager icon display	56
	Where printed output is saved.....	57
	Requirements for saving files in PDF or XSL format.....	57
	Installing GPL Ghostscript.....	59
	Where PDF and XSL-FO output is saved	60
	Using the Web Forms File Manager	60
	Using the Web Forms Mail Profile Manager	67
	Using the Web Forms Theme Manager	70
CHAPTER 6	Properties for .NET Web Forms	73
	About Web Forms properties	73
	Global Web configuration properties.....	74
	Creating custom global properties	79
	AutoPostBack.....	80
	Embedded	80
	HasFileManager	81
	HasMailManager	82
	HasPrintManager	82
	HasThemeManager.....	83
CHAPTER 7	Functions for .NET Web Forms	85
	About system functions for Web Forms applications	85
	DownloadFile	86
	GetConfigSettings.....	87
	GetDownloadFileURL	88
	OpenFileManager	89

OpenMailManager.....	89
OpenPrintManager.....	90
OpenThemeManager.....	90
UploadFiles.....	90

CHAPTER 8 Modified and Unsupported Features in Web Forms Projects... 93

About unsupported features.....	93
Unsupported objects.....	95
Unsupported system functions.....	96
Restrictions on supported controls.....	98
Modified display of visual controls.....	109
Unsupported functions for controls in Web Forms.....	111
Unsupported events for controls in Web Forms.....	115
Unsupported properties for controls in Web Forms.....	117

PART 3 WINDOWS FORMS TARGETS

CHAPTER 9 Deploying PowerBuilder Applications as .NET Windows

Forms.....	125
About PowerBuilder .NET Windows Forms applications.....	125
Creating a .NET Windows Forms target.....	127
Creating a .NET Windows Forms project.....	129
Setting properties for a .NET Windows Forms project.....	130
Deploying the project from PowerBuilder.....	134
Running the project from PowerBuilder.....	135

CHAPTER 10 Intelligent Deployment and Update..... 137

About intelligent deployment and update.....	137
Publishing an application for the first time.....	138
Installing the application on the user's computer.....	142
Updating the application.....	143
Using the bootstrapper.....	146
Rolling back.....	148
Using MobiLink synchronization.....	149

CHAPTER 11 Unsupported Features in Windows Forms Projects..... 151

About unsupported features.....	151
Unsupported nonvisual objects and structures in Windows Forms.....	153
Unsupported system functions in Windows Forms.....	157
Partially supported visual controls for Windows Forms.....	158

Unsupported functions for controls in Windows Forms..... 162
Unsupported events for controls in Windows Forms..... 163
Unsupported properties for controls in Windows Forms 164

PART 4 .NET ASSEMBLY AND WEB SERVICE TARGETS

CHAPTER 12 .NET Assembly and .NET Web Service Targets..... 169
The .NET Assembly target wizard..... 169
Modifying a .NET Assembly project 172
Supported datatypes 175
Deploying and running a .NET Assembly project..... 175
The .NET Web Service target wizard 176
Modifying a .NET Web Service project 178
Configuring ASP.NET for a .NET Web Service project..... 181
Deploying and running a .NET Web Service project..... 182

PART 5 .NET LANGUAGE INTEROPERABILITY

CHAPTER 13 Referencing .NET Classes in PowerScript 187
About conditional compilation 187
Writing code inside a .NET block 190
PowerScript syntax for .NET calls..... 191
Calling assembly methods from PowerScript..... 193
Support for .NET language features 194
 Bitwise operator support..... 195
 User-defined enumerations 196
 Accessing indexes for .NET classes 198
Handling exceptions in the .NET environment 198

CHAPTER 14 Best Practices for .NET Projects..... 201
Coding restrictions 201
Design-level considerations 204
Take advantage of global configuration properties 207
Use client-side events to delay postbacks 209

CHAPTER 15 Debugging and Troubleshooting 213
Debugging a .NET application 213
Debugging a .NET component..... 217
Troubleshooting deployment errors 218

Troubleshooting tips for Web Forms applications	219
Failure to deploy to local machine alias	219
Browser error messages	219
Problem with toolbars and tab controls	220
Failure to connect to database	220
DataWindows do not display	221
Pictures do not display	222
Excessive flickering on Web page.....	222
Posted events are not executed.....	222
External DLLs cannot be loaded	222
Print failure	223
Log files	223
Problems on Windows 2003.....	223
Troubleshooting tips for Windows Forms applications	224
Runtime errors.....	224
Publish errors	225
Installation errors.....	226
Update errors	227
Index	229

About This Book

Audience	This book is for programmers who plan to convert traditional client-server PowerBuilder® applications to PowerBuilder .NET Web Forms or Windows Forms applications, or to develop new .NET applications or .NET components in PowerBuilder.
How to use this book	This book describes how to use PowerBuilder .NET wizards to generate both Web Forms and Windows Forms applications and to generate Web services and .NET assemblies from PowerBuilder custom class user objects. It provides information on design and coding considerations for converting PowerBuilder applications to .NET applications. It also describes the client-side events and event handlers you can use to enhance the performance of your Web Forms applications.
Related documents	For a description of books in the PowerBuilder documentation set, see the preface of the PowerBuilder <i>Getting Started</i> book. The <i>Getting Started</i> book also has tutorials for .NET Web Forms and Windows Forms applications.
Other sources of information	<p>Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:</p> <ul style="list-style-type: none">• 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. <p>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.</p> <p>Refer to the <i>SyBooks Installation Guide</i> on the Getting Started CD, or the <i>README.txt</i> file on the SyBooks CD for instructions on installing and starting SyBooks.</p>

- 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://www.sybase.com/support/manuals/>.

Conventions

The formatting conventions used in this manual are:

Formatting example	Indicates
Retrieve and Update	When used in descriptive text, this font indicates: <ul style="list-style-type: none"> • Command, function, and method names • Keywords such as true, false, and null • Datatypes such as integer and char • Database column names such as emp_id and f_name • User-defined objects such as dw_emp or w_main
<i>variable or file name</i>	When used in descriptive text and syntax descriptions, oblique font indicates: <ul style="list-style-type: none"> • Variables, such as <i>myCounter</i> • Parts of input text requiring substitution, such as <i>pblname.pbd</i> • File and path names
File>Save	Menu names and menu items are displayed in plain text. The greater than symbol (>) shows you how to navigate menu selections. For example, File>Save indicates “select Save from the File menu.”
dw_1.Update()	Monospace font indicates: <ul style="list-style-type: none"> • Information that you enter in a dialog box or on a command line • Sample script fragments • Sample output fragments

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

Choosing a .NET Target

This part describes differences between .NET Windows Forms and Web Forms targets. It also describes configuration requirements for .NET Web Forms and Web Service targets.

Overview and Configuration of .NET Targets

About this chapter

In PowerBuilder 11 you can deploy PowerBuilder applications as ASP.NET Web or Windows Forms applications. You can also deploy custom class objects as Web service components or .NET assemblies.

This chapter provides an overview of the Web Forms and Windows Forms target choices and describes configuration requirements that apply to .NET Web Forms and .NET Web Service projects.

Contents

Topic	Page
Choosing a .NET application target	3
How .NET deployment works	5
Configuring ASP.NET for a .NET Web project	6
Checklist for deployment to production servers	12

Choosing a .NET application target

Web Forms applications have several advantages over traditional client-server and Windows Forms applications. Web Forms applications do not require client-side installation, are easy to upgrade, have no distribution costs, and offer broad-based user access. Any user with a Web browser and an online connection can run Web Forms applications.

Windows Forms applications with the smart client feature combine the reach of the Web with the power of local computing hardware. They provide a rich user experience, with a response time as quick as the response times of equivalent client-server applications. The smart client feature simplifies application deployment and updates, and can take advantage of Sybase's MobiLink technology to provide occasionally connected capability.

Table 1-1 displays some of the relative advantages and disadvantages of Web Forms and Windows Forms applications.

Table 1-1: Relative advantages of Web Forms and Windows Forms applications

Application type	Advantages	Disadvantages
Web Forms	<ul style="list-style-type: none">• No installation• Easy to upgrade• Broader reach	<ul style="list-style-type: none">• Slower response• Must be online
Windows Forms	<ul style="list-style-type: none">• Rich user experience• Quicker response time• Availability of client-side resources, such as 3D animation• Offline capability	<ul style="list-style-type: none">• Requires client-side installation• Difficult to upgrade
Windows Forms with smart client feature	<ul style="list-style-type: none">• Same advantages as Windows Forms• Easy to deploy and upgrade	<ul style="list-style-type: none">• Requires first time client-side installation

Smart client applications

The PowerBuilder 11.0 smart client feature makes Windows Forms applications easy to upgrade while maintaining the advantages of quick response times and the ability to use local resources. For more information, see Chapter 10, “Intelligent Deployment and Update.”

Although PowerBuilder continues to support traditional client-server as well as distributed applications, PowerBuilder 11.0 provides you with the ability to transform these applications into Web Forms and Windows Forms applications with relative ease.

The decision to convert an application to use Web Forms or Windows Forms depends upon the type of application you plan to convert. Simple inquiry, browsing, or reporting applications are suitable candidates for Web Forms deployment. If you need only part of an application to run in a browser, you can move this part and its dependent objects to a new target that you deploy with a Web Forms project.

Applications that require significant data entry, retrieve large amounts of data (for example, more than 3 MB per request), or have a complex user interface are more suitably deployed as Windows Forms.

If you need to deploy data entry intensive applications as Web Forms, you must allow for slower response times. However, you can enhance the performance of Web Forms applications by reducing postbacks to the server. You do this through the use of client-side events, or by refactoring code so that events associated with individual controls are combined and submitted in a single postback.

For more information on the relative advantages of Web Forms and Windows Forms, see the Microsoft Web site at [http://msdn2.microsoft.com/en-us/library/5t6z562c\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/5t6z562c(VS.80).aspx).

How .NET deployment works

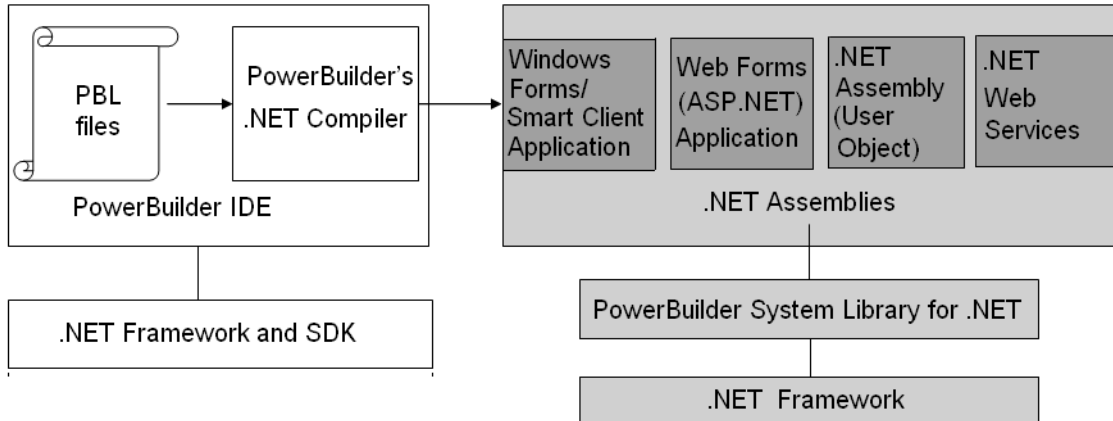
When you deploy a .NET project, PowerBuilder compiles existing or newly developed PowerScript code into .NET assemblies. At runtime, the generated .NET assemblies execute using the .NET Common Language Runtime (CLR). PowerBuilder's .NET compiler technology is as transparent as the P-code compiler in standard PowerBuilder client-server applications.

Depending on their application target type, the assemblies you generate from a .NET project are built into Web Forms or Windows Forms applications. If you generate assemblies from a component target type, the assemblies are deployed as independent .NET components or as Web services.

PowerBuilder Web Forms applications have a three-tier architecture, with the client running in a Web browser on the front end and PowerBuilder components running on the Microsoft IIS server using ASP.NET 2.0 technology. A session is created and is dedicated to processing each user request on the client side, ensuring that the applications are stateful. The session manages the runtime environment, makes required connections to the database, retrieves data, renders HTML responses, and keeps the session active in the server until the user closes the application or the session times out.

PowerBuilder Windows Forms applications run on the .NET 2.0 Framework using local computer hardware resources. The smart client feature permits you to publish Windows Forms applications to an IIS or FTP server, and leverages Microsoft's ClickOnce technology, making it easier for users to get and run the latest version of an application and easier for administrators to deploy it.

Figure 1-1 is a high level architectural diagram showing the conversion of PowerBuilder applications and custom class objects to applications and components on the .NET platform.

Figure 1-1: Conversion of applications and components to .NET

Configuring ASP.NET for a .NET Web project

You can configure ASP.NET for a Web Forms project before or after you deploy the project to an IIS 5.0 or later server. All files and directories that you access from a Web Forms application must have appropriate ASP.NET (IIS 5.0), IIS_WPG (IIS 6.0), or IIS_USRS (IIS 7.0) user permissions.

For an example of granting user permissions to a directory, see “Granting ASP.NET user permissions” on page 10.

Installing IIS

You can install IIS from the Control Panel, but you might need a Windows operating system CD. Select Add and Remove Programs from the Control Panel, then click Add/Remove Windows Components, select the Internet Information Services check box, and click Next. You can then follow instructions to install IIS.

If IIS 5.0 or later is installed after the .NET Framework 2.0, you must register IIS with ASP.NET manually or reinstall the .NET Framework 2.0. To manually register IIS with ASP.NET 2.0, go to the .NET Framework 2.0 path, run `aspnet_regiis.exe -i` in the command line console, and restart IIS.

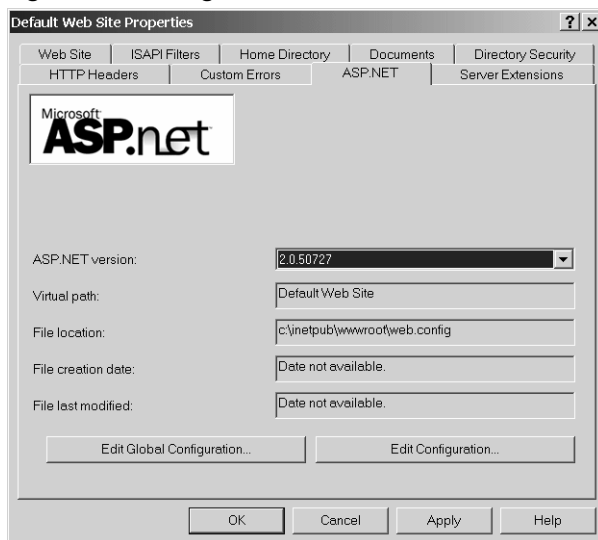
If you use IIS 7.0 for your Web applications and components, make sure you install the IIS 6 Compatibility Component and IIS 6 script tool.

Selecting the default ASP.NET version

If you installed multiple versions of the .NET Framework on the target Web server computer, you should make sure that IIS uses the 2.0 version for PowerBuilder .NET Web Forms applications. You can make this change globally, for all ASP.NET Web site applications, or for individual Web Forms applications that you deploy to IIS.

❖ **To configure the ASP.NET version for all new Web sites:**

- 1 Select Start>Run from the Windows Start menu.
- 2 Type “InetMgr” in the Run dialog box drop-down list.
The IIS Manager displays.
- 3 In the left pane of the IIS Manager, expand the local computer node and its Web Sites sub-node.
- 4 Right-click the Default Web Site node and select Properties from its pop-up menu.
The Default Web Site Properties dialog box displays.
- 5 Click the ASP.NET tab of the Default Web Site Properties dialog box and select 2.0.50727 or later for the ASP.NET version.

Figure 1-2: Setting the default ASP.NET version**Changing the ASP.NET version for an existing Web Forms project**

If you have already deployed a PowerBuilder Web Forms project, you can follow the procedure to configure the ASP.NET version for all new Web sites, but instead of right-clicking on the Default Web Site node in step 4, expand the node and right-click on the .NET Web Forms application that you deployed from PowerBuilder. Then proceed with step 5.

Viewing and modifying global properties in IIS Manager

Although you set global properties for a Web Forms application on the Configuration page of the Project painter before you deploy the project, you can also view and modify the global properties in the IIS Manager after the project is deployed.

For information about global properties generated with a PowerBuilder .NET Web Forms project, see “Global Web configuration properties” on page 74.

❖ **To view and edit global properties in IIS Manager:**

- 1 Expand the nodes in the left pane of the IIS Manager until you see the node for the Web Forms application whose properties you want to examine.

- 2 Right-click on the Web Forms application and select Properties from the pop-up menu.
- 3 Click the ASP.NET tab and change the ASP.NET version to 2.0.50727 if necessary.
- 4 Click Edit Configuration.

The ASP.NET Configuration dialog box displays for the current .NET Web Forms application. You can view its global properties in the list box at the bottom of the General tab.

Modifying a global property for the application

You modify a global property by selecting that property in the Application Settings list box and clicking Edit. You can then type in a new value for that property and click OK. The next time you run the Web Forms application, the new global property value is used.

Directory structure on the server

When you deploy a PowerBuilder .NET Web Forms application, PowerBuilder creates two top-level directories for the application under the IIS root. One of the directories takes the name of the application specified in the Web Forms project, and the other appends “_root” to the application name.

The *applicationName* directory contains the generated *cs* and *aspx* files, as well as subdirectories for any resource files, PowerBuilder libraries, and external modules that you deploy with your application.

The *applicationName_root* directory contains directories named File, Mail, Log, and Print. The File directory contains the Common, Session, User, and Icon subdirectories. The File\Common directory holds read-only files specified in the Web Forms project. The paths to the read-only files mirror the paths on the development computer, with the drive letter serving as the name for the top subdirectory under File\Common directory.

The subdirectories under the File\Common directory include the initial current directory that you assigned in the .NET Web Forms Application wizard or in the Project painter. If an application user performs write operations on a file in a File\Common subdirectory, a *SessionID* folder is created under the File\Session directory (or, if the application user has a permanent user account, a *UserName* folder is created under the File\User directory), and the read-only file is copied there in a mirrored path before a user can save the modified file.

The File\User directory contains files saved by logged-in users whose profiles are included in a permanent user database. For information about creating user profiles, see “Creating permanent user accounts” on page 47.

The File\Icon directory is used by the PowerBuilder .NET Web Forms runtime engine to convert .ICO files to .GIFs and .BMPs. Its contents are not visible to Web Forms application users.

Setting up a SQL Anywhere database connection

Before a PowerBuilder .NET Web Forms application connects to a SQL Anywhere® database, you must either start the database manually or grant the ASPNET user (IIS 5 on Windows XP), the IIS_WPG user group (IIS 6 on Windows 2003), or IIS_USRS (IIS 7 on Windows Vista) full permissions for the Sybase\Shared and Sybase SQL Anywhere directories, making sure to replace permissions of all child objects in those directories.

Starting the database manually

If your database configuration uses a server name, you must provide the database server name in the start-up options when you start the database manually, in addition to the name of the database file you are accessing.

Granting ASP.NET user permissions

If you do not grant the appropriate user permissions for Sybase directories and your database configuration is set to start the database automatically, your application will fail to connect to the database. SQL Anywhere cannot access files unless the ASPNET, IIS_WPG, or IIS_USRS user group has the right to access them.

❖ To grant an ASP.NET user full permissions for Sybase directories:

- 1 In Windows Explorer, right-click the Sybase, Sybase\Shared or Sybase SQL Anywhere directory and select Properties from the pop-up menu.

The Properties dialog box displays for the selected directory.

- 2 Select the Security tab of the Properties dialog box for the directory and click the Add button.

The Select Users, Computers, or Groups dialog box displays.

If the Security tab does not display

To display the Security tab, you might need to modify a setting on the View tab of the Folder Options dialog box for your current directory. You open the Folder Options dialog box by selecting the Tools>Folder Options menu item from Windows Explorer. To display the Security tab, you must clear the check box labeled “Use simple file sharing (Recommended)”.

- 3 Click Locations and choose the server computer name from the Locations dialog box and click OK.
- 4 Type ASPNET (IIS 5), IIS_WPG (IIS 6), or IIS_USRS (IIS 7) in the list box labeled “Enter the object names to select” and click OK.

If valid for your server, the account name you entered is added to the Security tab for the current directory. (You can check the validity of a group or user name by clicking Check Names before you click OK.)

- 5 Select the new account in the top list box on the Security tab, then select the “Full Control” check box under the Allow column in the bottom list box.
- 6 Click the Advanced button.

The Advanced Security Settings dialog box displays for the current directory.

- 7 Select the check box labeled “Replace permission entries on all child objects with entries shown here that apply to child objects” and click OK.

A Security dialog box displays, warns you that it will remove current permissions on child objects and propagate inheritable permissions to those objects, and prompts you to respond.

- 8 Click Yes at the Security dialog box prompt, then click OK to close the Properties dialog box for the current directory.

Tracing runtime exceptions

The *pbtrace.log* file is created in the *applicationName_root* directory. This file records all runtime exceptions thrown by the application and can be used to troubleshoot the application.

Setting up IE Web Controls on the server

PowerBuilder .NET Web Forms use Internet Explorer Web Controls to display correctly and to provide functionality for the Tab, TreeView, and Toolbar controls.

You can download IE Web Controls from the Microsoft Web site at <http://www.asp.net/IEWebControls/Download.aspx>. The download comes with a Readme file that provides instructions for installing the controls.

After you install the IE Web Controls by running the *build.bat* file included in the download, you must copy the controls to a *webctrl_client\1_0* directory under the IIS root.

❖ **To copy the IE Web Controls:**

- 1 Open a DOS command box.
- 2 Change directories to the directory where you installed the IE Web Controls.
- 3 Type the following line at the command prompt, modifying the server IIS root directory if you do not use the default *c:\Inetpub\wwwroot* directory:

```
xcopy /s /i .\build\Runtime  
c:\Inetpub\wwwroot\webctrl_client\1_0 /y
```

- 4 Press Enter.

This creates the following directory structure under the root:

```
\webctrl_client\1_0  
    [images]  
    [treeimages]
```

The *\webctrl_client\1_0* directory should contain the following files: *MultiPage.htc*, *TabStrip.htc*, *toolbar.htc*, *treeview.htc*, *webservice.htc*, and *webserviced.htc*

Checklist for deployment to production servers

For deployment of all .NET target types, production servers must have:

- Windows XP SP2, Windows 2003, or Vista operating system
- .NET Framework 2.0

- PowerBuilder runtime dynamic link libraries in the system path or global assembly cache

You can use the PowerBuilder Runtime Packager tool to deploy runtime DLLs. For a list of PowerBuilder runtime DLLs, see the chapter on “Deploying Applications and Components” in *Application Techniques*. For .NET Web Forms and Web Service targets, see also “Deploying to a production server” on page 23.

In addition, for .NET Web Forms and Web Service targets, production servers must have:

- IIS 5, IIS 6, or IIS 7

For information on configuring IIS, see “Installing IIS” on page 6.

- ASP.NET Framework 2.0

For information on configuring the .NET Framework, see “Selecting the default ASP.NET version” on page 7

- IE Web Controls installed (Web Forms only)

For information on installing IE Web Controls, see “Setting up IE Web Controls on the server” on page 12.

- ASP.NET permissions for all files and directories used by your applications

For an example of how to grant ASP.NET permissions, see “Setting up a SQL Anywhere database connection” on page 10. For command line instructions granting ASP.NET permissions to deployed application directories, see “Granting ASP .NET user permissions” on page 25.

For information on three different methods for deploying .NET Web Forms applications to a production server, see “Deploying to a production server” on page 23. These methods are also valid for deployment of .NET Web Service components.

PART 2

Web Forms Targets

This part describes how to create and deploy Web Forms applications.

Moving PowerBuilder Applications to the Web

About this chapter

In PowerBuilder 11 you can deploy PowerBuilder applications as ASP.NET Web applications.

This chapter explains how to generate PowerBuilder applications as Web Forms applications.

Contents

Topic	Page
About PowerBuilder Web Forms applications	17
Creating a PowerBuilder .NET Web Forms target	18
Deploying and running a .NET Web Forms project	22
Sharing data across sessions	26

About PowerBuilder Web Forms applications

The PowerBuilder .NET Web Forms solution employs ASP.NET technology. It has a three-tier architecture, with the browser client as the front end, and the PowerBuilder components on the IIS server as the middle tier. The database tier remains unchanged.

Moving an existing application from client-server architecture to three-tier Web architecture typically requires a significant effort in modifying the application code and the tolerance of various functionality restrictions due to constraints of the Web environment. The PowerBuilder .NET Web Forms solution is intended to ease the deployment of existing client-server applications to the Web and allow you to use your PowerBuilder skills to create new Web applications.

You must take into account the Internet bandwidth available, the rendering capability of client Web browsers, and IIS server environment factors when determining whether .NET Web Forms are an optimal solution for new or existing applications.

Creating a PowerBuilder .NET Web Forms target

System requirements

You must install version 2.0 of the Microsoft .NET Framework on the same computer as PowerBuilder 11, and you must make sure that the system PATH environment variable includes the location of the .NET Framework. If you installed the 1.x and 2.0 versions of the .NET Framework, you must make sure the PATH variable lists the 2.0 version first.

Some functionality, such as application toolbars, requires the installation of IE Web Controls on the IIS server where you deploy a .NET Web Forms target.

If you are deploying .NET applications from a computer with the Vista operating system, you must run PowerBuilder as the computer administrator.

For more information about installation and configuration, see “Configuring ASP.NET for a .NET Web project” on page 6.

About the .NET Web Forms target

You can use the PowerBuilder .NET Web Forms Application target wizard to create a Web Forms target “from scratch” or from an existing PowerBuilder application.

The existing application object that you select to use as a Web Forms application can be an application object from any type of PowerBuilder target. By default, if the existing application is already included in a target in the current workspace, the wizard reuses the entire library list from the existing target as the library list for the Web Forms target that the wizard creates.

After the wizard creates a Web Forms target from an existing application, all objects from that application are visible in the System Tree for the Web Forms target except project objects for other types of PowerBuilder targets.

About the .NET Web Forms project

Whether you use the target wizard to create a new target from scratch or from an existing application, the target wizard always creates a new project. It automatically launches the .NET Web Forms Application project wizard. A .NET Web Forms project object is required to deploy the Web Forms application to an IIS 5.0 or later server. Once the application is deployed to a server, end users can run it from a Web browser.

Although you can always start the .NET Web Forms Application project wizard from the Project tab of the New dialog box, you can start it for a .NET Web Forms target type only. If the current workspace does not have a target of this type, PowerBuilder does not let you run the .NET Web Forms Application project wizard.

Table 2-1 lists optional and required items in the .NET Web Forms Application project wizard:

Table 2-1: .NET Web Forms Application project wizard fields

Wizard field	Description
Project name	Name of the .NET Web Forms project.
Project library	Library where you want to store the .NET Web Forms project.
Web application name	Name of the .NET Web Forms application. By default, this is the name of the application for the current PowerBuilder target.
Application URL preview	Address for starting the .NET Web Forms application in a browser (minus the <i>default.aspx</i> or <i>default.htm</i> start-up file name).
Resource file and directory list	Specifies a list of resource files, or directories containing resource files, that you want to deploy with the project. When you select a directory, the resource files in all of its subdirectories are also selected by default. However, after you complete the wizard, you can clear the check box in the Recursive column on the Resource Files tab page for the project. If you do that, the resource files in the selected directory, but not in any of its subdirectories, are selected for deployment.
Win32 dynamic library file list	Specifies any Win32 DLLs that you want to include with your project. Modules in this list are deployed to the <i>bin</i> directory in the application Web site under the virtual root folder.
JavaScript file list	Specifies JavaScript files you want to deploy with the project.
Generate setup file option and Setup file name	Select this option and a setup file name if you are not deploying directly to an IIS server.
Direct deploy to IIS and IIS server address	Select this option to deploy to an IIS server and enter the address of the server where you want to deploy the .NET Web Forms application.

Using the .NET Web Forms Project painter

After you click Finish in the project wizard, PowerBuilder creates a .NET Web Forms project and opens the project in the Project painter. The Project painter displays the values you entered in the wizard and allows you to modify them. The painter also includes functionality that is not available in the .NET Web Forms Application project wizard.

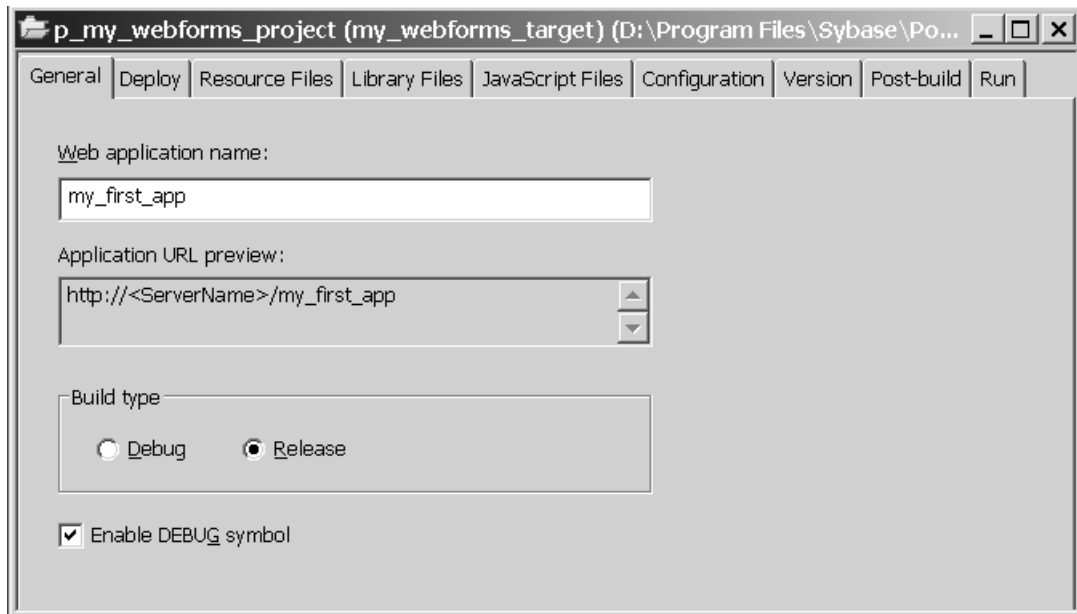
Table 2-2: Additional functionality in the Project painter

Project tab page	Functionality not available in the .NET Web Forms wizard
General	<p>Includes radio button options that determine whether the project is deployed as a debug build (default selection) or a release build. You use debug builds for debugging purposes. Release builds have better performance, but when you run a release build in the debug mode, the debugger does not stop at breakpoints.</p> <p>Also includes the Enable DEBUG Symbol check box that you can select to activate code inside conditional compilation blocks using the DEBUG symbol. This selection does not affect and is not affected by the project's debug build or release build setting. This check box is selected by default.</p>
Resource Files	<p>The wizard automatically includes the resource files from all subdirectories of a directory that you add to the wizard's Resource Files page. In the Project painter, a check box displays under the Recursive column for each directory in the Resource Files page list box. You can clear the check box to deploy only the files in the directory that is listed.</p>
Library Files	<p>The Library Files tab has separate list boxes for target libraries (PBLs and PBDs) and for dynamic Win32 library files (DLLs) that you want to deploy with your project. The PBLs you select are generated as PBDs.</p> <p>By default, all target libraries are selected, but you need to select a PBL only if it contains DataWindow or Query objects that you use in your application. If your target library list includes a PBD file that contains other types of PowerBuilder objects, such as functions or user objects, you cannot reference those objects in your Web Forms application.</p> <p>These types of objects must be contained in a PBL file rather than in a PBD file before you deploy them to a Web Forms target. For a Web Services client, you can import a PBX file into a target PBL using the Import PB Extension item on the library's pop-up menu, rather than using the PBD file that contains the SoapConnection and SoapError classes.</p>
Configuration	<p>On this Project painter page, you can modify global properties for the project before it is deployed. You or the application server manager can also change global properties after the project is deployed.</p> <p>For more information about global properties, see "Global Web configuration properties" on page 74.</p>

Project tab page	Functionality not available in the .NET Web Forms wizard
Version	You can specify version information for the project on this Project painter page. The version information includes values for the product name, company name, description, and copyright, as well as major, minor, build, and revision version numbers for the product, file, and assembly that you generate when you build the project. The values you enter display in the generated assembly file's Properties dialog box in Windows Explorer. They are viewable on the Web Forms server, but are not typically available to end users of Web Forms applications.
Post-build	You can use this Project painter page to select an application, such as a code obfuscator program, to process the generated Web Forms application immediately after it is deployed. You can select different applications for post-build processing of debug and run versions of your project.
Run	Contains the Application field where you can enter the path to a browser you want to have run the Web Forms application and the Arguments field where you can enter the URL for the Web Forms application. By default, the path to the Internet Explorer browser is displayed for the Application field. The Arguments field is populated by default with the value for the project's Application URL Preview, with Localhost as the default server name.

Figure 2-1 displays the General page of the Project painter for a .NET Web Forms project.

Figure 2-1: .NET Web Forms Application Project painter



Deploying and running a .NET Web Forms project

Deploying and running the project from the PowerBuilder UI

When a .NET Web Forms project is open in the Project painter and no other painters are open, you can select Design>Deploy Project from the Project painter to deploy the project. When all painters are closed, including the Project painter, you can right-click a .NET Web Forms project in the System Tree and select Deploy from its pop-up menu.

The Output window displays the progress of the deployment and provides a list of application functions, events, and properties that are not supported in the Web Forms version of the application. Most of these warnings are benign and do not prevent users from running the application as Web Forms.

If the 2.0 version is the only version of the Microsoft .NET Framework installed on the server, or if you configured the server to use the 2.0 version for all Web sites by default, you can run the application immediately after you deploy it. You can run the application from PowerBuilder by selecting Design>Run Project from the Project painter menu or selecting the Run Project toolbar icon from the Project painter toolbar. The System Tree pop-up menu for the .NET Web Forms project also has a Run Project menu item.

Deploying to a setup file

If you are deploying a .NET project to an MSI file, you must have a file named *License.rtf* in the PowerBuilder *DotNET\bin* directory. The PowerBuilder setup program installs a dummy *License.rtf* file in this directory, but you should modify this file's contents or replace the file with another file of the same name. The *License.rtf* file should contain any license information you want to distribute with your application. You can run the .NET application only after the setup file is extracted to an IIS server. The contents of the *License.rtf* file display in the setup file extraction wizard.

After you create and distribute the MSI file to an IIS server, you must extract the MSI file on the server. By default the extraction directory is set to *C:\Program Files\Webform\applicationName*, and the extraction wizard creates the *C:\Program Files\Webform\applicationName\applicationName* and *C:\Program Files\Webform\applicationName\applicationName_root* virtual directories, where *applicationName* is the name of your application.

Although you do not need to modify the default extraction directory to run the application, the extraction wizard does let you change the location of the application directories you extract. If you prefer to keep all your applications directly under the server's virtual root, you could set the extraction directory to server's *Inetpub\wwwroot* directory.

Deploying to a production server

You can deploy a Web Forms application to a production server either by:

- Extracting an MSI file that you build from a Web Forms project
- Deploying directly from the development machine to a mapped server
- Copying all application folders and files from IIS on a local server to IIS on a production server

Production servers must meet the requirements described in “Configuring ASP.NET for a .NET Web project” on page 6. You must install all database clients and have access to all data sources on the production machine. For applications that you deploy to a production server, you should add required database driver DLLs to the Win32 dynamic library list on the Library Files tab page of your Web Forms projects. If you are using ODBC to connect to a database, you should add the *PBODB110.INI* file to the list of resource files on the Resource Files tab page of Web Forms projects.

The production server must have the following DLLs in its system path: *atl71.dll*, *msvcr71.dll*, *msvcp71.dll*, *pbshr110.dll*, and if your application uses DataWindow objects, *pbdwm110.dll*. You can also use the Runtime Packager to deploy required PowerBuilder runtime files to the ASP.NET server. After you install the package created by the Runtime Packager, you must restart the server.

For a complete list of required runtime files and for information on the Runtime Packager, see “Deploying Applications and Components” in *Application Techniques*.

Deploying to a remote server

You can deploy directly to a mapped server only if the server is in the same domain or workgroup as the development computer. In addition, you must add the development machine user’s Windows login ID as a member of the Administrators group on the remote machine hosting the IIS server. Also, you must share the wwwroot directory with the `wwwroot$` share name.

If you copy a Web Forms application from a development machine to a production server, you must copy both the *applicationName* and *applicationName_root* folders (and their contents) that were created when you deployed the application locally. Direct deployment to a mapped server automatically adds the necessary ASP.NET user permissions to access these directories, but if you copy files to the server, you must add these permissions manually.

About the directory file structure

For information on the directory file structure of a deployed Web Forms application under the IIS virtual root directory (`\\inetpub\wwwroot`), see “Using the Web Forms File Manager” on page 60.

Granting ASP .NET user permissions

If you copy files to a production server, or extract your Web Forms application from an MSI file, you can use Windows Explorer to grant ASP.NET permissions to the application directories. This method is described in “Setting up a SQL Anywhere database connection” on page 10. You can also grant ASP.NET permissions from a command line. The commands are different depending on whether your server is running IIS 5, 6, or 7.

Table 2-3: Commands granting permissions to Web Forms directories

IIS version	Commands for granting appropriate user permissions
5	<pre>cacls applicationName\temp /t /e /c /g ASPNET:f cacls applicationName_root /t /e /c /g ASPNET:f</pre>
6	<pre>cacls applicationName\temp /t /e /c /g IIS_WPG:f cacls applicationName_root /t /e /c /g IIS_WPG:f</pre>
7	<pre>cacls applicationName\temp /t /e /c /g IIS_IUSRS:f cacls applicationName_root /t /e /c /g IIS_IUSRS:f</pre>

Event logging on the production server

If you log Web Forms application events to a production server’s event log (by setting the PBTraceTarget global property to “EventLog”), you must have a registry entry key for PBExceptionTrace. If you use an MSI file to deploy an application to a production server, the PBExceptionTrace key is created automatically. If you deploy directly to a mapped production server or if you copy a Web Forms application to a production server, you must import the PBExceptionTrace key or create it manually.

When you deploy to a local machine, PowerBuilder creates the following key: `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\PBExceptionTrace`. You can export this key to a .REG file and import it to the production server’s registry.

For information on the PBTraceTarget global property, see “Global Web configuration properties” on page 74.

If your Web Forms application uses any ActiveX DLLs, such as *HTML2RTF.DLL* or *RTF2HTML.DLL*, you must also register these files on the production server.

Running the project from a Web browser

When you run the project from PowerBuilder, the Web browser that opens does not include the browser menu and toolbar. This is because PowerBuilder does not append the starting page, *default.aspx*, to the URL listed in the project. You can see the application in a browser window that includes the browser menu and toolbar by typing the URL in the browser location window or address bar. The URL address is not case-sensitive.

Starting an application with command line parameters

If your application requires command line parameters, you can assign values to the `PBCommandParm` global property before you deploy the application. For information on setting global properties, see “Global Web configuration properties” on page 74.

Application users can override the `PBCommandParm` parameter set at design time by adding it at the end of the application’s URL, preceded by a question mark. Multiple parameters are separated by the ASCII character code for an empty space (%20). For example, the following address, entered on a single line, uses two start-up parameters for the `mypbapp` Web Forms application deployed to the `www.mysite.com` Web site:

```
http://www.mysite.com/mypbapp/default.aspx?PBCommandParm=p1%20p2
```

If you do not include the starting page, `default.aspx`, in a URL that you type in a browser address bar, or if you append `default.htm` as the starting page, IIS still redirects you to the `default.aspx` page, but the browser menu and toolbar do not display.

Sharing data across sessions

Sharing DataWindow objects

You can share the data from primary, delete, and filter buffers of read-only DataWindow objects across Web Forms application sessions. The `Web.config` file global property `PBCachedAndSharedDWs` is available for this purpose. You must set its value to the string of comma-delimited names of the DataWindow objects you want to share across application sessions.

For information on modifying global properties, see “Configuring ASP.NET for a .NET Web project” on page 6.

The following restrictions apply to DataWindow controls that have a DataWindow object included in the `PBCachedAndSharedDWs` property setting:

- Only a single invocation of `Retrieve` is allowed, and the `Retrieve` call must not include parameters.
- No filtering or sorting is allowed.
- No deletions, insertions, data modifications, or updates are allowed.

Sharing DDDW objects

- No invocation of `ShareData` or `ShareDataOff` is allowed.

When this form of sharing is used, the retrieval events are not fired. This is because the `Retrieve` method shares the data in the cache and no actual retrieval occurs.

It is also possible to share the data of `DropDownDataWindow` objects across Web Forms application sessions. The global property `PBCachedAndSharedDDDWs` is used for this purpose. You can set its value to a string of comma-delimited names of `DataWindow` objects. Each `DataWindow` object that you list can then be shared as the child `DataWindow` of a `DropDownDataWindow` column.

Response windows affect DDDW visibility

In a Web Forms application, response windows are components of main windows rather than separate browser instances. By default, when a response window is opened, DDDW columns are temporarily hidden behind a layer displaying the response window. The columns become visible again when the response window is closed. You can prevent the temporary visibility issue by changing the value of the `PBDataWindowEnableDDDW` global property.

For more information about rendering DDDW columns, see “Modified display of `DataWindow` objects and controls” on page 110.

The following restrictions apply to `DataWindowChild` object references included in the `PBCachedAndSharedDDDWs` property setting:

- No invocation of `Retrieve` is allowed.
- No filtering or sorting is allowed.
- No deletions, insertions, data modifications, or updates are allowed.
- No invocation of `ShareData` or `ShareDataOff` is allowed.

Client-Side Events and Default Event Handlers

About this chapter

This chapter describes the client-side events available to Web Forms DataWindow controls and the default JavaScript event handlers that post back to the server.

Contents

Topic	Page
About client-side programming	29
Default event handlers	31
Client-side support for the Web DataWindow control	33
Alphabetical list of Web DataWindow client-side events	35

About client-side programming

Using client-side events to interrupt default event handlers

The use of client-side events can improve application performance because they do not require round trips to the server. In most cases, an event that is triggered in a PowerBuilder Web Forms application calls a default JavaScript event handler that posts back to the server and triggers the same event on the server side control. However, when you code a client-side event for a DataWindow control, the call to the default JavaScript event handler for that event is aborted and the round trip to the server can be avoided.

To code for a client-side event at design time, you must enclose an event handler assignment in a conditional compilation code block in a PowerBuilder painter Script view. The start tag for the code block includes a symbol to indicate that the code inside the block is for a .NET Web Forms application. The event handler assignment is a hook into a JavaScript file that you also assign in a conditional compilation code block.

Although coding for a client-side event normally interrupts postbacks to the server, you can explicitly code for a postback in your customized JavaScript event handler by calling `Document.Form.Submit` or by calling a default event handler for the triggered event.

Example code for an event handling script

The following is an example of a customized, client-side JavaScript event handler for the `ItemChanged` event of a `DataWindow`. The event handler determines whether the item changed is in the first or second column of the `DataWindow`. If the item is in one of the first two columns, this event handler calls the default JavaScript event handler that rejects item changes. In this case, the default event handler does not cause a postback. If the item changed is not in the first or second column, no client-side action is taken, and the server-side action is delayed until a postback is triggered by a different event or function call:

```
//Start MyScriptFile.js
function MyItemChanged(sender, rowNumber, columnName,
newValue)
{
    if(columnName == "column1" || columnName == "column2")
    {
        // The default function is invoked
        return PBDataWindow_ItemChangedReject(sender,
            rowNumber, columnName, newValue)
    }
    else
    {
        //do nothing
    }
}
//End MyScriptFile.js
```

The hook into the customized JavaScript event handler is added at design time in a conditional compilation code block:

```
#IF DEFINED PBWEBFORM THEN
dw_1.JavaScriptFile = "MyScriptFile.js"
dw_1.OnClientItemChanged = "MyItemChanged"
#END IF
```

Default event handlers and postbacks

The default event handlers for the `ItemChanged` and `ItemError` events do not trigger postbacks. If active, the default `ItemChanged` event handler returns immediately to reject the entered value or causes the Web Forms application to wait for a cascade of user events to occur before a postback is allowed. The cascade of events that must occur before a postback is triggered is: `ItemChanged`, `Clicked`, `RowFocusChanging`, `RowFocusChanged`, and `ItemFocusChanged`.

Some versions of the default Clicked event handler set a timer for postbacks because the DHTML DoubleClicked event also triggers the Clicked event.

If a DataWindow object's HTMLGen.PagingMethod property is set to `XMLClient!`, postbacks are not called until an Update is issued, since the data is stored in its entirety in the client browser cache. Also, if the corresponding server-side event does not contain any script, the default event handlers do not cause a postback or cause client-side Web Forms to be re-rendered.

For more information on default event handlers, see “Default event handlers” next.

Default event handlers

Default event handlers for the Web DataWindow control are contained in the *PBDataWindow.js* file that deploys with your application to the *applicationName*\Scripts directory under the server's virtual root. The default event handlers typically cause a postback or delayed postback to the corresponding server-side event. Default event handlers can call more than one server-side event, but each default event handler name includes a reference to the main event that it handles.

The choice of handlers that attach to each event follows the logic described by Table 3-1. The table also indicates whether the event handler causes a postback, a delayed postback, or no postback.

If you call a customized client-side event handler, the default event handler does not get invoked, postbacks are not made to the server, and the corresponding server-side event does not get triggered. You can explicitly call a default event handler from a customized event handler if you want to trigger the corresponding server-side event. When you call a default event handler directly in a JavaScript function, you must use the same arguments and return value that you would for the principal client-side event that it handles.

For information on client-side event signatures, see the event descriptions under “Alphabetical list of Web DataWindow client-side events” on page 35.

Table 3-1: List of default event handlers by event type

Client-side Event	Default JavaScript handler (postback action)	Used under the following conditions for server-side events:
Clicked	PBDataWindow_Clicked (postback)	Clicked is handled, but DoubleClicked is not
		Clicked and ButtonClicked are handled, but DoubleClicked is not
		Clicked and ButtonClicking is handled, but DoubleClicked is not
	PBDataWindow_DelayedClicked (delayed postback)	Clicked and DoubleClicked are handled
		Clicked, DoubleClicked, and ButtonClicked are handled
PBDataWindow_ClickedDifferentRow (postback)	RowFocusChanging is handled, but Clicked and DoubleClicked are not	
	RowFocusChanged is handled, but Clicked and DoubleClicked are not	
PBDataWindow_DelayedClickedDifferentRow (delayed postback)	RowFocusChanging and DoubleClicked are handled, but Clicked is not	
	RowFocusChanged and DoubleClicked are handled, but Clicked is not	
DoubleClick	PBDataWindow_DoubleClicked (postback)	DoubleClick is handled
RButtonDown	PBDataWindow_RButtonDown (postback)	RButtonDown is handled
ButtonClicked	PBDataWindow_ButtonClicked (postback)	ButtonClicked is handled and/or ButtonClicking is handled
ButtonClicking	PBDataWindow_ButtonClicking (postback)	ButtonClicked is handled and/or ButtonClicking is handled
ItemFocusChanged	PBDataWindow_ItemFocusChanged (postback)	ItemFocusChanged is handled
	PBDataWindow_ItemFocusChanged_AND_ItemChanged_OR_ItemError (postback)	ItemChanged and ItemError are handled, but ItemFocusChanged is not
	PBDataWindow_ItemFocusChanged_AND_ItemChanged (postback)	ItemChanged is handled, but ItemFocusChanged and ItemError are not
	PBDataWindow_ItemFocusChanged_AND_ItemError (postback)	ItemError is handled, but ItemChanged and ItemFocusChanged are not
ItemError	PBDataWindow_ItemError (no postback)	ItemChanged is handled and/or ItemError is handled
ItemChanged	PBDataWindow_ItemChangedReject (no postback)	ItemChanged is handled

Client-side Event	Default JavaScript handler (postback action)	Used under the following conditions for server-side events:
RowFocusChanged	PBDataWindow_RowFocusChanged (postback)	RowFocusChanging is handled, but ItemFocusChanged is not
		RowFocusChanged is handled, but ItemFocusChanged is not

Client-side support for the Web DataWindow control

The Web Forms version of the DataWindow is a subclass of the DataWindow .NET™ Web DataWindow control. The client-side programming capabilities of the Web DataWindow enable the use of client-side JavaScript event handlers.

The ClientEvent properties of the Web DataWindow have also been exposed, allowing the creation of customized event handlers that can override the default event handlers in the *PBDataWindow.js* file. The names of the ClientEvent properties consist of the name of a client-side event with an “OnClient” prefix. For example, the ClientEvent property that corresponds to the Clicked event would be OnClientClicked. You can circumvent the default event handler for the Clicked event by setting OnClientClicked to the name of a JavaScript function that uses the client-side Clicked event arguments.

The Web DataWindow client control supports the events listed in Table 3-2. The signatures of the client-side events and the effects of their return values are the same as for the Web DataWindow control in DataWindow .NET. For a description of each event, see “Alphabetical list of Web DataWindow client-side events” on page 35.

Table 3-2: Web DataWindow control client-side events

Event	Arguments	Return Codes
ButtonClicked	<i>sender, rowNumber, buttonName</i>	0 – Continue processing
ButtonClicking	<i>sender, rowNumber, buttonName</i>	0 – Execute action assigned to button, then trigger ButtonClicked 1 – Do not execute action or trigger ButtonClicked
Clicked	<i>sender, rowNumber, objectName</i>	0 – Continue processing 1 – Prevent focus change

Event	Arguments	Return Codes
DoubleClicked	<i>sender, rowNum, objectName</i>	0 – Continue processing 1 – Prevent focus change
ItemChanged	<i>sender, rowNum, columnName, newValue</i>	0 – Accept data value 1 – Reject data value and prevent focus change 2 – Reject data value but allow focus change
ItemError	<i>sender, rowNum, columnName, newValue</i>	0 – Reject data value and show error message 1 – Reject data value with no error message 2 – Accept data value 3 – Reject data value but allow focus change
ItemFocusChanged	<i>sender, rowNum, columnName</i>	0 – Continue processing
RButtonDown	<i>sender, rowNum, objectName</i>	0 – Continue processing 1 – Prevent focus change
RowFocusChanged	<i>sender, newRowNumber</i>	0 – Continue processing
RowFocusChanging	<i>sender, currentRowNumber, newRowNumber</i>	0 – Continue processing 1 – Prevent focus change

About return values for DataWindow events

In client events, you can use a return statement as the last statement in the event script. The datatype of the value is number.

For example, in the ItemChanged event, set the return code to 2 to reject an empty string as a data value:

```
if (newValue = "") {
    return 2;
}
```

This example prevents focus from changing if the user tries to go back to an earlier row:

```
function dwCustomer_RowFocusChanging(sender,
    currentRowNumber, newRowNumber)
{
    if (newRowNumber < currentRowNumber)
        { return 1; }
}
```

This example displays a message box informing the user which column and row number were clicked:

```
function dwCustomer_Clicked(sender, rowNum,
    objectName)
{
    alert ("You clicked the " + objectName +
        " column in row " + rowNum)
}
```

Note that displaying an Alert message box for all clicked objects in a DataWindow could prevent data entry or modification.

Alphabetical list of Web DataWindow client-side events

The list of Web DataWindow control client-side events follows in alphabetical order.

For information on calling client-side scripts, see “About client-side programming” on page 29.

ButtonClicked

Description Occurs when the user clicks a button inside a DataWindow object.

Argument	Description
sender	String. Identifier for the button the user clicked.
row	Number. The number of the current row when the user clicked the button.
objectName	String. The name of the control within the DataWindow under the pointer when the user clicked.

Applies to Web DataWindow client control

Return codes There are no special outcomes for this event. The only code is:

0 Continue processing

Usage ButtonClicked fires only for buttons with the UserDefined action. Other buttons cause the page to be reloaded from the server. The ButtonClicked event executes code after the action assigned to the button has occurred.

Postback calls to the ButtonClicked server-side event

The corresponding server-side event can be triggered by the following default event handlers: PBDDataWindow_ButtonClicked, PBDDataWindow_ButtonClicking, PBDDataWindow_Clicked, and PBDDataWindow_DelayedClicked.

This event is fired only if you have not selected Suppress Event Processing for the button. If Suppress Event Processing is on, only the Clicked event and the action assigned to the button are executed when the button is clicked.

If Suppress Event Processing is off, the Clicked event and the ButtonClicked event are fired. If the return code of the ButtonClicking event is 0, the action assigned to the button is executed and the ButtonClicked event is fired. If the return code of the ButtonClicking event is 1, neither the action nor the ButtonClicked event is executed.

See also ButtonClicking

ButtonClicking

Description Occurs when the user clicks a button. This event occurs before the ButtonClicked event.

Argument	Description
sender	String. Identifier for the button the user clicked.
row	Number. The number of the current row when the user clicked the button.
objectName	String. The name of the control within the DataWindow under the pointer when the user clicked.

Applies to Web DataWindow client control

Return codes Set the return code to affect the outcome of the event:

- 0 Execute the action assigned to the button, then trigger the ButtonClicked event
- 1 Prevent the action assigned to the button from executing and the ButtonClicked event from firing

Usage Use the ButtonClicking event to execute code before the action assigned to the button occurs. If the return code is 0, the action assigned to the button is then executed and the ButtonClicked event is fired. If the return code is 1, the action and the ButtonClicked event are inhibited.

Postback calls to the ButtonClicking server-side event

The corresponding server-side event can be triggered by the following default event handlers: PBDDataWindow_ButtonClicked, PBDDataWindow_ButtonClicking, PBDDataWindow_Clicked, and PBDDataWindow_DelayedClicked.

This event is fired only if you have not selected Suppress Event Processing for the button.

The Clicked event is fired before the ButtonClicking event.

See also ButtonClicked

Clicked

Description

Occurs when the user clicks anywhere in a DataWindow control.

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row the user clicked.
objectName	String. The name of the control within the DataWindow under the pointer when the user clicked.

Applies to

Web DataWindow client control

Return codes

Set the return code to affect the outcome of the event:

- 0 Continue processing
- 1 Prevent the focus from changing

Usage

When the user clicks on a DataWindow button, the Clicked event occurs before the ButtonClicking event. When the user clicks anywhere else, the Clicked event occurs when the mouse button is released.

Postback calls to the Clicked server-side event

The corresponding server-side event can be triggered by the following default event handlers: PBDDataWindow_Clicked and PBDDataWindow_DelayedClicked.

Examples

This script in an *.aspx* file that submits the value of the selected row in the DataWindow to the server:

```
function objdwcCustomers_Clicked(sender, rowNumber,
objectName) {
    document.Form1.rownum.value = rowNumber;
    document.Form1.submit();
}
```

See also

ButtonClicked
 ButtonClicking
 DoubleClicked
 ItemFocusChanged
 RButtonDown
 RowFocusChanged
 RowFocusChanging

DoubleClick

Description Occurs when the user double-clicks anywhere in a DataWindow control.

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row the user double-clicked.
objectName	String. The name of the control within the DataWindow under the pointer when the user double-clicked.

Applies to Web DataWindow client control

Return codes Set the return code to affect the outcome of the event:

- 0 Continue processing
- 1 Prevent the focus from changing

Usage When the user double-clicks on a DataWindow button, the DoubleClicked event occurs before the ButtonClicking event. When the user double-clicks anywhere else, the DoubleClicked event occurs when the mouse button is released.

Postback calls to the DoubleClicked server-side event

The corresponding server-side event can be triggered by the following event handlers: PBDDataWindow_DoubleClicked, PBDDataWindow_DelayedClicked, and PBDDataWindow_DelayedClickedDifferent Row.

Examples This script in an *.aspx* file submits the value of the selected row in the DataWindow to the server:

```
function objdwCustomers_DoubleClicked(sender,
    rowNumber, objectName) {
    document.Form1.rownum.value = rowNumber;
    document.Form1.submit();
}
```

See also ButtonClicked
ButtonClicking
Clicked
RButtonDown

ItemChanged

Description Occurs when a field in a DataWindow control has been modified and loses focus (for example, the user presses Enter, the Tab key, or an arrow key or clicks the mouse on another field within the DataWindow). It occurs before the change is applied to the item. ItemChanged can also occur when the Update function is called.

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row containing the item whose value is being changed.
columnName	String. The name of the column containing the item.
newValue	String. The new data the user has specified for the item.

Applies to Web DataWindow client control

Return codes Set the return code to affect the outcome of the event:

- 0 (Default) Accept the data value
- 1 Reject the data value and do not allow focus to change
- 2 Reject the data value but allow the focus to change

Usage The ItemChanged event does not occur when the DataWindow control itself loses focus.

Postback calls to the server-side ItemChanged event

The corresponding server-side event is triggered by default event handlers only after a cascade of events occurs: ItemChanged, Clicked, RowFocusChanging, RowFocusChanged, and ItemFocusChanged. Default postback scripts for this event are PBDDataWindow_ItemFocusChanged_AND_ItemChanged and PBDDataWindow_ItemFocusChanged_AND_ItemChanged_OR_ItemError. The default event handler PBDDataWindow_ItemChangedReject does not cause a postback, but instead rejects the changed value entered by the user.

See also ItemError

ItemError

Description Occurs when a field has been modified, the field loses focus (for example, the user presses Enter, Tab, or an arrow key or clicks the mouse on another field in the DataWindow), and the data in the field does not pass the validation rules for its column.

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row containing the item whose new value has failed validation.
columnName	String. The name of the column containing the item.
newValue	String. The new data the user has specified for the item.

Applies to Web DataWindow client control

Return codes Set the return code to affect the outcome of the event:

- 0 (Default) Reject the data value and show an error message box
- 1 Reject the data value with no message box
- 2 Accept the data value
- 3 Reject the data value but allow focus to change

Usage If the Return code is 0 or 1 (rejecting the data), the field with the incorrect data regains the focus.

The ItemError event occurs instead of the ItemChanged event when the new data value fails a validation rule. You can force the ItemError event to occur by rejecting the value in the ItemChanged event.

Postback calls to the server-side ItemError event

Default postback scripts for this event are called only after an ItemFocusChanged event occurs on the client side. The default event handlers that invoke the server-side ItemError event are PBDDataWindow_ItemFocusChanged_AND_ItemError and PBDDataWindow_ItemFocusChanged_AND_ItemChanged_OR_ItemError. The default event handler PBDDataWindow_ItemError does not cause a postback, but instead rejects the value entered by the user.

Examples This script in the .aspx file displays an alert message:

```
function objwdw_ItemError(sender, rowNumber,
    columnName, newValue) {
    alert("ItemError: " + rowNumber + columnName +
        newValue);
}
```

See also [ItemChanged](#)

ItemFocusChanged

Description Occurs when the current item in the control changes.

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row containing the item that has just gained focus.
columnName	String. The name of the column containing the item.

Applies to Web DataWindow client control

Return codes There are no special outcomes for this event. The only code is:

- 0 Continue processing

Usage ItemFocusChanged occurs when focus is set to another column in the DataWindow, including when the DataWindow is first displayed. The row and column together uniquely identify an item in the DataWindow.

In Web Forms targets, once a DataWindow loses focus and a postback event is triggered, the DataWindow loses memory of its current column. If the same cell regains the focus, the ItemFocusChanged event is triggered because the current column is lost after the page posts back to the client.

Postback calls to the server-side ItemFocusChanged event

The corresponding server-side event can be triggered by the following event handler: `PBDataWindow_ItemFocusChanged`.

See also [RowFocusChanged](#)
[RowFocusChanging](#)

RButtonDown

Description Occurs when the user right-clicks anywhere in a DataWindow control.

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row the user right-clicked.
objectName	String. The name of the control within the DataWindow under the pointer when the user right-clicked.

Applies to Web DataWindow client control

Return codes Set the return code to affect the outcome of the event:

- 0 Continue processing
- 1 Prevent the focus from changing

Usage When the user right-clicks on a DataWindow button, the RButtonDown event occurs before the ButtonClicking event. When the user right-clicks anywhere else, the RButtonDown event occurs when the mouse button is released.

Postback calls to the RButtonDown server-side event

The corresponding server-side event can be triggered by the following event handler: PBDDataWindow_RButtonDown.

Examples This script in an *.aspx* file submits the value of the selected row in the DataWindow to the server:

```
function objdwCustomers_RButtonDown(sender,
    rowNumber, objectName) {
    document.Form1.rownum.value = rowNumber;
    document.Form1.submit();
}
```

See also ButtonClicked
ButtonClicking
Clicked
DoubleClick

RowFocusChanged

Description Occurs when the current row changes in the DataWindow.

Argument	Description
sender	String. Identifier for the client-side control.
newRow	Number. The number of the row that has just become current.

Applies to Web DataWindow client control

Return codes There are no special outcomes for this event. The only code is:

0 Continue processing

Usage The SetRow function, as well as user actions, can trigger the RowFocusChanged and ItemFocusChanged events.

Postback calls to the server-side RowFocusChanged event

The corresponding server-side event can be triggered by the following default event handlers: PBDataWindow_RowFocusChanged, PBDataWindow_ClickedDifferentRow, and PBDataWindow_DelayedClickedDifferentRow.

Examples This script in the *.aspx* file displays an alert message when the row focus changes:

```
function objdw_RowFocusChanged(sender, newRowNumber) {  
    alert("Focus changed to row " + newRowNumber);  
}
```

See also ItemFocusChanged
RowFocusChanging

RowFocusChanging

Description Occurs when the current row is about to change in the DataWindow. (The current row of the DataWindow is not necessarily the same as the current row in the database.)

The RowFocusChanging event occurs just before the RowFocusChanged event.

Argument	Description
sender	String. Identifier for the client-side control.
currentRow	Number. The number of the row that is current (before the row is deleted or its number changes). If the DataWindow object is empty, currentrow is 0 to indicate there is no current row.
newRow	Number. The number of the row that is about to become current. If the new row is going to be an inserted row, newrow is 0 to indicate that it does not yet exist.

Applies to Web DataWindow client control

Return codes Set the return code to affect the outcome of the event:

- 0 Continue processing (setting the current row)
- 1 Prevent the current row from changing

Usage Typically the RowFocusChanging event is coded to respond to a mouse-click or keyboard action that would change the current row in the DataWindow object.

Postback calls to the server-side RowFocusChanging event

The corresponding server-side event can be triggered by the following default event handlers: PBDDataWindow_RowFocusChanged, PBDDataWindow_ClickedDifferentRow, and PBDDataWindow_DelayedClickedDifferentRow.

See also ItemFocusChanged
RowFocusChanged

User Management and Registry Operations in Web Forms

About this chapter

This chapter describes how to create permanent user accounts for Web Forms applications and the use of registry functions in these applications.

Contents

Topic	Page
Creating permanent user accounts	47
Managing permanent user accounts	51
Using the registry functions	53

Creating permanent user accounts

Using the ASP.NET membership feature

Due to the stateless nature of the HTTP protocol, file and directory operations in a Web application typically do not persist after a user session has ended. However, for PowerBuilder .NET Web Forms applications, you can store user names in a database and persist data and files created by application users across Web Forms sessions. PowerBuilder .NET Web Forms can take advantage of the ASP.NET membership feature to maintain permanent user accounts and store files created or modified by application users.

The default ASP.NET membership provider is defined in the *machine.config* file that is typically installed in the `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG` directory. The *machine.config* file assigns SQL Server Express (`.\SQLEXPRESS`) as the default membership data source.

Installing SQL Server Express

You can download the SQL Server 2005 Express Edition from the Microsoft download site at <http://www.microsoft.com/downloads/details.aspx?familyid=220549b5-0b07-4448-8848-dcc397514b41&displaylang=en>.

If you are using a different DBMS

Before you install SQL Server Express, you might need to uninstall SQL Native Client before installing SQL Server 2005 Express. The SQL Server Express setup produces an error if it finds an incompatible version of SQL Native Client. The SQL Server Express setup includes a compatible version of SQL Native Client that it installs if it does not find an existing version of this driver on the server computer.

You do not need to install SQL Server Express if you are using SQL Server for the permanent user database, but then you must replace the default connection string in the *machine.config* file or add a connection string to the *web.config* file for the Web Forms application. Changes to the *machine.config* file affect all .NET Web applications.

The connection string you add to the *web.config* file should have the following format for a remote database server using SQL Authentication:

```
<connectionStrings>
  <add name="MySQLServer"
    connectionString="Server=dbsevername;
    Database=aspnetdb; User Id=uid; password=pwd"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

The following connection string specifies a local SQL Server database with Windows Authentication (SSPI):

```
<connectionStrings>
  <add name="MyDbConn"
    connectionString="Initial Catalog=MyDb;
    Data Source=MyServer;
    Integrated Security=SSPI;" />
</connectionStrings>
```

Creating permanent user accounts

Permanent user accounts are disabled by default. After a Web Forms application is successfully deployed, the IIS server administrator can use the following procedure to set up permanent user accounts. This procedure uses the default ASP.NET membership provider and data source assignment, although a note in the procedure describes a required change to the *web.config* file when you use a nondefault connection string.

❖ **To create permanent user accounts:**

- 1 Run `aspnet_regsql` from a DOS command prompt window.
This starts the ASP.NET SQL Server Setup wizard.

You can enter `Aspnet_regsql /?` at the command line to obtain a list of optional parameters for SQL Server or SQL Server Express that you can use to bypass the wizard. If you are using a local SQL Server database, for example, you can enter `aspnet_regsql -S (local) -E -A m`, where `-S (local)` indicates that the server is local, `-E` indicates that the connection uses Windows Authentication, and `-A m` adds the membership feature.

- 2 Use the ASP.NET SQL Server Setup wizard to create the user database.
In the wizard, you can enter `.\SQLEXPRESS` as the server name. This is the default name for the local server as defined in the *machine.config* file. The wizard should give you a success message.
- 3 In a text editor, modify the *web.config* file in the virtual root directory for your deployed Web Forms application (typically, `C:\Inetpub\wwwroot\applicationName`) to remove comment tags (`<!--` and `-->`) from the following lines:

```
<!-- <roleManager enabled="true" /> -->
...
<!-- <add name="AspNetSqlMembershipProvider"
...
passwordStrengthRegularExpression="" /> -->
```

These lines should now appear in the *web.config* file as:

```
<roleManager enabled="true" />
...
<add name="AspNetSqlMembershipProvider"
...
passwordStrengthRegularExpression="" />
```

If you are using a nondefault database

You can change the `connectionStringName` parameter assignment in the uncommented lines to a value other than `LocalSqlServer` if you modified the connection string in the *machine.config* file, or if you added a connection string in the application *web.config* file. For example, if you named a connection string as `MySqlServer`, you should change the `connectionStringName` parameter value to `MySqlServer`.

- 4 Make sure that the appropriate user or user group (ASPNET for IIS 5, IIS_WPG for IIS 6, or IIS_USRS for IIS 7) has write authority for the virtual root directory of your deployed Web Forms application.

The next step in this procedure creates the App_Data directory and saves database files to this directory. However, it cannot do this if the ASPNET user (Windows XP), the IIS_WPG user group (Windows 2003), or the IIS_USRS user group (Windows Vista) does not have write authority for the Web Forms virtual root directory.

If you proceed to the next step without modifying write permissions, you might get an error page indicating that you do not have write authority for this directory. The error page also explains how to grant write authority for the directory.

- 5 Open the *UsersInit.aspx* file for the Web Forms application in a Web browser.

The full URL for this file is typically

<http://ServerName/ApplicationName/UsersInit.aspx>. When you open this file in a Web browser, the App_Data directory is created under the application virtual root directory and the permanent accounts database is created with the following entries:

User name	Password	Role
admin	a123456&	admin
user	a123456&	user

The *UsersInit.aspx* returns a success page after the above user accounts are created.

Delete the UsersInit.aspx file after this step

For security reasons, you should delete the *UsersInit.aspx* file after you go to the next step or complete this procedure.

- 6 Open the *Login.aspx* file for the Web Forms application in a Web browser.

The full URL for this file is typically

<http://ServerName/ApplicationName/Login.aspx>. The Login page displays.

- 7 On the Login page, enter `admin` for the User Name and `a123456&` for the Password.

The Welcome page for an administrator role has a hyperlink labeled Users that opens a page to manage users.

- 8 Click the Users hyperlink, then click the Search button in the page to manage users.

The page for managing users displays the application users in the permanent user database.

User Name: [Create New User](#) [Logout](#)

Username	Email	Last Activity Date	Is Online	User Details
admin	admin@admin	4/19/2006 11:32:52 AM	<input checked="" type="checkbox"/>	Edit
user	user@user	4/19/2006 10:58:32 AM	<input type="checkbox"/>	Edit

- 9 Click the Create New User hyperlink.

The page for adding users displays.

- 10 Enter a new user name, password, and e-mail. Enter the password a second time in the Confirm Password text box and click Create User.

A new user account is added to the database for the current Web Forms application.

- 11 Select the Admin Role check box if the new user should have administrative privileges, and click Finish.

When you return to the page for managing users and click the Search button again, you should see the user account you created in the list of user accounts.

- 12 Repeat steps 10-12 to create as many user accounts as necessary, then click the Logout hyperlink to log out from the user management role.

If you redeploy

Accounts that you create are maintained in the database after you redeploy the Web Forms application, but you must edit the *web.config* file as described in step 4 above after each redeployment.

Managing permanent user accounts

In your administrator role, in addition to creating permanent user accounts, you can edit, delete, and unlock accounts, and you can reset user passwords. To perform any of these tasks, you must first set up your Web site membership provider as described in the procedure for “Creating permanent user accounts” on page 47.

❖ **To edit, delete, or unlock user accounts**

- 1 Open the *Login.aspx* file for the Web Forms application in a Web browser.

The full URL for this file is typically `http://ServerName/ApplicationName/Login.aspx`. The Login page displays.

- 2 On the Login page, log in with an admin role account.

The default login for an admin account is described in the procedure for “Creating permanent user accounts” on page 47.

The Welcome page for an administrator role has a hyperlink labeled Users that opens a page to manage users.

- 3 Click the Users hyperlink, then click the Search button in the page to manage users.

The page for managing users displays the application users in the permanent user database.

- 4 Click the Edit hyperlink for a user in the list of user accounts.

The page for editing and deleting user accounts displays. An Unlock User button appears on this page only when a user is locked out. Lockouts occur when the number of attempts to log in with a faulty password exceeds the number of attempts authorized by the `MaxInvalidPasswordAttempts` parameter in the application *web.config* file.

[Back to Users](#) [Logout Home Page](#)

Update User	Unlock User	Delete User
User Name	user	
E-mail	<input type="text" value="user@user.com"/>	
Enable	<input checked="" type="checkbox"/>	
Admin Role	<input type="checkbox"/>	
<hr/>		
New Password	<input type="text"/>	Reset Password

- 5 Enter any changes you want for the user role or e-mail, and click Update User to apply those changes.

The Update User button also applies your selections for whether the membership user can be authenticated (Enable check box) and whether the user has administrative privileges (Admin Role check box).

- 6 If you want to change the user password, enter a new password for the user account and click Reset Password to apply the change.
- 7 If the selected user account is currently locked, click Unlock User to unlock the account and allow the user to log back in.
- 8 If you want to remove the current user account from the permanent user database, click Delete User.
- 9 Repeat steps 4-8 for all the user accounts you want to edit, delete, or unlock.
- 10 Click the Logout hyperlink to log out of your user management role.

Using the registry functions

Searching and setting registry entries

PowerBuilder Web Forms applications can read registry entries at the server side and can write registry entries to a *registry.xml* file.

The RegistryGet, RegistryKeys, or RegistryValues functions can search the server registry for registry entries if a *registry.xml* file does not exist in the *applicationName_root/File/Common* directory under the IIS virtual root, where *applicationName* is the name of the current Web Forms application. The registry search functions also search the server registry when the *registry.xml* file exists but the entries you are searching for are not contained in the current application's *registry.xml* file.

The RegistrySet function creates the *registry.xml* file—if one does not already exist—in the *applicationName_root/File/Session/SessionID* for the current Web Forms session or in the *applicationName_root/File/User/UserName* directory when the current user has logged in with a permanent user account. If a *registry.xml* file already exists, the arguments of the RegistrySet function are added to the contents of the existing *registry.xml* file.

The RegistrySet function can also copy a *registry.xml* file from *applicationName_root/File/Common* to *applicationName_root/File/Session/SessionID* or to *applicationName_root/File/User/UserName*, but it writes to the *registry.xml* file in the *SessionID* or *UserName* directory only.

For information about permanent user accounts, see “Creating permanent user accounts” on page 47.

Rules for registry searching and setting

The following rules apply to registry search and setting operations:

- Searches for registry entries, keys, or values are conducted first in the *registry.xml* file. The search uses the server registry only if the registry entry, key, or value cannot be found in the *registry.xml* file.
- Application users can set or write registry entries only in the *registry.xml* file in the *applicationName_root/File/Session/SessionID* folder for the current session or in the *applicationName_root/File/User/UserName* folder for the current user.
- Application users can delete registry keys or values only from the *registry.xml* file in the *SessionID* folder for the current session or the *UserName* folder for the current user.

Adding a registry.xml file

The IIS server administrator can add a *registry.xml* file to the *applicationName_root/File/Common* directory under the virtual root for IIS Web sites. If the Web Forms application searches for a registry entry, key, or value, the *registry.xml* file is copied to the *applicationName_root/File/Session/SessionID* folder for the current session or to the *applicationName_root/File/User/UserName* folder for the current user.

The following is sample content of a *registry.xml* file:

```
<?xml version="1.0"?>
<RegistryRoot>
  <k name="HKEY_LOCAL_MACHINE">
    <k name="Software">
      <k name="MyApp">
        <k name="Fonts">
          <v name="Title">MyTrueType</v>
        </k>
      </k>
    </k>
  </k>
</RegistryRoot>
```

The first time the registry function is called, the system copies *registry.xml* from the Common directory to the *SessionID* or *UserName* directory; after that, the system uses the *registry.xml* copy under the *SessionID* or *UserName* directory.

Print, File, Mail Profile, and Theme Managers

About this chapter

This chapter describes the print, file, and mail profile managers that application users can access from a PowerBuilder .NET Web Forms application.

Contents

Topic	Page
Using the Web Forms Print Manager	55
Using the Web Forms File Manager	60
Using the Web Forms Mail Profile Manager	67
Using the Web Forms Theme Manager	70

Using the Web Forms Print Manager

Print function support

In Web Forms applications, output from supported PowerScript print functions is published as PDF files on the server side. These PDF files are visible in the client-side Web browser through links in the Web Forms Print Manager, and they can be printed on the client side.

The following system print functions are supported in .NET Web Forms applications: Print, PrintCancel, PrintClose, PrintDefineFontDefine, PrintLine, PrintOpen, PrintOval, PrintPage, PrintRect, PrintRoundRect, PrintSetSpacing, PrintText, PrintWidth, PrintX, PrintY. PrintSetFont is also supported, but its return value is not the same as in a standard PowerBuilder application.

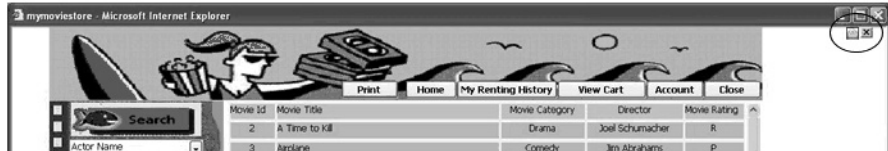
File operation output

You can also use the PowerScript SaveAs method to print DataWindows and their data as PDF or XSL files. These files are not visible in the Web Forms Print Manager and cannot be printed on the client side without the intervention of the IIS administrator.

Print Manager icon display

When supported print functions are used to print text in a Web Forms application, a printer icon displays in the right-top corner of the main browser window.

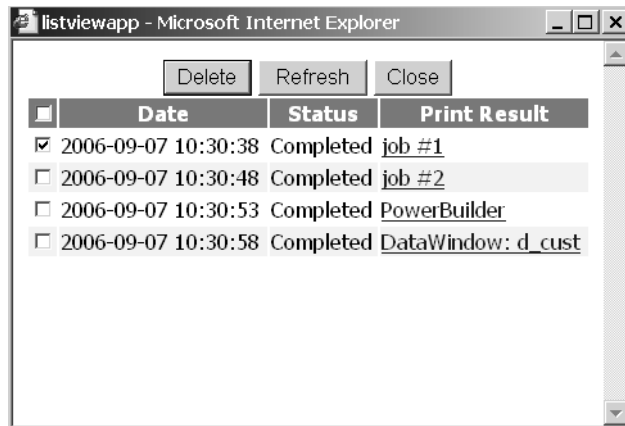
Figure 5-1: Print icon on a Web Forms application



The application user can double-click the icon to open the Web Forms application Print Manager. The Print Manager lets the application user open a window to view the printed output as PDF files.

Figure 5-2 shows the Print Manager with hyperlinks to printed files.

Figure 5-2: Print Manager for a Web Forms application



If you do not want the Print Manager icon to display on a specific window in your application, you can set the HasPrintManager property for that window to false. The Print Manager icon automatically disappears on browser refresh after all the printed files are removed from the Print Manager window.

You can also code an application event to open the Print Manager by calling the OpenPrintManager function.

For information on the HasPrintManager property, see HasPrintManager on page 82. For information on the OpenPrintManager function, see OpenPrintManager on page 90.

Where printed output is saved

Printed output is saved to files in the *applicationName_root\Print\Session\SessionID* directory under the virtual root for IIS Web sites, or in the *applicationName_root\Print\User\UserName* directory if the current application user is logged in with a permanent user account profile. The *applicationName_root\Print\Session* and the *applicationName_root\Print\User* directories are created when you deploy your application. The *SessionID* or *UserName* directory is created by the ASP.NET runtime engine after a `PrintOpen` call.

The *SessionID* directory created under the `Print\Session` directory uses the same session ID number as the subdirectory created under the *applicationName_root\File\Session* directory when the user saves a `DataWindow` as a PDF or writes to a file from the current application session, or when the `PBWebFileProcessMode` global property has been set to `Copy` mode. The actual *SessionID* directory name is a long 24-character string with letters and numbers such as `cdxge1554rkxxsbn1221uh55`. Unless the user creating the printed files has logged in as a permanent user, the *SessionID* directories are deleted when the Web Forms session is ended.

Requirements for saving files in PDF or XSL format

The default PDF printing feature uses the Sybase `DataWindow PS` printer to print output to a PostScript (PS) file, and then convert it to a PDF file format. You must grant print permissions to the `ASPNET`, `IIS_WPG`, or `IIS_USRS` user group for the Sybase `DataWindow PS` printer.

Alternatively, you could use the Apache Formatting Objects (FO) processor to save a `DataWindow` and its data in the PDF or XSL-FO format.

PostScript printing
method

The Sybase `DataWindow PS` printer profile is added automatically to a computer's printer list when you save a `DataWindow` to a PDF file from a PowerBuilder application. This does not occur automatically with a Web Forms application; however, Web Forms users can use the Sybase `DataWindow PS` printer that you create on the server computer from a standard client-server application at design time or runtime.

You can also add the Sybase `DataWindow PS` profile manually to the server computer using the Windows Add Printer wizard. If a PostScript driver has not been previously installed on the IIS server computer, the Add Printer wizard might ask you to insert the Windows installation CD.

Once a postscript driver is installed, you (or the server administrator) can add a Sybase DataWindow PS profile from the Install Printer Software page of the wizard in one of the following ways:

- Click the Have Disk button and browse to the *Adist5.inf* file (installed with PowerBuilder) in the *Shared\PowerBuilder\drivers* directory, or to another PostScript driver file.
- Select a printer with PS in its name (such as “Apple Color LW 12/660 PS”) from the list of printers of the wizard.

You must then rename the printer to “Sybase DataWindow PS” on the Name Your Printer page of the Add Printer wizard or in the Properties dialog box for the added printer.

To enable PDF printing from a Web Forms application using the postscript processing method, you must also install Ghostscript on the IIS server computer.

For more information about installing Ghostscript, see “Installing GPL Ghostscript” next.

Apache FO
processing method

If a Web Forms application uses the Apache processor to save a DataWindow and its data in PDF or XSL-FO format, you must include the *fop-0.20.4* directory and the Java Runtime Environment (JRE) on the server computer. The *bin\client* folder of the JRE must be in the server computer’s system path.

The processor directory and the JRE must be in the same path as the PowerBuilder runtime files. For example, if *pbvm110.dll* and the other PowerBuilder runtime files are included in a server computer directory called *ServerPB*, the Apache processor must be copied to *ServerPB\fop-0.20.4* and the JRE to *ServerPB\jre*, respectively. However, you do not need to place a copy of the JRE in this location if the full JDK is installed on the server computer and is in its classpath.

The following JAR files must be in the server computer’s classpath:

- *fop-0.20.4\build\fop.jar*
- *fop-0.20.4\lib\batik.jar*
- *fop-0.20.4\lib\xalan-2.3.1.jar*
- *fop-0.20.4\lib\xercesImpl-2.1.0.jar*
- *fop-0.20.4\lib\xml-apis.jar*
- *fop-0.20.4\lib\avalon-framework-cvs-20020315.jar*

You might also need to restart the IIS server before you can use this method to print to a PDF file from a Web Forms application.

DBCS platforms

If the Web Forms server computer is a DBCS platform, you also need to include a file that supports DBCS characters in the Windows font directory, for example, C:\WINDOWS\fonts. For more information about configuring fonts, see the Apache Web site at <http://xml.apache.org/fop/fonts.html>.

Installing GPL Ghostscript

To enable Web Forms users to save their data in PDF format using the postscript processing method, you must download and install Ghostscript on the IIS server computer as described in the procedure that follows. Ghostscript is not required on the client for Web Forms applications.

The use of Ghostscript is subject to the terms and conditions of the General Public License (GPL). A copy of the GPL is available on the GNU Project Web server at <http://www.gnu.org/licenses/gpl.html>.

❖ **To install Ghostscript:**

- 1 Download the self-extracting executable file for the Ghostscript version you want from one of the locations listed on the Ghostscript Web site at <http://www.ghostscript.com/awki>.
- 2 Run the executable file to install Ghostscript on the server computer.

The default installation directory is *C:\program files\gs*. You can select a different directory and/or choose to install shortcuts to the Ghostscript console and readme file.

Location of files When a Web Forms application user saves a DataWindow object as a PDF, the Web Forms server searches in the following locations for an installation of Ghostscript:

- The Windows registry
- The relative path of the *pbdwm110.dll* file (typically in the Sybase\Shared\PowerBuilder directory)
- The system PATH environment variable

If Ghostscript is installed using the Ghostscript executable file, the path is added to the Windows registry.

If the Ghostscript files are in the relative path of the *pbdwm110.dll* file, they must be installed in this directory structure:

```
dirname\pbdwm110.dll
dirname\gs\gsN.NN
dirname\gs\fonts
```

where *dirname* is the directory that contains the runtime DLLs and *N.NN* represents the release version number for Ghostscript.

For information about fonts supplied with Ghostscript, see the APFL Ghostscript Web site at <http://www.ghostscript.com/doc/current/Fonts.htm>.

You must also make sure the default PostScript printer driver and related files in *Sybase\Shared\PowerBuilder\drivers* are included in the IIS server path.

Where PDF and XSL-FO output is saved

If a full path is not provided in the SaveAs command, the PDF and XSL-FO files that users generate from Web Forms DataWindow objects are saved by default in the virtual root path for IIS Web sites under the *applicationName_root\File\Session\SessionID\currentInitialDirectory* or the *applicationName_root\File\User\UserName\currentInitialDirectory* directory, where *currentInitialDirectory* is a directory that you assign at design time for the Web Forms application.

IIS server administrator role

The IIS server administrator can change the default current initial directory by modifying the *PBCurrentDirectory* global property in the ASP.NET Configuration Settings dialog box or in the *Web.Config* file for the Web Forms application.

Using the Web Forms File Manager

Virtual file system

When you deploy a PowerBuilder application as a .NET Web Forms application, PowerBuilder creates a virtual file system that Web Forms users can access from client-side Web browsers. The virtual file system is maintained on the server. Application users can read from and write to files in the virtual file system as long as user permissions for file operations are not restricted.

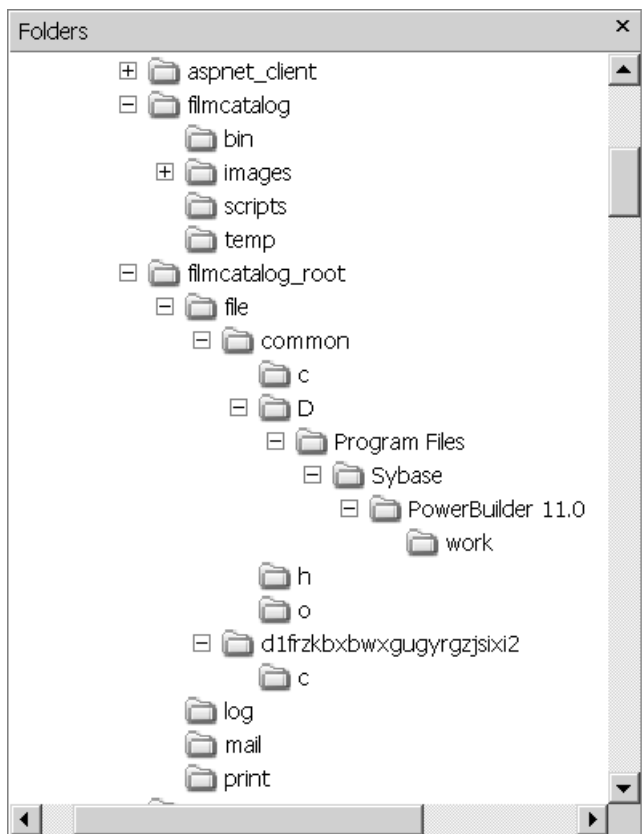
File operations with external functions

You cannot use external functions to do file operations in Web Forms targets.

The virtual file system for a PowerBuilder .NET Web Forms application is contained in the *applicationName_root*\File\Common directory under the virtual root of the IIS server, where *applicationName* is the name of the current Web Forms application.

Subdirectories of the Common directory store read-only files that are shared by all users of a Web Forms application. PowerBuilder creates these subdirectories at deployment time. A top-level subdirectory is created for each development computer drive containing a file deployed with the PowerBuilder application. The entire path to each application file is mirrored in the virtual file system to reflect the path to the application files on the desktop file system. The name of each top-level subdirectory in the virtual file system consists only of a drive letter that mirrors the desktop drive from where an application file was copied.

Figure 5-3 shows the Common directory and its subdirectories for the FilmCatalog Web Forms application. It also shows a *SessionID* subdirectory with a single subdirectory where a PDF file was written in Share mode. At runtime, the Web Forms application creates a *SessionID* folder in the File\Session directory for each user for storing files uploaded or created by that user. The exact name of the *SessionID* directory is generated by the ASP.NET runtime engine.

Figure 5-3: Virtual file system under the IIS root directory**File process mode**

There are two file process modes: Share mode and Copy mode. The `PBWebFileProcessMode` global property defines the mode for the current Web Forms application. It is set to Share mode by default.

Share mode: Files are copied from the Common directory to the `File\Session\SessionID` or the `File\User\UserName` directory only as needed.

Copy mode: In Copy mode, the first time a file operation is called, all folders and files under the Common directory are copied to the `SessionID` or `UserName` directory. In Copy mode, all file operations are handled in subdirectories of the `SessionID` or `UserName` directory.

The File Manager presents a merged view of the files under the Common and `SessionID` or `UserName` directories. If a read-only file in the Common directory has the same name as a read-write file in the `SessionID` or `UserName` directory, only the `SessionID` or `UserName` file is displayed.

Although users can delete and move folders or files that they create under the *SessionID* or *UserName* directory, files and folders that are copied from the Common directory cannot be deleted because the File Manager presents a merged view of these virtual file paths, and removing a file or folder from the *SessionID* or *UserName* directory does not cause its removal from the Common directory.

The common dialog boxes for all file operations are supported regardless of file process mode. You can display these dialog boxes with the `GetOpenFileName`, `GetSaveFileName`, and `GetFolder` functions.

Using the File Manager

When you set the `PBFileManager` property to true, the File Manager icon normally displays in every window of your Web Forms application. Users can open the File Manager at any time by clicking the File Manager icon. You can also code an application event to open the File Manager by calling the `OpenFileManager` function.

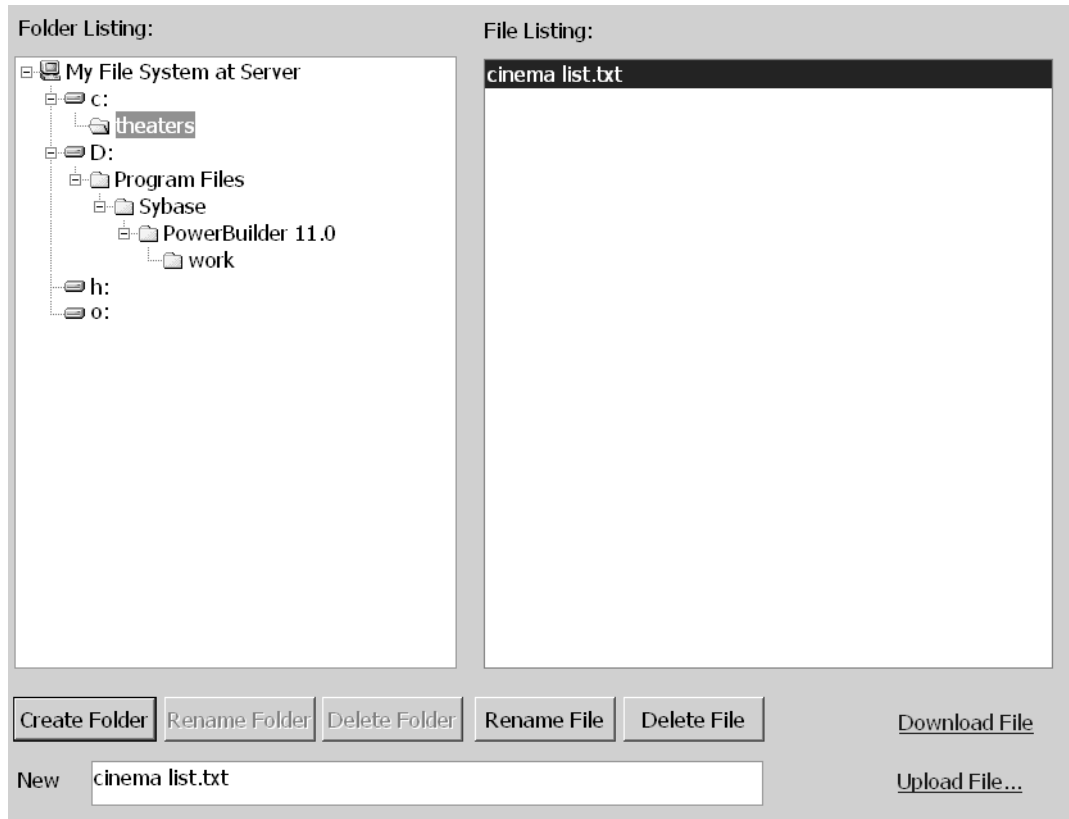
Although you can choose to render the File Manager icon at design time, you can change your selection after deployment by modifying the application's `PBFileManager` global property. If you do not want the File Manager icon to display on a specific window in your application, you can set the `HasFileManager` property for that window to false.

For information on the `HasFileManager` property, see `HasFileManager` on page 81. For information on the `OpenFileManager` function, see `OpenFileManager` on page 89.

The File Manager icon displays in the upper right corner of Web Forms, just to the right of the Mail Profile Manager icon when that icon is also rendered. The File Manager opens in the current browser window when a user clicks the File Manager icon.

Figure 5-4 shows the File Manager for a Web Forms application.

Figure 5-4: File Manager displaying an uploaded text file



Creating a directory

The File Manager allows users to create directories, rename and delete selected files or directories, and upload and download selected files. It allows users to view all files in the virtual file system for the Web Forms application unless those files are located in a directory or subdirectory listed in the PBDenyDownloadFolders global property.

❖ To create a folder using the File Manager

- 1 Select the directory in the left pane under which you want to create a folder.
- 2 Type the name you want for the new folder in the New text box.
- 3 Click Create Folder.

The new directory is created in the *SessionID* (or *UserName*) path under the directory you selected in Step 1. No other application user can use the Web Forms File Manager to see the new directory. When an application user leaves the current session, the *SessionID* directory and any files uploaded to it are removed. (If an application user is logged in with a permanent user account, the *UserName* directory and its contents are not removed.)

Renaming and deleting a directory

You can rename a directory by selecting it in the left pane, entering a new name in the New text box, and clicking Rename Folder. You delete a directory by selecting it in the left pane and clicking Delete Folder.

You cannot rename or delete a directory if it was not created in the current Web Forms session. The Rename Folder and Delete Folder buttons are disabled when a directory under the Common path of the virtual file directory is selected in the left pane of the File Manager.

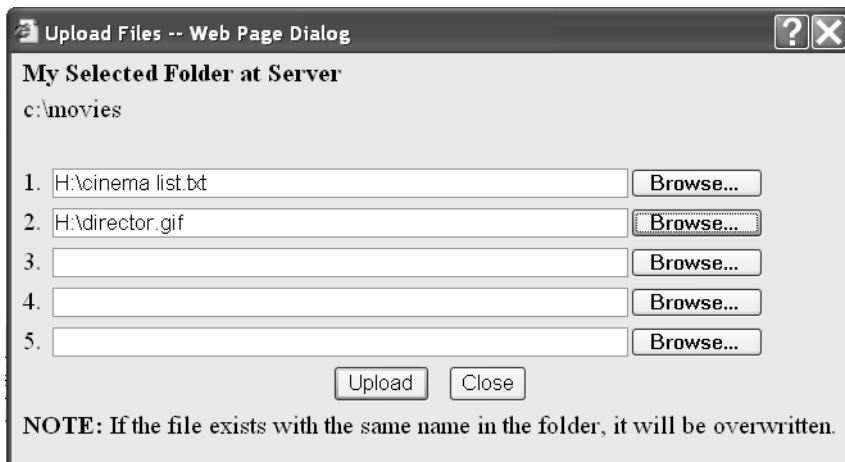
Users can close the File Manager and return to the current Web Forms window by clicking the close (x) button at the upper right corner of the manager frame.

Uploading files

The files that a user uploads through the Web Forms File Manager are saved under the *SessionID* (or *UserName*) path. The uploaded files are copied from the client-side computer. They are deleted from the server-side *SessionID* path (but not from the *UserName* path) at the end of the Web Forms session.

Figure 5-5 shows the PowerBuilder Upload File dialog box for a Web Forms application.

Figure 5-5: Upload dialog box



❖ **To upload a file using the File Manager:**

- 1 In the left pane of the File Manager, select the directory where you want to copy a file.

The Upload File link displays to the right of the New text box.

- 2 Click the Upload File link.

The PowerBuilder Upload File dialog box displays.

- 3 Type the file name or browse to the file or files you want to upload.

- 4 Click Upload.

A message displays in the dialog box to indicate whether the upload is successful.

- 5 Click Close & Refresh to close the dialog box and refresh the file listings in the right pane of the File Manager.

Downloading a file

Users can download any file listed in the right pane of the File Manager. The files are downloaded to the client-side computer from either the *SessionID* (*UserName*) or Common path on the server. (The actual server path is never displayed in the virtual file directory.)

❖ To download a file using the File Manager

- 1 From the right pane of the File Manager, select the file you want to download.

The Download File link displays near the bottom right corner of the File Manager, just above the Upload File link.

- 2 Click Download File.

The File Download dialog box lists the file name and file type and the name of the server from which the file can be downloaded. It prompts you to save the file or cancel the download. (On some operating systems, the File Download dialog box can also include Open and More Info buttons.)

- 3 Click Save.

The Save As dialog box displays.

- 4 Browse to the path on the local computer where you want to save the file, and click Save.

The Download Complete dialog box displays. Its appearance depends on the client operating system. It typically prompts you to open the downloaded file, open the folder where you saved the file, or close the dialog box.

- 5 Click Close to close the Download Complete dialog box and return to the File Manager.

Using the Web Forms Mail Profile Manager

Using the Mail Profile Manager

If you set the PBMailManager global property to true on the Configuration tab for a Web Forms application, application users can open the Mail Profile Manager at any time from that application. Although you can choose to render the Mail Manager icon at design time, the IIS server administrator can change your selection after deployment by modifying the PBMailManager global property in the application's *Web.Config* file.

When you set the PBMailManager property to true, the Mail Manager icon displays in every window of your application. If you do not want the Mail Manager icon to display on a specific window in your application, set the HasMailManager property for that window to false.

You can also code an application event to open the Mail Profile Manager by calling the `OpenMailManager` function.

For information on the `HasMailManager` property, see `HasMailManager` on page 82. For information on the `OpenMailManager` function, see `OpenMailManager` on page 89.

The Mail Manager icon displays in the upper right corner of the Web Forms page, just to the left of the File Manager icon, when that icon is also rendered. The Mail Profile Manager opens in the current browser window after a user clicks the Mail Manager icon.

Automatic display of the Mail Profile Manager

The Mail Profile Manager displays automatically if a user triggers a `MailSend` call from a Web Forms application before a mail profile has been defined or if a default mail profile has not been set.

Figure 5-6 shows the Mail Profile Manager for a Web Forms application.

Figure 5-6: Mail Profile Manager for Web Forms applications

The screenshot shows a dialog box titled "Mail Profile Manager". At the top, there is a "Profile Name" dropdown menu. Below it is a checked checkbox labeled "Set as default mail profile". The dialog is divided into two main sections: "User Profile" and "Outgoing Mail". The "User Profile" section contains two text input fields: "Name" and "E-mail address". The "Outgoing Mail" section contains two text input fields: "Server Address" and "Port" (with the value "25" entered). Below the "Outgoing Mail" section is a sub-section titled "Requires authentication" with an unchecked checkbox. This sub-section contains two text input fields: "UserID" and "Password". At the bottom of the dialog, there are three buttons: "Create / Update", "Delete", and "Close".

The Mail Profile Manager is divided into sections for user profile information and outgoing mail parameters. Information that the application user enters in the Mail Profile Manager can be saved in a profile that is available to the Web Forms application.

Table 5-1 lists and describes the fields in the Web Forms Mail Profile Manager.

Table 5-1: Mail Profile Manager fields

Section	Field	Description
—	Profile Name	Name of the mail profile.
	Set as Default Mail Profile	Select to make the current mail profile the default profile for a Web Forms application.
User Profile	Name	Display name for the user.
	E-mail address	E-mail address the Web Forms user wants to use.
Outgoing Mail	Server address	Address for the outgoing mail server, such as <code>smtp.sybase.com</code> .
	Port	The default outgoing mail port is 25.
	Requires authentication	Select this check box if the outgoing mail server requires authentication.
	User ID	Alias used to log in to the e-mail server.
	Password	Password for the user ID. The password a user enters is encoded using an MD5 algorithm.

Creating, updating, and deleting mail profiles

Users can always enter new mail profile names in the Profile Name drop-down list. After entering all the Mail Profile Manager fields for a given profile, the application user must click Create/Update to save the entries to a profile file in the `applicationName_root\Mail\session\sessionID` virtual file directory. The profile is saved in an XML file with an encoded version of the user password. Unless the user has logged in as a permanent user, all mail profiles are deleted after the user terminates an application session.

Permanent user accounts

If the application user is logged in as a permanent user, the XML mail profile file is saved in the `applicationName_root\Mail\user\userName` directory.

For information about the permanent user functionality, see Chapter 4, “User Management and Registry Operations in Web Forms.”

An application user can display an existing mail profile by selecting it in the Profile Name drop-down list. The user can then edit the fields of the selected profile and save those changes by clicking Create/Update, or can remove the profile by clicking Delete. The Delete button is enabled only after an existing mail profile is selected in the Profile Name drop-down list.

Modifications required for Web Forms applications

Before you issue a MailSend call, you must create a MailSession object. This requirement is the same in Web Forms and standard client-server applications. However, in standard applications, you must also issue MailLogon and MailLogoff calls for the MailSession object. This is not necessary for Web Forms applications, and these calls are ignored by the PowerBuilder to .NET compiler if you include them in a Web Forms application.

For a standard PowerBuilder client-server application, you can use a MailSend call without arguments to open a new message window in the client's default mail application. Because you cannot do this from a Web Forms application, you can use a MailSend call only if you include a *mailmessage* argument.

You populate a MailMessage object the same way for a Web Forms application as you do for a standard client-server application. The properties of the MailMessage object include the text, subject line, and recipient information for the message that the application user sends. Some of the properties of a MailMessage object are ignored in a Web Forms application. For a list of unsupported properties, see Table 8-4 on page 98.

Unsupported mail functions

The MailAddress, MailDeleteMessage, MailGetMessages, MailHandle, MailLogon, MailLogoff, MailReadMessage, MailRecipientDetails, MailResolveRecipient, and MailSaveMessage functions are not supported in Web Forms applications. Although these functions are ignored by the PowerBuilder to .NET compiler, they do not produce application errors and do not interfere with supported mail functionality in Web Forms applications.

Using the Web Forms Theme Manager

The Theme Manager allows users to change the appearance of controls in Web Forms applications. By default, the controls display with themes that are consistent with the operating system of the client browser. However, the Theme Manager allows users to change the controls to display with Windows XP or Windows Classic themes regardless of the underlying operating system. You can also change the default themes for all browsers by modifying the PBDefaultTheme global property at design time.

For a description of global properties, see “Global Web configuration properties” on page 74.

Another global property, `PBThemeManager`, determines whether the Theme Manager is available to users at runtime. When you set the `PBThemeManager` property to true, the Theme Manager icon normally displays in every window of your Web Forms application. Users can open the Theme Manager at any time by clicking the Theme Manager icon. You can also code an application event to open the Theme Manager by calling the `OpenThemeManager` function.

Although you can choose to render the Theme Manager icon at design time, if you do not want it to display on a specific window in your application, you can set the `HasThemeManager` property for that window to false.

For information on the `HasThemeManager` property, see `HasThemeManager` on page 83. For information on the `OpenThemeManager` function, see `OpenThemeManager` on page 90.

The Theme Manager icon displays in the upper right corner of Web Forms, just to the left of the Mail Profile Manager icon when that icon is also rendered. The Theme Manager opens in the current browser window when a user clicks the Theme Manager icon.

About this chapter

This chapter describes global properties and built-in control properties that are available to .NET Web Forms applications.

Contents

Topic	Page
About Web Forms properties	73
Global Web configuration properties	74
Creating custom global properties	79
AutoPostBack	80
Embedded	80
HasFileManager	81
HasMailManager	82
HasPrintManager	82
HasThemeManager	83

About Web Forms properties

In addition to PowerScript properties that are converted to .NET properties and JavaScript attributes, a .NET Web Forms application has global properties that you can set at design time on the Configuration tab of the Project painter or after deployment in the generated *Web.Config* file for your application.

Several built-in control properties are also available for Web Forms applications that are not valid for other types of PowerBuilder targets. You must surround the calls to the built-in control properties in a conditional compilation block for .NET Web Forms. You can set these properties to reduce postbacks, embed hyperlinked Web pages, or remove the display of file, mail, print, and theme manager icons from specific windows when a global display property is set. You cannot set global properties in script.

For information on global properties, see “Global Web configuration properties” next.

Global Web configuration properties

Table 6-1 lists global properties of a PowerBuilder .NET Web Forms application that you can set at design time on the Configuration tab of the Project painter before you deploy the application. The default global properties and their values display only when the “System defined configuration settings” radio button is selected. This radio button is selected by default.

For information on modifying global properties after you deploy a Web Forms application, see “Viewing and modifying global properties in IIS Manager” on page 8.

Additional global properties are described in Table 6-2. They are included in the *Web.Config* file that is generated in the main application directory under the IIS virtual root directory.

You can also create custom global properties. For information about custom global properties, see “Creating custom global properties” next.

When you select a global property from the Key and Value list and click the Edit button, the Set Configuration Value dialog box displays. You can use this dialog box to change the values of system or custom global properties.

Table 6-1: Global properties on the Project painter Configuration tab

Property	Default value	Description
PBAutoTriggerMenuSelectedEvents	False	Indicates whether to trigger the menu Selected event for all menu items before Web Forms are rendered in the browser. The Selected event can be handled on the server only for simple tasks related to the appearance of the menu items. The Selected event is always disabled on the client side to prevent unnecessary postbacks when a menu item is highlighted.
PBCachedAndSharedDDWs	—	A comma-delimited set of names for DataWindow objects that you want to use in DropDownDataWindow edit style controls for sharing across application sessions. For more information, see “Sharing data across sessions” on page 26.
PBCachedAndSharedDWs	—	A comma-delimited set of names for DataWindow objects that you want to share across application sessions. For more information, see “Sharing data across sessions” on page 26.
PBCommandParm	—	Sets command line parameters for your application. Users can override the default by setting this property in a URL.

Property	Default value	Description
PBCultureSource	Server	Enumeration that specifies the source of regional settings for data formats. Values are Server or Client. For more information about regional settings, see “Use regional formats based on client or server settings” on page 205
PBDataWindowEnableDD DW	False	Indicates whether to render a DropDownDataWindow (DDDW) or a DropDownList control for a column using the DDDW edit style. Values are true to render the drop-down object as a DDDW, or false to render it as a list box. The value you set applies to all DDDW objects in the application, although if you set this value to true, you can still render a specific DDDW object as a list box by setting its HTMLGen.GenerateDDDWFrames property (the “Generate DDDW Frames” field on the Web Generation page of the DataWindow painter Properties view) to false.
PBDataWindowGoToButt onText	Go	Sets the label for a navigation bar button that takes a user to a designated DataWindow page.
PBDataWindowGoToDesc ription	Go To:	Sets the label for the navigation control that takes a user to a designated DataWindow page.
PBDataWindowNavigatio nBarPosition	PBDWBottom	Sets the position where the page navigation controls display. Values are PBDWBottom for display at the bottom of the DataWindow, PBDWTop for display at the top of the DataWindow, or PBDWTopAndBottom for display at the top and bottom of the DataWindow.
PBDataWindowPageCount PerGroup	10	Sets a limit to the number of pages that display for the Numeric or NumericWithQuickGo style navigation bars.
PBDataWindowPageNavig atorType	NextPrev	Values are NextPrev, Numeric, QuickGo, NextPrevWithQuickGo, or NumericWithQuickGo. For more information, see “Take advantage of global configuration properties” on page 207.
PBDataWindowQuickGoP ageNavigatorType	DropDownList	Sets the type of control to use for the Quick Go navigation bar. Values are DropDownList or Edit.
PBDataWindowRowsPerP age	20	Sets the number of DataWindow rows to display in Web Forms when the HTMLGen.PageSize property of the DataWindow object is not set. HTMLGen.PageSize and PBDataWindowRowsPerPage have no effect on DataWindow objects with the Label presentation style. Composite and Crosstab presentation styles do not support pagination.
PBDataWindowScriptCall backDDDW	False	Set this to true to load a DropDownDataWindow on demand using ASP.NET script callbacks when PBDataWindowEnableDDDW is also set to true.

Property	Default value	Description
PBDataWindowStatusInfo Format	Page {C} of {T}	Sets the text to display for the DataWindow page count {C} and the total number of pages {T}. The other variables you can use are placeholders for the starting {S} and ending {E} page of a group range that you set in the PBDataWindowPageCountPerGroup global property.
PBDBFetchBuffers	1	The default value causes values to be entered in the database trace log for each fetch request when tracing is enabled. Set to 0 to disable tracing on each fetch request.
PBDBLogFileName	c:\dbtrace.log	The name of the database log file when tracing is enabled. The log file is saved under the application root directory in the virtual file system on the IIS server. You can use PowerBuilder file functions to open and read the log file.
PBDBShowBindings	1	The default value causes metadata from the database result set columns to be entered in the database trace log when tracing is enabled. Set to 0 to disable these entries.
PBDBShowDBINames	0	If you set this value to 1, the original database interface command names are included in a database trace log file when tracing is enabled. By default, these names are not included in the log file.
PBDBSqlTraceFile	c:\pbtrsql.log	The name of the database log file when SQL command tracing is enabled. The log file is saved under the application root directory in the virtual file system on the IIS server. You can use PowerBuilder file functions to open and read the log file.
PBDBSumTiming	1	The default value causes the cumulative total of timings since the database connection began to be entered in the database trace log file when tracing is enabled. Set to 0 to disable these entries.
PBDBTiming	1	The default value causes the time required to process database interface commands to be entered in the database trace log file when tracing is enabled. Set to 0 to disable these entries.
PBDefaultTheme	Auto	The default theme selection causes controls in the Web Forms application to display with Windows Classic themes for Windows 2000 and 2003 operating systems and Windows XP themes for all other operating systems. If you want the controls to display with Windows XP themes, regardless of the client operating system, you can select "XP". If you want the controls to display in all client browsers with a Windows Classic appearance, you can select "Classic".

Property	Default value	Description
PBDeleteTempFileInterval	600 (minutes)	Sets the number of minutes before temporary files created by composite DataWindows are deleted. A value of 0 prevents the temporary files from being deleted.
PBDenyDownloadFolders	c:\~pl_	A comma-delimited string of directory names. Application users are not able to use the Web Forms File Manager to download files in any of the directories listed in this string.
PBEventLogID	1100	The event ID if exceptions are logged to the EventLog.
PBFileManager	False	Set to true if you want to render the File Manager icon in a Web Forms application.
PBFormExitMessage	"If there is any unsaved data, it will be lost."	Use to set custom message when users exit the current form. The custom message is sandwiched between two default sentences in the same message box: "Are you sure you want to navigate away from this page?" and "Press OK to continue, or Cancel to stay on the current page."
PBIdleInterval	0 (seconds)	Factor that adjusts the interval for the Idle event in a Web Forms application. The actual interval is determined by the application idle interval multiplied by the PBIdleInterval value. A value of 0 prevents the Idle event from being triggered.
PBJVMLogFileName	<i>vm.out</i>	Name of the file that logs information about the JVM for applications using a JDBC connection. By default, this file is saved to the <i>applicationName_root</i> \Log directory under the virtual root directory of the Web server.
PBLibDir	c:\~pl_	The directory on the server where dynamic libraries are generated.
PBMailManager	False	Set to true if you want to render the Mail Manager icon in a Web Forms application.
PBMailTimeout	1200000 (milliseconds)	Time in milliseconds before an SMTP session expires. The default value is equivalent to 20 minutes, which is also the HTTP session timeout period. It should be set higher if the mail includes file or data attachments. The SMTP session also expires after an e-mail is sent from the Web Forms application.
PBMaxSession	0	Sets the maximum number of Web Forms sessions that can be open at the same time. The default value of 0 places no limitation on the number of sessions that can be open simultaneously.
PBShowDenyDownloadFolders	False	Set to true to allow application users to see the server-side folders to which you restrict download access by listing them in the PBDenyDownloadFolders global property. By default, these folders are not visible in the File Manager .

Property	Default value	Description
PBShowFormExitMessage	True	Set to false to prevent a message box from displaying when application users exit the current form. If you set this to true, you can add a custom message to the message box by setting the PBFormExitMessage global property.
PBTempDir	c:\temp	A temporary directory under the virtual file root on the server.
PBThemeManager	False	Set to true if you want to render the Theme Manager icon in a Web Forms application.
PBTimerInterval	0 (seconds)	Factor that adjusts the interval for the window Timer event in a Web Forms application. The actual interval is determined by the window's Timer interval multiplied by the PBTimerInterval value. A value of 0 prevents the Timer event from being triggered.
PBTrace	Enabled	Indicates whether to log exceptions thrown by the Web Forms application. Values are Enabled or Disabled.
PBTraceFileName	<i>PBTrace.log</i>	Name of the file that logs exceptions thrown by the Web Forms application. By default, this file is saved to the <i>applicationName_root\Log</i> directory under the virtual root directory on the Web Forms server.
PBTraceLevel	Critical	By default, the .NET runtime logs critical exceptions only. However, if you set this property to SystemFunction, the .NET runtime logs all exceptions caught by system functions.
PBTraceTarget	File	Defines where to log exceptions thrown by the Web Forms application. Values are File or EventLog.
PBWebFileProcessMode	Share	Share mode maintains files in a read-only state when a write file operation is not explicitly coded. If an application requires multiple file operations, you might want to change this property setting to Copy mode. For more information on Share and Copy mode, see "Using the Web Forms File Manager" on page 60.
PBWindowDefaultHeight	600 (pixels)	Specifies the default height for MDI, MDIHelp, and main type windows when they are opened as maximized for the first time.
PBWindowDefaultWidth	1003 (pixels)	Specifies the default width for MDI, MDIHelp, and main type windows when they are opened as maximized for the first time.

Property	Default value	Description
PBYieldTimeout	10000 (milliseconds)	Time in milliseconds before the Yield function causes a postback to the server. Yield calls are ignored if you set this value to 0. When you set this value to 0, however, you must make sure your application does not call Yield inside a loop, as in the following example: <pre>do while flag yield() loop</pre>

Table 6-2 displays global properties that you cannot set on the Configuration tab. However, you can change these properties in the *Web.Config* file after deployment. Also, the selection that you make for Web Application Name on the General tab affects default values generated for these global properties.

Table 6-2: Additional global properties for a Web Forms application

Property	Default value	Description
FileFolder	<i>WebAppDir</i> .\appName_root\file	Base directory for the virtual file manager. It contains the File\Common directory structure and files that mirror paths for the application resource files on the development computer. If you switch to Copy mode, a <i>sessionID</i> directory is created under the File\Session directory that mirrors the File\Common directory structure and file contents.
MailFolder	<i>WebAppDir</i> .\appName_root\mail	Base directory for the mail manager.
PrintFolder	<i>WebAppDir</i> .\appName_root\print	Base directory for files that your application prints in PDF format.
LogFolder	<i>WebAppDir</i> .\appName_root\log	Folder that contains the <i>PBTrace.log</i> file.

Creating custom global properties

You can create custom global properties for a Web Forms project after you select the “Custom defined configuration settings” radio button on the Configuration tab page of the Project painter. This selection enables the Add button. Clicking Add causes the Add User Defined Configuration Setting dialog box to display. You use this dialog box to add a custom global property and a value for that property.

You cannot use a system global property name as the name for a custom global property. When you select a custom global property in the Key and Value list box on the Configuration tab page of the Project painter, the Edit and Delete buttons become enabled. You click the Edit button to change the value of a custom global property. You click Delete to remove a custom global property and its value.

AutoPostBack

Applies to	CheckBox and RadioButton controls
Description	You can reduce postbacks and improve performance by setting the AutoPostBack property for certain controls to false. When you set a control's AutoPostBack property to false, all events related to that control are triggered only in the processing of the next postback caused by another control in the Web Forms application.
Usage	<p>In scripts</p> <p>You must surround the AutoPostBack property in a conditional compilation code block for Web Forms applications:</p> <pre>#IF DEFINED PBWEBFORM THEN cbx_1.AutoPostBack = false #END IF</pre>

Embedded

Applies to	StaticHyperLink controls
Description	When you set the Embedded property to true for a StaticHyperLink control, the IFRAME element is used to embed the Web page that you defined in the control's URL property. The Web page displays inline on the Web Forms page.

Usage**In scripts**

The following code sets the Embedded property to “true”. You must surround the Embedded property in a conditional compilation code block for Web Forms applications:

```
#IF DEFINED PBDOTNET THEN
    static_hyperlink.Embedded = true
#END IF
```

HasFileManager

Applies to

Window controls

Description

This property is valid for .NET Web Forms applications only when the PBFileManager global property is set to true. The global property allows the File Manager icon to display on all window forms in your Web Forms applications. The File Manager icon gives users access to a file manager on the Web Forms server.

By default, the HasFileManager property is set to true and the PBFileManager is set to false. When you change the PBFileManager global property to true, all window forms display the File Manager icon unless you set the HasFileManager for a particular window to false.

Icon display in MDI Web Forms applications

In MDI applications, manager icons display on the frame window. To hide the File Manager icon when the PBFileManager global property is set to true, you must set the HasFileManager property of both the frame and the active sheet to false.

Usage**In scripts**

You must surround the HasFileManager property in a conditional compilation code block for Web Forms applications:

```
#if defined PBWEBFORM then
    w_mywindow.HasFileManager = false
#end if
```

HasMailManager

Applies to Window controls

Description This property is valid for .NET Web Forms applications only when the PBMailManager global property is set to true. The global property allows the Mail Profile Manager icon to display on all window forms in your Web Forms applications. The Mail Profile Manager icon gives users access to a mail profile manager on the Web Forms server.

By default, the HasMailManager property is set to true and the PBMailManager is set to false. When you change the PBMailManager global property to true, all window forms display the Mail Profile Manager icon unless you set the HasMailManager for a particular window to false.

Icon display in MDI Web Forms applications

In MDI applications, manager icons display on the frame window. To hide the Mail Profile Manager icon when the PBMailManager global property is set to true, you must set the HasMailManager property of both the frame and the active sheet to false.

Usage

In scripts

You must surround the HasMailManager property in a conditional compilation code block for Web Forms applications:

```
#if defined PBWEBFORM then
    w_mywindow.HasMailManager = false
#end if
```

HasPrintManager

Applies to Window controls

Description This property is valid for .NET Web Forms applications only when the Print Manager is activated. You activate the Print Manager by calling a supported print method.

When activated, the Print Manager icon displays on all window forms in your Web Forms applications. The Print Manager icon gives users access to a print manager for files on the Web Forms server.

Icon display in MDI Web Forms applications

In MDI applications, manager icons display on the frame window. To hide the Print Manager icon after it is activated, you must set the HasPrintManager property of both the frame and the active sheet to false.

By default, the HasPrintManager property is set to true. When you activate the Print Manager, all window forms display the Print Manager icon unless you set the HasPrintManager for a particular window to false.

For information on activating the Print Manager, see “Using the Web Forms Print Manager” on page 55.

Usage

In scripts

You must surround the HasPrintManager property in a conditional compilation code block for Web Forms applications:

```
#if defined PBWEBFORM then
    w_mywindow.HasPrintManager = false
#end if
```

HasThemeManager

Applies to

Window controls

Description

This property is valid for .NET Web Forms applications only when the PBThemeManager global property is set to true. The global property allows the Theme Manager icon to display on all window forms in your Web Forms applications. The Theme Manager icon allows users to change the display of controls in your application.

By default, the HasThemeManager property is set to true and the PBThemeManager is set to false. When you change the PBThemeManager global property to true, all window forms display the Theme Manager icon unless you set the HasThemeManager for a particular window to false.

Icon display in MDI Web Forms applications

In MDI applications, manager icons display on the frame window. To hide the Theme Manager icon when the PBThemeManager global property is set to true, you must set the HasThemeManager property of both the frame and the active sheet to false.

Usage

In scripts

You must surround the `HasThemeManager` property in a conditional compilation code block for Web Forms applications:

```
#if defined PBWEBFORM then
    w_mywindow.HasThemeManager = false
#end if
```

About this chapter

This chapter describes system functions that are restricted to .NET Web Forms applications. You cannot use them in other types of PowerBuilder applications.

Contents

Topic	Page
About system functions for Web Forms applications	85
DownloadFile	86
GetConfigSettings	87
GetDownloadFileURL	88
OpenFileManager	89
OpenMailManager	89
OpenPrintManager	90
OpenThemeManager	90
UploadFiles	90

About system functions for Web Forms applications

PowerBuilder provides system functions that are specific for .NET Web Forms applications. These functions allow you to open the various managers for Web Forms applications, to obtain global configuration settings, to download files for viewing or printing by the application user, or to upload files to the Web server.

You must surround calls to these system functions in a conditional compilation block for .NET Web Forms. These functions cannot be used with standard PowerBuilder client-server applications.

Functionality for downloading and uploading files is also available from the File Manager. The Print Manager allows application users to view files printed to the Web server in PDF format. You can enable the managers through global properties or by calling Web Forms system functions.

For information on using the managers, see Chapter 5, “Print, File, Mail Profile, and Theme Managers.”

DownloadFile

Description Downloads a file from the Web server to a client machine.

Syntax `void DownloadFile (string serverFile , boolean open)`

Argument	Description
<i>serverFile</i>	A string containing the name of the file on the application’s virtual file path on the Web server.
<i>open</i>	A boolean that determines whether to access the file in open mode or download mode. Values are: <ul style="list-style-type: none">• true Display the file directly in a browser window (open mode).• false Display a dialog box that lets the user open the file, save the file, or cancel the download operation (download mode).

Return value None

Usage Some types of files cannot be displayed directly in a browser window. For these types of files, the *open* argument is disregarded. Instead, the File Download dialog box displays as if you set the *open* argument to false, but the dialog box provides no option to open the file directly. In this case, users can only save the file to disk or cancel the download operation.

If the file you indicate in the *serverFile* argument is not present on the server, application users do not see an error message. You can use the FileExists PowerShell function to make sure the file exists in the server directory before you call DownloadFile.

Examples The following example opens the file *aaa.txt* in download mode:

```
#if defined PBWEBFORM then
    DownloadFile("c:\aaa.txt", false)
#end if
```

The download mode causes the File Download dialog box to display, giving the user the choice of opening the file, saving the file, or cancelling the operation. The File Download dialog box displays the file name, the file type, the number of bytes in the file, and the name of the server that hosts the file.

The following code opens a dialog box that allows users to select a directory and download multiple files from the same directory:

```

string docpath, docname[]
boolean lb_open
integer i, li_cnt, li_rtn, li_filename

lb_open = true //or false
li_rtn = GetFileOpenName("Select File", docpath, &
+ docname[], "DOC", &
+ "Text Files (*.TXT),*.TXT," &
+ "Doc Files (*.DOC),*.DOC," &
+ "All Files (*.*), *.*", &
"C:\Program Files\Sybase", 18)
IF li_rtn < 1 THEN return
li_cnt = Upperbound(docname)
// if only one file is picked, docpath contains the
// path and file name
if li_cnt = 1 then
mle_1.text = string(docpath)
#if defined PBWEBFORM then
DownloadFile(string(docpath), lb_open)
#end if
else
// if multiple files are picked, docpath contains
// the path only - concatenate docpath and docname
for i=1 to li_cnt
string s
s = string(docpath) + "\" + (string(docname[i]))
#if defined PBWEBFORM then
DownloadFile(s, lb_open)
#end if
mle_1.text += s + "~r~n"
next
end if

```

See also [GetDownloadFileURL](#)

GetConfigSettings

Description	Returns the value of a global configuration property.
Syntax	string GetConfigSettings (string key)

Argument	Description
<i>key</i>	A string for the name of a global property in the <i>Web.Config</i> file.

Return value String. Returns the value of the global property passed in the *key* parameter.

Examples The following code returns “N/A” for not applicable if the global property “myKey” is not found:

```
string v, k
k = "myKey"

#if defined PBWEBFORM then
    v = GetConfigSetting(k)
#else
    v = "N/A"
#endif if
```

See also DownloadFile

GetDownloadFileURL

Description Returns the URL for a file on the Web server.

Syntax string **GetDownloadFileURL** (string *serverFile* , boolean *open*)

Argument	Description
<i>serverFile</i>	A string containing the name of the file on the application’s virtual file path on the Web server.
<i>open</i>	A boolean that determines whether to access the file in open mode or download mode. Values are: <ul style="list-style-type: none"> • true Display the file directly in a browser window (open mode). • false Display a dialog box that lets the user open the file, save the file, or cancel the download operation (download mode).

Return value String. Returns the URL of the file in ASCII format.

Usage The *open* argument applies only if a Web Forms application user copies the returned URL in the current browser Address box or if you set a hyperlink in the current browser to the returned URL address.

Examples The following code places the URL for a text file in a MultiLineEdit box and includes it as a hyperlink in a StaticHyperLink control:

```
#if defined PBWEBFORM then
    string s
    s = GetDownloadFileUrl("c:\aaa.txt", false)
    mle_1.text += "~r~n" + s
    shl_1.url = s //shl_1: static hyperlink
#end if
```

See also DownloadFile

OpenFileManager

Description Opens the Web Forms File Manager.

Syntax void **OpenFileManager** ()

Return value None

Usage For information on using the File Manager, see “Using the Web Forms File Manager” on page 60.

Examples You can use this code to open the Web Forms File Manager:

```
#if defined PBWEBFORM then
    OpenFileManager()
#end if
```

OpenMailManager

Description Opens the Web Forms Mail Profile Manager.

Syntax void **OpenMailManager** ()

Return value None

Usage For information on using the Mail Profile Manager, see “Using the Web Forms Mail Profile Manager” on page 67.

Examples You can use this code to open the Web Forms Mail Profile Manager:

```
#if defined PBWEBFORM then
    OpenMailManager()
#end if
```

OpenPrintManager

Description	Opens the Web Forms Print Manager.
Syntax	void OpenPrintManager ()
Return value	None
Usage	For information on using the Print Manager, see “Using the Web Forms Print Manager” on page 55.
Examples	You can use this code to open the Web Forms Print Manager:

```
#if defined PBWEBFORM then
    OpenPrintManager()
#end if
```

OpenThemeManager

Description	Opens the Web Forms Theme Manager.
Syntax	void OpenThemeManager ()
Return value	None
Usage	For information on using the Theme Manager, see “Using the Web Forms Theme Manager” on page 70.
Examples	You can use this code to open the Web Forms Theme Manager:

```
#if defined PBWEBFORM then
    OpenThemeManager()
#end if
```

UploadFiles

Description	Opens the Upload Files dialog box that enables an application user to upload files from the local machine to the Web server.
Syntax	void UploadFiles (string <i>serverFolder</i> , long <i>bgColor</i> , int <i>fileNum</i> , boolean <i>showServerFolder</i> , string <i>description</i> , string <i>allowExts</i> {, string <i>callbackFunctionName</i> }{, PowerObject <i>po</i> })

Argument	Description
<i>serverFolder</i>	The folder on the server that contains the file or files you want to copy to the client.
<i>bgColor</i>	A long for the background color of the Upload Files dialog box.
<i>fileNum</i>	An integer for the number of text boxes to display in the Upload Files dialog boxes. Application users can upload as many files as there are text boxes in a single upload operation.
<i>showServerFolder</i>	A boolean specifying whether to display the server folder name in the Upload Files dialog box. Values are: <ul style="list-style-type: none"> • true Display the server folder name. • false Do not display the server folder name.
<i>description</i>	Text that you want to display near the top of the Upload Files dialog box. You can use an empty string if you do not want to display additional text in this dialog box.
<i>allowExts</i>	A string that lets you limit the files a user can upload to files with the extensions you list. If you set this argument to an empty string, files with any file extension can be uploaded. If you list multiple extensions, you must separate each extension with a semicolon. You must include the “.” (dot) in the extensions you list.
<i>callbackFunctionName</i> (optional)	The callback function that lets you know whether the file is correctly uploaded to the Web server.
<i>po</i> (optional)	The name of a PowerBuilder object that has the callback function set in the <i>callbackFunctionName</i> argument.

Return value

None.

Usage

You can use the `UploadFiles` function in conjunction with a private callback function that you create for a PowerBuilder object.

The callback function should return an integer and take a string array for its only argument. The callback function script should include an iteration to fill up the string array with the names of files selected by the application user to upload to the server. For example, the following code can be added to a callback function “`myuploadfiles_callback`” with an `upfiles[]` string array argument:

```
int i
for i = 1 to upperbound(up_files)
```

```
        this.mle_1.text += "~r~n" + up_files[i]
    next
    return i
```

If the “myuploadfiles_callback” function is created on the window `w_main`, you can use this window name as the value of the `po` argument in your Upload Files call. If you create the “myuploadfiles_callback” function as a global function, you can use the UploadFiles callback syntax without the `po` argument.

If your application uses sequential UploadFiles calls in the same script, only the callback function in the last of the UploadFiles calls is valid. The other UploadFiles calls can still upload selected files to the Web server, but further processing of the names of the uploaded files does not occur, even when the syntax for these calls includes a callback function that codes for such processing.

If the last UploadFiles call in a script containing sequential UploadFiles calls does not use a callback function, no callback processing occurs.

Examples

The following example uploads the file to the application’s virtual root `d:\hhh` directory on the server, sets the color of the Upload Files dialog box to the background color of the `w_main` application window, limits the number of files to be uploaded in a single operation to 3, does not display the server directory name in the Upload Files dialog box but does display the “my description” text, and limits the types of files that can be uploaded to JPG and TXT files:

```
#if defined PBWEBFORM then
    UploadFiles("d:\hhh", w_main.BackColor, 3, false,
        "my description", ".jpg;.txt",
        "myuploadfiles_callback", w_main)
#end if
```

This example uses green as the background color for the Upload Files dialog box, limits the number of files to be uploaded in a single operation to 1, displays the server folder name in the Upload Files dialog box, and does not restrict the types of files a user can upload to the Web server:

```
#if defined PBWEBFORM then
    UploadFiles("c:\hhh", RGB(0, 255, 0), 1, true,
        "", "", "myuploadfiles_callback", w_main)
#end if
```

See also

DownloadFile
GetDownloadFileURL

Modified and Unsupported Features in Web Forms Projects

About this chapter

You can use most of the PowerScript methods and properties in PowerBuilder applications that you deploy with a .NET Web Forms project. This chapter lists features of PowerBuilder and PowerScript that are not supported in Web Forms targets.

Contents

Topic	Page
About unsupported features	93
Unsupported objects	95
Unsupported system functions	96
Restrictions on supported controls	98
Modified display of visual controls	109
Unsupported functions for controls in Web Forms	111
Unsupported events for controls in Web Forms	115
Unsupported properties for controls in Web Forms	117

About unsupported features

When you deploy a PowerBuilder application as a Web Forms application to an IIS server, PowerBuilder lists any unsupported features in the Output window. For the most part, unsupported features fail silently in the Web Forms application, but unexpected results can also occur. If an unsupported feature prevents the PowerBuilder to .NET compiler from compiling your application, the failure and its cause are noted in the Output window in PowerBuilder.

DataWindow support

Presentation styles Currently all DataWindow presentation styles are supported except RichText and OLE. All DataWindow dialog boxes (Specify Retrieval Arguments, Specify Retrieval Criteria, Import File, Save As, Print, Sort, Filter, and Crosstab) are supported.

DataWindow expressions Most of the built-in functions for DataWindow expressions are supported, but they do not include the Describe, LookupDisplay, Case, Page, PageAbs, ProfileInt, and ProfileString expression functions or the aggregate expression functions. User-defined expression functions are also not supported in Web Forms applications.

Client-side DataWindow expressions

DataWindow expressions that change UI properties are not supported on the client side. To work around this issue, you can trigger the Clicked or RowFocusChanged event to force a postback. DataWindow expressions are fully supported on the server side with the exception of expression functions noted above.

Controls in DataWindow controls The controls you can add to a DataWindow are not all supported in Web Forms applications. The Oval, RoundedRectangle, InkPicture, OLE Object, and OLE Database Blob controls are not supported in a Web Forms DataWindow. For a list of unsupported properties of controls that are supported in Web Forms DataWindow objects, see Table 8-5.

JavaScript keywords You cannot use JavaScript reserved words to name fields or bands in a DataWindow control that you deploy to the Web. The list of reserved words is available on the Sun Microsystems Web site at <http://docs.sun.com/source/816-6410-10/keywords.htm>.

DataWindow pagination The Web DataWindow control uses a simplified version of DataWindow pagination rules, and provides a choice of page navigation bars instead of scroll bars to support page navigation.

For information on pagination display in Web Forms DataWindow controls, see “Take advantage of global configuration properties” on page 207. For a description of changes to the visual display of DataWindow controls in Web Forms applications, see “Modified display of DataWindow objects and controls” on page 110.

Printing DataWindow objects Although the PrintDataWindow or PrintScreen print functions are not supported, users can save DataWindow objects and their data as PDF files, and can print the current Web Forms page using a browser’s print menu when those are available. (Browser menus are available only when the *default.aspx* page name is included in the URL used to start the Web Forms application.)

- Mail support** Although you can send e-mail from Web Forms applications, there is no support for receiving e-mail. When you call MailSend, you must supply a MailMessage argument. The MailSend syntax without a parameter is not supported.
- The MailSend function returns an enumerated value of type MailReturnCode. The following values of the MailReturnCode enumeration are not supported in Web Forms applications:
- | | |
|------------------------------|-----------------------------|
| MailReturnAccessDenied | MailReturnNoMessages |
| MailReturnDiskFull | MailReturnTextTooLarge |
| MailReturnInsufficientMemory | MailReturnTooManyFiles |
| MailReturnInvalidMessage | MailReturnTooManyRecipients |
| MailReturnMessageInUse | MailReturnTooManySessions |
- PBNI feature** You cannot use PBNI extensions in .NET Web Forms targets.
- Hot keys** Hot keys, shortcut keys, and accelerator keys are not supported in .NET Web Forms targets.
- Functions on .NET primitive types** You cannot call functions on .NET primitive types that map to PowerBuilder primitive types. See Table 13-3 for the list of datatype mappings from .NET to PowerBuilder.

Unsupported objects

The PowerScript objects listed in Table 8-1 cannot be used in applications deployed to ASP.NET.

Table 8-1: Unsupported PowerScript objects in Web Forms applications

Category	Objects
Data reproduction	Pipeline
EAServer integration	RemoteObject
Menu	MenuCascade

Category	Objects
OLE	OleStorage OleStream OmStorage OmStream
Profiling and tracing	ProfileCall ProfileClass ProfileLine ProfileRoutine Profiling TraceFile TraceTree TraceTreeNode and descendants TraceActivityNode and descendants
Timing	Timing
Tablet PC	InkEdit InkPicture

Using structures in inherited objects Using local structures in inherited objects can prevent deployment of a .NET project. To deploy the project, replace all local structures defined in inherited objects with global structures.

Unsupported system functions

Table 8-2 lists categories of system functions that are not supported or are deferred until a future release:

Table 8-2: Unsupported system functions by category

Category	Functions
Clipboard functions	Clipboard, also any object function that uses the clipboard, such as Copy, Paste, Clear, and so on
DDE functions	CloseChannel, ExecRemote, GetCommandDDE, GetCommandDDEOrigin, GetDataDDE, GetDataDDEOrigin, GetRemote, OpenChannel, RespondRemote, SetDataDDE, SetRemote, StartHotLink, StartServerDDE, StopHotLink, StopServerDDE
Debugging functions	DebugBreak
Garbage collection functions	GarbageCollect, GarbageCollectGetTimeLimit, GarbageCollectSetTimeLimit

Category	Functions
Help functions	ShowHelp, ShowPopupHelp
Input method functions	IMEGetCompositionText, IMEGetMode, IMESetMode
Mail functions	MailAddress, MailDeleteMessage, MailGetMessages, MailHandle, MailLogoff, MailLogon, MailReadMessage, MailRecipientDetails, MailResolveRecipient, MailSaveMessage
Messaging functions	Post, Send
Miscellaneous functions	DoScript, DraggedObject, Handle, Run, Restart
Print functions	PrintDataWindow, PrintScreen, PrintSend, PrintSetPrinter, PrintSetup, PrintSetupPrinter
Profiling and tracing functions	TraceBegin, TraceClose, TraceDisableActivity, TraceDump, TraceEnableActivity, TraceEnd, TraceError, TraceOpen, TraceUser

Partially supported system function

IsNull function The IsNull function is supported for simple datatypes only. It is not very useful for structure and class objects, since in .NET targets, uninitialized variables always return true for an IsNull call even when they are not explicitly set to null. However, you can use IsValid to test for valid instances of these object types. You can also use IsNull for class objects after they have been created.

Timer function The concept of “current window” does not exist in Web Forms applications. Therefore, you must use the optional PowerScript syntax with the window name parameter and the name of an active window as the parameter value. The Timer function fails when the active window does not exist.

Yield function Due to the thread-model design of Web Forms applications, you cannot use the Yield function and the Selected event of a menu object concurrently. Doing this causes a JavaScript error.

Registry functions System registry functions can read and write registry entries, keys, and values on the server side, but do not perform these operations on the server computer’s registry in the same way as they do on a client computer’s registry in a standard PowerBuilder application.

For more information on system registry functions, see “Using the registry functions” on page 53.

Restrictions on supported controls

Almost all PowerBuilder controls are supported in .NET Web Forms applications. However some of the methods and properties on supported controls do not work in Web Forms applications. Table 8-3 lists functions, events, and properties that are not supported on any control.

Table 8-3: Unsupported functions, events, and properties

Category	Unsupported feature
Control Functions	Clear (supported for EditMask controls) Cut, Copy, Paste CanUndo, Undo Drag Print (can be used for DataWindows and DataStores to print to PDF files) SetActionCode SetRedraw
Events	Drag and drop events GetFocus, LoseFocus events (supported when a call to the SetFocus function causes the focus change) Help event MouseMove event Other event
Properties	Accelerator AccessibleDescription AccessibleName AccessibleRole DragAuto DragIcon IMEMode

Table 8-4 lists the functions, events, and properties that are not supported on some individual objects or controls. Table 8-4 does not include the items listed in Table 8-3. The entry “No additional” in Table 8-4 indicates that all items except those listed in Table 8-3 are supported for that control.

Table 8-4: Additional unsupported functions, events, and properties by control

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
Animation (Playback behavior depends on Windows Media Player on client side.)	Play (supported, but parameters ignored) Seek	Click DoubleClick Help Start Stop	OriginalSize Transparent

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
ClassDefinition	No additional	No additional	LibraryName VariableList (supported, but the sequence of variables might differ in .NET applications)
DataStore	Same as DataWindow	Destructor Error ItemChanged PrintEnd PrintPage PrintStart	No additional
DataWindow control	AcceptText CopyRTF, PasteRTF Find, FindNext GenerateResultSet GetText ImportClipboard InsertDocument LineCount OLEActivate Position PrintCancel ReplaceText ResetInk SaveInk functions Scroll Selected functions SelectText functions SetActionCode SetDetailHeight SetFocus SetRedraw SetText ShowHeadFoot TextLine	EditChanged GetFocus LoseFocus PrintEnd PrintMarginChange PrintPage PrintStart ScrollHorizontal ScrollVertical (The Clicked event is not triggered on editable text columns that already have focus. For more information, see “Partially supported control events” on page 117.)	ControlMenu HSplitScroll Icon LiveScroll MaxBox MinBox Resizable RightToLeft Title TitleBar

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
DataWindow object (See “DataWindow support” on page 93 for a list of controls that are not supported in a DataWindow object. See Table 8-5 for unsupported properties of controls that you can place in a DataWindow object.)	Has no functions	Has no events	<i>Bandname.Height.Autosize</i> <i>Bandname.Pointer</i> <i>Bandname.Text</i> <i>Grid.ColumnMove</i> <i>Header.#.Suppress</i> <i>Help.Property</i> <i>HorizontalScrollProperty</i> <i>HideGrayLine</i> <i>Label.Ellipse_Property</i> <i>Label.Shape</i> (support rectangle shape only) <i>OLE.Client</i> <i>Pointer</i> <i>Print.Preview.Property</i> <i>Retrieve.AsNeeded</i> <i>RichText.Property</i> <i>Row.Resize</i> <i>Sparse</i> <i>Storage.Property</i> <i>Tree.Property</i> <i>VerticalScrollProperty</i> <i>Zoom</i>
DataWindowChild	Same as DataWindow	Has no events	No additional
DatePicker	GetCalendar	Clicked CloseUp DoubleClick DropDown UserString ValueChanged	AllowEdit DropDownRight ShowUpDown TodaySection WeekNumbers
DropDownListBox, DropDownPictureList Box	Position ReplaceText SelectedLength SelectedStart SelectedText SelectText	DoubleClick	AllowEdit AutoHScroll Limit RightToLeft ShowList

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
EditMask	CanUndo LineCount LineLength Position ReplaceText Scroll SelectedLength SelectedLine SelectedStart SelectedText SelectText TextLine Undo	No additional	AutoHScroll AutoSkip AutoVScroll DisplayData DropDownRight HideSelection IgnoreDefaultButton Increment Limit MinMax Spin TabStop UseCodeTable
Graph	Clipboard ImportClipboard ImportFile SaveAs SetFocus	No additional	FocusRectangle
HProgressBar	No additional	DoubleClicked	SmoothScroll
HScrollBar	No additional	RButtonDown	No additional
HTrackBar	SelectionRange	No additional	SliderSize TickFrequency TickMarks
ListBox	SetTop Top	DoubleClicked	DisableNoScroll ExtendedSelect RightToLeft TabStop

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
List View	Arrange EditLabel FindItem (partial support: see “Control functions with partial support” on page 115 for more information) GetItemAtPointer GetOrigin SetOverlayPicture	BeginDrag BeginLabelEdit BeginRightDrag DeleteAllItems EndLabelEdit ItemActivate Key RightClicked RightDoubleClicked Sort	AutoArrange ButtonHeader DeleteItems ExtendedSelect FixedLocations GridLines HeaderDragDrop HideSelection LabelWrap LayoutRTL OneClickActivate RightToLeft ShowHeader TrackSelect TwoClickActivate UnderlineCold UnderlineHot
ListViewItem	No additional	No additional	CutHighlighted DropHighlighted ItemX ItemY
MailFileDescription	No additional	No additional	FileType Position
MailMessage	No additional	No additional	ConversationID DateReceived MessageType MessageSent ReceiptRequested Unread
MailRecipient	No additional	No additional	EntryID
MailSession	MailDeleteMessage MailGetMessages MailHandle MailLogon MailLogoff MailReadMessage MailRecipientDetails MailResolveRecipient MailSaveMessage (For MailSend restrictions, see “Mail support” on page 95.)	No additional	MessageID SessionID

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
Menu	No additional	Selected (Can be supported for simple tasks that are run prior to the rendering of Web Forms in a client browser. For more information, see “Partially supported control events” on page 117.)	BitmapBackColor BitmapGradient MenuAnimation MenuBitmaps MenuImage MenuTitles MenuTitleText MergeOption MicroHelp TitleBackColor TitleGradient ToolbarAnimation ToolbarItemDown ToolbarItemDownName ToolbarItemSpace
MonthCalendar	GetDisplayRange	Clicked DoubleClicked	AutoSize MaxSelectCount ScrollRate TodaySection WeekNumbers
MultiLineEdit	LineCount LineLength Position Scroll SelectedLine SelectedStart TextLine	RButtonDown	AutoHScroll AutoVScroll HideSelection TabStop
Picture	No additional	No additional	FocusRectangle Map3DColors
PictureButton	No additional	No additional	Map3DColors
PictureHyperLink	No additional	No additional	FocusRectangle Map3DColors
PictureListBox	SetTop Top	DoubleClicked	DisableNoScroll ExtendedSelect RightToLeft TabStop
RadioButton	No additional	No additional	BorderStyle

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
RichTextEdit	CopyRTF DataSource Find FindNext GetAlignment GetParagraphSetting GetSpacing GetTextColor GetTextStyle InputField functions InsertDocument (supported for TXT format only) InsertPicture IsPreview LineCount LineLength PageCount PasteRTF Position Preview PrintEx ReplaceText SaveDocument (see “Partial support for SaveDocument function” on page 107) Scroll functions Selected functions SelectText functions Set functions (except SetFocus, which is supported) ShowHeadFoot TextLine	DoubleClicked FileExists InputFieldSelected Key Modified Mouse events PictureSelected RButtonUp	Accelerator BottomMargin ControlCharsVisible HeaderFooter HScrollbar InputField properties LeftMargin Modified PictureAsFrame PopMenu Resizable RightMargin RulerBar SelectedStartPos SelectedTextLength StatusBar TabBar TopMargin VScrollbar WordWrap
ScriptDefinition	No additional	No additional	AliasName ExternalUserFunction (not supported for system functions; supported for external functions only) LocalVariableList Source SystemFunction

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
SimpleTypeDefinition	No additional	No additional	LibraryName
SingleLineEdit	No additional	RButtonDown	AutoHScroll HideSelection
StaticHyperLink	No additional	No additional	FillPattern FocusRectangle RightToLeft
StaticText	No additional	No additional	FillPattern FocusRectangle RightToLeft
Tab	No additional	DoubleClicked RightDoubleClicked	Alignment (supported in TabsOnTop style when ShowPicture is set to false) FixedWidth (supported in TabsOnTop style, single-line mode) FocusOnButtonDown Multiline (supported in TabsOnTop style) Perpendicular (supported in single-line mode) RaggedRight (supported in TabsOnTop style; always true in TabsOnLeft style) TabPosition (Enum values supported for TabsOnTop and TabsOnLeft only)
TreeView	AddStatePicture DeleteStatePicture DeleteStatePictures EditLabel GetItemAtPointer SetDropHighlight SetFirstVisible SetLevelPictures SetOverlayPicture	BeginDrag BeginLabelEdit BeginRightDrag EndLabelEdit Key Notify RightDoubleClicked Sort	DeleteItems DisableDragDrop EditLabels FullRowSelect HideSelection LayoutRTL LinesAtRoot PictureHeight PictureWidth RightToLeft SingleExpand StatePictureHeight StatePictureWidth TrackSelect

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
TreeViewItem	No additional	No additional	Bold CutHighlighted DropHighlighted ExpandedOnce HasFocus OverlayPictureIndex Selected
TypeDefinition	No additional	No additional	LibraryName
UserObject	AddItem DeleteItem EventParmDouble EventParmString InsertItem	No additional	ColumnsPerPage LibraryName LinesPerPage Style TabBackColor UnitsPerColumn UnitsPerLine
VariableDefinition	No additional	No additional	OverridesAncestorValue Supported only for descriptions of instance variables: IsConstant ReadAccess WriteAccess Supported only for descriptions of instance variables and primitive-type properties, such as int, string, long, and so on: InitialValue
VProgressBar	No additional	DoubleClicked	SmoothScroll
VScrollBar	No additional	RButtonDown	No additional

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
VTrackBar	SelectionRange	No additional	SliderSize TickFrequency TickMarks
Window	DDE functions GetToolBar GetToolBarPos InputField functions SetMicroHelp SetToolBar SetToolBarPos	DDE events Deactivate DoubleClick Hide Key Mouse events SystemKey ToolBarMoved	Border, ClientEdge ColumnsPerPage ContextHelp HScrollbar KeyboardIcon LinesPerPage PaletteWindow Resizable RightToLeft TitleBar ToolBar properties (except ToolBarVisible) UnitsPerColumn UnitsPerLine VScrollbar Supported for child, popup, and response windows only: Center ControlMenu MaxBox MinBox

Partial support for SaveDocument function

The SaveDocument function for RichTextEdit controls is supported for the TXT format, but HTML tags are saved in the text file. SaveDocument can also save text and images to HTML and DOC file formats, however, it cannot correctly save Unicode characters in these file formats. SaveDocument does not support RTF or PDF formats in Web Forms applications.

Controls in
DataWindow objects

Table 8-5 lists the properties that are not supported in Web Forms applications for controls that you can place in a DataWindow object.

Table 8-5: Unsupported properties of controls in a DataWindow

Control in DataWindow	Unsupported properties
Button	<i>AccessibleProperty</i> , Font.Escapement, Font.Width, HideSnaked, Moveable, Pointer, Resizable, SlideLeft, SlideUp, VTextAlign
Column	<i>AccessibleProperty</i> , CheckBox.Scale, CheckBox.Other, ddlb.AllowEdit, ddlb.Limit, ddlb.ShowList, ddlb.Sorted, ddlb.UseAsBorder, dddw.AllowEdit, dddw.Lines, dddw.PercentWidth, dddw.ShowList, dddw.UseAsBorder, Edit.AutoHScroll, Edit.AutoVScroll, Edit.Case, Edit.HScrollBar, Edit.VScrollBar, EditMask.CodeTable, EditMask.DDCal_Property, EditMask.SpinProperty, Font.Escapement, Font.Width, Height.Autosize, HideSnaked, Ink.Property, InkEdit.Property, Moveable, Pointer, RadioButtons.Scale, Resizable, SlideLeft, SlideUp, UseEllipsis
Computed field	<i>AccessibleProperty</i> , Font.Escapement, Font.Width, Height.Autosize, HideSnaked, Moveable, Pointer, Resizable, SlideLeft, SlideUp
Graph	<i>AccessibleProperty</i> , HideSnaked, Moveable, Pointer, Resizable, SlideLeft, SlideUp
Group box	<i>AccessibleProperty</i> , Font.Escapement, Font.Width, Moveable, Pointer, Resizable, SlideLeft, SlideUp
Line (diagonal line is unsupported)	Moveable, Pen.Style, Pen.Width, Pointer, Resizable, SlideLeft, SlideUp
Picture	<i>AccessibleProperty</i> , HideSnaked, Invert, Moveable, Pointer, Resizable, SlideLeft, SlideUp
Rectangle	Brush.Hatch, Moveable, Pen.Style, Pen.Width, Pointer, Resizable, SlideLeft, SlideUp
Report	Border, Height.Autosize, Height, HideSnaked, Moveable, NewPage, Pointer, Resizable, SlideLeft, SlideUp, Trail_Footer
Text	<i>AccessibleProperty</i> , Font.Escapement, Font.Width, HideSnaked, Moveable, Pointer, Resizable, SlideLeft, SlideUp

Modified display of visual controls

Windows themes

By default, the rendering of visual controls in Web Forms applications uses themes consistent with the operating system of the client browser. However, by changing the value of the `PBDefaultTheme` global property, you can change the rendering of visual controls so that they display in the same way on all browsers, regardless of the underlying operating system.

If you select “XP” as the value for `PBDefaultTheme`, visual controls display with XP themes even when XP themes are not enabled on the client or Web server. If you select “Classic” as the `PBDefaultTheme` value, controls display with Windows Classic themes in all browsers.

You can let the application user change the control appearance by enabling the Theme Manager. The Theme Manager allows the end user to change the themes type to Windows Classic or Windows XP in a specific browser, however it does not let the user change the `PBDefaultTheme` value on the server.

For information about enabling the Theme Manager, see “Using the Web Forms Theme Manager” on page 70.

Visual properties and controls

Table 8-6 describes the behavior of visual properties and controls that differs in Web Forms applications from the behavior of the same properties or controls in a standard PowerBuilder environment. For a description of changes to the visual display of DataWindow controls in Web Forms applications, see “Modified display of DataWindow objects and controls” next.

Table 8-6: Modified display of supported visual properties and controls

Visual component or control	Behavior in Web Forms applications
Animation	When autoplay is set to false, the initial frame of the animation displays as a black area.
Border style: StyleBox!	The borders for RichTextEdit controls display as a white box frame around the outside of the control, with black lines along the top and left interior edges of the frame.
Border style: StyleLowered!	The borders for CheckBox, DatePicker, DropDownListBox, DropDownPictureListBox, EditMask, ListView, MonthCalendar, MultiLineEdit, PictureButton, SingleLineEdit, and TreeView controls display as a blue box (the default XP theme display) surrounding the control. Changing the color scheme does not alter the border color. RichTextEdit controls display with a thicker frame than in standard PowerBuilder applications.

Visual component or control	Behavior in Web Forms applications
Border style: StyleRaised!	The borders for GroupBox controls that use a raised border style are not as distinct as in standard PowerBuilder applications. RichTextEdit controls display with a thicker frame than in standard PowerBuilder applications.
Border style: StyleShadowBox!	For RichTextEdit controls, this style displays like the StyleBox! border style, except that the white-line box frame is slightly thicker.
Color Selection dialog box	Does not use a vertical track bar to change colors.
CommandButton	Text alignment is set to the left when the text length exceeds the control's width, not to the center of the button.
EditMask	You cannot use the Shift key to select text in the control.
Listview	Icon colors for the listview items appear inverted when selected.
SingleLineEdit	Password characters can display in a strange font. To get consistent behavior in all environments, use TrueType fonts only.
StaticText	Text is truncated to fit the size of the control, even if that is in the middle of a word.
Tab	If you change the X and Y positions of a user object on a Tab control when the MultiLine property is set to true and the tab positions are set to TabsOnTop, the user object can overlap the tab page tabs. If you need to change the position of the user object or if you want to place it so that it covers the entire tab page without overlapping the tabs, you must first set MultiLine to false.
Treeview	Bitmap pictures for the treeview items are displayed in their original sizes. Also, when you call SelectItem (0), a selected item does not lose focus. In Web Forms applications, at least one node must remain selected.
Window	MDI sheet windows display as tab pages instead of cascading sheets.

Modified display of DataWindow objects and controls

In Web Forms applications, DataWindow objects with the freeform presentation style can display part of a row when the height of all rows exceeds the height of the DataWindow control. For example, if one and a half rows can be displayed, the DataWindow displays one and a half rows. In standard PowerBuilder applications, only entire rows are displayed.

ScrollToRow The ScrollToRow method changes the row specified in the method argument to be the current row, but the specified row displays differently in standard PowerBuilder and Web Forms DataWindow controls. In Web Forms applications, when the ScrollToRow call causes the DataWindow to scroll up, the top of the specified row aligns with the top of the DataWindow control. When the ScrollToRow call causes the DataWindow to scroll down, the specified row displays in one of the following ways:

- If the row height is greater than the DataWindow control height, the top of the specified row aligns with the top of the DataWindow control
- If the row height is less than the DataWindow control height, the bottom of the specified row aligns with the bottom of the DataWindow control

For information on pagination display in Web Forms DataWindow controls, see “Take advantage of global configuration properties” on page 207.

Drop-down edit styles in DataWindow objects By default, DropDownDataWindow (DDDW) objects display as list boxes in Web Forms applications. When you open a response window or a message box in front of a DataWindow that has DDDW objects displayed as list boxes or that has columns with DropDownListBox (DDLB) edit styles, the DDDW objects and DDLB columns disappear until the response window or message box is closed.

The same temporary object and column disappearance occurs when an event such as Clicked, DropDown, ItemFocusChanged, or RowFocusChanged is handled. This is due to a limitation of the HTML SELECT element used to create a list box. You can prevent the disappearance of DDDW objects and DataWindow columns with the DDLB edit style by setting the PBDataWindowEnableDDDW global property to true. With this setting, DDLB column edit styles are automatically rendered as DDDW edit styles in Web Forms applications, and DDDW objects are not changed to list boxes.

For information on global properties, see “Global Web configuration properties” on page 74.

Unsupported functions for controls in Web Forms

Table 8-7 lists unsupported functions, the controls on which they are not supported, and any notes that apply to specific controls. If your application uses these functions, rework it to avoid their use.

Table 8-7: Unsupported functions by control in Web Forms projects

Function	Controls not supporting function
AcceptText	DataWindow
AddData	Graph (supported for all datatypes except string values)
AddItem	UserObject
AddStatePicture	TreeView
Arrange	ListView
CanUndo	All controls
Check	Menu (supported in all menu controls, but check mark appearance is not the same as in PowerBuilder applications)
Clear	Most controls (supported in EditMask controls)
Clipboard	Graph
CloseChannel	Window
Copy	All controls
CopyRTF	DataStore, DataWindow, RichTextEdit
Cut	All controls
DataSource	RichTextEdit
DeleteItem	UserObject
DeleteStatePicture	TreeView
DeleteStatePictures	TreeView
Drag	Most controls (supported in list box controls)
EditLabel	ListView, TreeView
EventParmDouble	UserObject
EventParmString	UserObject
ExecRemote	Window
Find	DataWindow, RichTextEdit
FindNext	DataWindow, RichTextEdit
GenerateResultSet	DataStore, DataWindow
GetAlignment	RichTextEdit
GetCalendar	DatePicker
GetCommandDDE	Window
GetCommandDDEOrigin	Window
GetContextService	Window (supported for ClassDefinition, ScriptDefinition, TypeDefinition, and VariableDefinition objects)
GetDataDDE	Window
GetDataDDEOrigin	Window
GetDisplayRange	MonthCalendar

Function	Controls not supporting function
GetNextSheet	Window (returns sheet instead of frame)
GetItemAtPointer	ListView, TreeView
GetOrigin	ListView
GetParagraphSetting	RichTextEdit
GetRemote	Window
GetSpacing	RichTextEdit
GetText	DataWindow
GetTextColor	RichTextEdit
GetTextStyle	RichTextEdit
GetToolbar	Window
GetToolbarPos	Window
ImportClipboard	DataWindow, Graph
ImportFile	Graph
InputField functions	RichTextEdit
InsertData	Graph (supported for all datatypes except string value.)
InsertDocument	DataWindow
InsertItem	UserObject
InsertPicture	RichTextEdit
IsPreview	RichTextEdit
LineCount	DataWindow, EditMask, MultiLineEdit, RichTextEdit
LineLength	DataWindow, EditMask, MultiLineEdit, RichTextEditt
OLEActivate	DataWindow
OpenChannel	Window
PageCount	RichTextEdit
Paste	All controls
PasteRTF	DataStore, DataWindow, RichTextEdit
Position	DataWindow, DropDownListBox, DropDownPictureListBox, EditMask, MultiLineEdit, RichTextEdit
Preview	RichTextEdit
Print	All controls (can be used for DataWindows and DataStores to print to PDF files)
PrintEx	RichTextEdit
ReplaceText	DataWindow, DropDownListBox, DropDownPictureListBox, EditMask, RichTextEdit
RespondRemote	Window
SaveAs	Graph

Function	Controls not supporting function
Scroll	DataWindow, EditMask, MultiLineEdit, RichTextEdit
Seek	Animation
SelectedColumn	RichTextEdit
SelectedLength	DataWindow, DropDownListBox, DropDownPictureListBox, EditMask, RichTextEdit
SelectedLine	DataWindow, EditMask, MultiLineEdit, RichTextEdit
SelectedStart	DataWindow, DropDownListBox, DropDownPictureListBox, EditMask, MultiLineEdit, RichTextEdit
SelectedText	DataWindow, DropDownListBox, DropDownPictureListBox, EditMask, RichTextEdit
SelectionRange	HTrackbar, VTrackbar
SelectText	DataWindow, DropDownListBox, DropDownPictureListBox, EditMask, RichTextEdit
SelectTextAll	RichTextEdit
SelectTextLine	RichTextEdit
SelectTextWord	RichTextEdit
SetActionCode	All controls
SetAlignment	RichTextEdit
SetDataDDE	Window
SetDetailHeight	DataWindow
SetDropHighLight	TreeView
SetFirstVisible	TreeView
SetFocus	DataWindow, Graph
SetLevelPictures	TreeView
SetMicroHelp	Window
SetOverlayPicture	Listview, TreeView
SetParagraphSetting	RichTextEdit
SetPosition	RichTextEdit
SetRedraw	All controls
SetRemote	Window
SetSpacing	RichTextEdit
SetText	DataWindow
SetTextColor	RichTextEdit
SetTextStyle	RichTextEdit
SetToolbar	Window
SetToolbarPos	Window

Function	Controls not supporting function
SetTop	ListBox, PictureListBox
ShowHeadFoot	DataWindow, RichTextEdit
StartHotLink	Window
StartServerDDE	Window
StopHotLink	Window
StopServerDDE	Window
TextLine	DataWindow, EditMask, MultiLineEdit, RichTextEdit
Top	ListBox, PictureListBox
Undo	All controls

Control functions with partial support

FindItem In Web Forms applications, the FindItem function is supported for all list box controls and the TreeView control. The syntax for finding an item by its label is also fully supported for the ListView control. However, the syntax for finding an item by its relative position in a ListView control is only partially supported. In Web Forms applications, the cuthighlighted and drophighlighted arguments are not supported and DirectionAll! is the only supported value for the direction argument.

Unsupported events for controls in Web Forms

Table 8-8 lists unsupported events, the controls on which they are not supported, and any notes that apply to specific controls. If your application uses these events, rework it to avoid their use.

Custom events

Custom events based on PowerBuilder Message (pbm) event IDs are not supported in Web Forms applications. However, you can call user-defined events without event IDs using the TriggerEvent and PostEvent functions.

Table 8-8: Unsupported events by control in Web Forms projects

Event	Controls
BeginDrag	All controls
BeginLabelEdit	ListView, TreeView
BeginRightDrag	All controls

Event	Controls
Clicked	DatePicker, MonthCalendar (supported for DataWindow, but not triggered on editable controls that already have focus)
CloseUp	DatePicker
Deactivate	Window
DeleteAllItems	ListView
DoubleClick	DatePicker, DropDownListBox, DropDownPictureListBox, HProgressBar, MonthCalendar, RichTextEdit, Tab, VProgressBar, Window (supported for other controls, but the Clicked event is not triggered on a double-click in a Picture or StaticText control)
DragDrop	All controls
DragEnter	All controls
DragLeave	All controls
DragWithin	All controls
DropDown	DatePicker
EditChanged	DataWindow
EndLabelEdit	ListView, TreeView
FileExists	RichTextEdit
GetFocus	All controls
Help	All controls
Hide	Window
HotLinkAlarm	Window
InputFieldSelected	RichTextEdit
ItemActivate	ListView
ItemChanged	DataStore
Key	All controls
LoseFocus	All controls
MouseDown	RichTextEdit, Window
MouseMove	RichTextEdit, Window
MouseUp	RichTextEdit, Window
Notify	TreeView
Other	All controls
PrintEnd	DataWindow
PrintMarginChange	DataWindow
PrintPage	DataWindow
PrintStart	DataWindow
RButtonDown	MultiLineEdit, SingleLineEdit, HScrollBar, VScrollBar

Event	Controls
RButtonUp	RichTextEdit
RemoteExec	Window
RemoteHotLinkStart	Window
RemoteHotLinkStop	Window
RemoteRequest	Window
RemoteSend	Window
RightClicked	ListView
RightDoubleClicked	All controls
ScrollHorizontal	DataWindow
ScrollVertical	DataWindow
Selected	Menu
Sort	ListView, TreeView
SystemKey	Window
ToolbarMoved	Window
ValueChanged	DatePicker

Partially supported
control events

Clicked event In a Web Forms application, if an editable DataWindow text column does not have focus, clicking it sets the focus on the column and triggers the Clicked event. If the column already has focus, clicking it does not trigger the Clicked event. This intended behavior reduces postbacks.

Selected event The Selected event on a menu is not generally supported in Web Forms applications. However, if you set the `AutoTriggerMenuSelectedEvents` global property to true, the Selected event is supported for simple tasks that can be run prior to the rendering of Web Forms in a client browser.

Unsupported properties for controls in Web Forms

Table 8-9 lists unsupported properties, the controls on which they are not supported, and any notes that apply to specific controls. If your application uses these properties, rework it to avoid their use.

Table 8-9: Unsupported properties in Web Forms projects

Property	Controls
Accelerator	All controls
AccessibleDescription	Most controls

Property	Controls
AccessibleName	Most controls
AccessibleRole	All controls
AllowEdit	DatePicker, DropDownListBox, DropDownPictureListBox
AutoArrange	ListView
AutoHScroll	DropDownListBox, DropDownPictureListBox, EditMask, MultiLineEdit, SingleLineEdit
AutoSize	MonthCalendar
AutoSkip	EditMask
AutoVScroll	EditMask, MultiLineEdit
BitmapBackColor	Menu
BitmapGradient	Menu
Border	Window
BorderStyle	RadioButton
BottomMargin	RichTextEdit
ButtonHeader	ListView
Center	Window (supported in child, popup, and response windows)
ClientEdge	Window
ColumnsPerPage	UserObject, Window
ContextHelp	Window
ControlCharsVisible	RichTextEdit
ControlMenu	DataWindow, Window (supported in child, popup, and response windows)
DeleteItems	ListView, TreeView
DisableDragDrop	TreeView
DisableNoScroll	ListBox, PictureListBox
DisplayData	EditMask
DisplayOnly	MultiLineEdit (supported, but control cannot get focus when set to true)
DragAuto	All controls
DragIcon	All controls
DropDownRight	DatePicker, EditMask
EditLabels	TreeView
ExtendedSelect	ListBox, ListView, PictureListBox
FillPattern	StaticHyperLink, StaticText
FixedLocations	ListView

Property	Controls
FocusOnButtonDown	Tab
FocusRectangle	Graph, Picture, PictureHyperlink, StaticText, StaticHyperlink
FullRowSelect	TreeView
GridLines	ListView
HeaderDragDrop	ListView
HeaderFooter	RichTextEdit
Height	RoundRectangle (does not change height if width is changed first; for HTrackBar, this property has no effect in a Windows application, but does in a Web application)
HideSelection	EditMask, ListView, MultiLineEdit, TreeView
HScrollbar	ListBox, RichTextEdit, Window
HSplitScroll	DataWindow
Icon	DataWindow
IgnoreDefaultButton	EditMask
IMEMode	All controls
Increment	EditMask
InputField properties	RichTextEdit
KeyboardIcon	Window
LabelWrap	ListView
LayoutRTL	ListView, TreeView
LeftMargin	RichTextEdit
LibraryName	UserObject
Limit	DropDownListBox, DropDownPictureListBox, EditMask
LinesAtRoot	TreeView
LinesPerPage	UserObject, Window
LiveScroll	DataWindow
Map3DColors	Picture, PictureButton, PictureHyperLink
MaxBox	DataWindow, Window (supported in child, popup, and response windows)
MaxSelectCount	MonthCalendar
MenuAnimation	Menu
MenuBitmaps	Menu
MenuTitles	Menu
MenuTitleText	Menu
MergeOption	Menu

Property	Controls
MicroHelp	Menu
MinBox	DataWindow, Window (supported in child, popup, and response windows)
MinMax	EditMask
Modified	RichTextEdit
OneClickActivate	ListView
OriginalSize	Animation
PaletteWindow	Window
PictureAsFrame	RichTextEdit
PictureHeight	Tree View
PictureWidth	Tree View
PopupMenu	RichTextEdit
Resizable	DataWindow, RichTextEdit, Window
RightMargin	RichTextEdit
RightToLeft	DataWindow, DropDownListBox, DropDownPictureListBox, ListBox, ListView, PictureListBox, StaticText, StaticHyperlink, TreeView, Window
RulerBar	RichTextEdit
Scrolling	ListView
ScrollRate	MonthCalendar
SelectedStartPos	RichTextEdit
SelectedTextLength	RichTextEdit
ShowHeader	ListView
ShowList	DropDownListBox, DropDownPictureListBox
ShowToolbarText	Window (supported, but width of text is changed)
ShowUpDown	DatePicker
SingleExpand	Tree View
SliderSize	HTrackBar, VTrackBar
SmoothScroll	HProgressBar, VProgressBar (smooth scrolling is supported, but not step increments)
Spin	EditMask
StatePictureHeight	Tree View
StatePictureWidth	Tree View
StatusBar	RichTextEdit
Style	UserObject
TabBackColor	UserObject

Property	Controls
TabBar	RichTextEdit
TabStop	EditMask, ListBox, MultiLineEdit, PictureListBox
TickFrequency	HTrackBar, VTrackBar
TickMarks	HTrackBar, VTrackBar
Title	DataWindow
TitleBackColor	Menu
TitleBar	DataWindow, Window
TitleGradient	Menu
TodaySection	DatePicker, MonthCalendar
ToolBarAlignment	Window
ToolBarAnimation	Menu
ToolBarFrameTitle	Application
ToolBarHeight	Window
ToolBarItemDown	Menu
ToolBarItemDownName	Menu
ToolBarItemSpace	Menu
ToolBarPopMenuText	Application
ToolBarSheetTitle	Application
ToolBarUserControl	Window
ToolBarWidth	Window
ToolBarX	Window
ToolBarY	Window
TopMargin	RichTextEdit
TrackSelect	ListView, TreeView
TwoClickActivate	ListView
UnderlineCold	ListView
UnderlineHot	ListView
UnitsPerColumn	UserObject, Window
UnitsPerLine	UserObject, Window
UseCodeTable	EditMask
VScrollbar	RichTextEdit, Window
WeekNumbers	DatePicker, MonthCalendar
Width	VTrackBar (has no effect in a Windows Form application, but does in a Web Forms application)
WordWrap	RichTextEdit

Windows Forms Targets

This part describes how to create and deploy Windows Forms applications.

Deploying PowerBuilder Applications as .NET Windows Forms

About this chapter

PowerBuilder includes the .NET Windows Forms Application wizard, which helps you deploy your PowerBuilder applications as .NET Windows Forms applications. You can also use intelligent update technology to deploy and maintain applications.

This chapter explains how to generate, deploy, and run PowerBuilder applications as Windows Forms applications.

Contents

Topic	Page
About PowerBuilder .NET Windows Forms applications	125
Creating a .NET Windows Forms target	127
Creating a .NET Windows Forms project	129
Setting properties for a .NET Windows Forms project	130
Deploying the project from PowerBuilder	134
Running the project from PowerBuilder	135

About PowerBuilder .NET Windows Forms applications

PowerBuilder applications that have a rich user interface that relies on resources available on the client computer, such as a complex MDI design, graphics, or animations, or that perform intensive data entry or require a rapid response time, make good candidates for deployment as .NET Windows Forms applications. For a comparison of design considerations between Web Forms and Windows Forms applications, see “Choosing a .NET application target” on page 3.

- Adapting an existing application The changes required to transform a PowerBuilder application into a Windows Forms application depend on the nature of the application, the scripting practices used to encode the application functionality, and the number of properties, functions, and events the application uses that are not supported in the .NET Windows Forms environment.
- For a list of restrictions, most of which apply to both Windows and Web Forms applications, see Chapter 14, “Best Practices for .NET Projects.”
- For tables of unsupported and partially supported objects, controls, functions, events, and properties, see Chapter 11, “Unsupported Features in Windows Forms Projects.”
- Setting up a target and project You set up a target for a .NET Windows Forms application using the wizard on the Target page of the New dialog box. You can start from scratch and create a new library and new objects, use an existing application object and library, or use the application object and library list of an existing target.
- You define some of the characteristics of the deployed application in the .NET Windows Forms Application wizard. Additional properties are set in the Project painter. For more information, see “Creating a .NET Windows Forms target” on page 127, “Creating a .NET Windows Forms project” on page 129, and “Setting properties for a .NET Windows Forms project” on page 130.
- Smart client applications One of the choices you can make in the wizard or Project painter is whether the application will be deployed as a smart client application. A smart client application can work either online (connected to distributed resources) or offline, and can take advantage of “intelligent update” technology for deployment and maintenance. For more information, see Chapter 10, “Intelligent Deployment and Update.”
- Deploying from the Project painter When you deploy a PowerBuilder application from the .NET Windows Forms Project painter, PowerBuilder builds an executable file and deploys it along with any PBLs, PBDs, resources, .NET assemblies, and other DLLs that the application requires. For more information, see “Deploying the project from PowerBuilder” on page 134.
- Deploying to a production environment The simplest way to deploy a Window Forms application to a production environment is to use smart client deployment. If you cannot or do not want to use smart client deployment, use the following procedure to install the application.
- ❖ **To deploy a .NET Windows Forms application:**
- 1 Install the .NET Framework 2.0 on the target computer.

- 2 Generate a PowerBuilder .NET components MSI file using the PowerBuilder Runtime Packager.
For more information about using the Runtime Packager, see the chapter on deploying applications and components in *Application Techniques*.
- 3 Install the generated MSI file on the target computer and restart the computer.
- 4 Copy the output from the build directory to the target computer.
- 5 Install any required database client software and configure related DSNs.
- 6 If necessary, register ActiveX controls used by your application.

Using preprocessor symbols

If you share PBLs among different kinds of target, such as a target for a standard PowerBuilder application and a Windows Forms target, you might want to write code that applies to a specific target. For example, use the following template to enclose a block of code that should be parsed by the pb2cs code emitter in a Windows Forms target:

```
#if defined PBWINFORM then
    /*action to be performed in a Windows Forms target*/
#else
    /*other action*/
#endif if
```

You can use the Paste Special>Preprocessor pop-up menu item in the Script view to paste a template into a script.

For more information about using preprocessor symbols, see “About conditional compilation” on page 187.

Creating a .NET Windows Forms target

System requirements

You must install version 2.0 of the Microsoft .NET Framework on the same computer as PowerBuilder 11. For intelligent update applications, you must also install the .NET Framework 2.0 SDK. Make sure that the system PATH environment variable includes:

- The location of the .NET Framework. The location of the GA version is typically *C:\Windows\Microsoft.NET\Framework\v2.0.50727*.

- For intelligent update applications, the location of the .NET Framework SDK *Bin* directory. This is typically *C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin* or *C:\Program Files\Microsoft.NET\SDK\v2.0\Bin*.

The SDK is available from the Microsoft .NET Framework Developer Center at <http://msdn2.microsoft.com/en-us/netframework/aa731542.aspx>.

If you installed the 1.x and 2.0 versions of the .NET Framework or SDK, you must make sure the PATH variable lists the 2.0 version first.

To publish your application as a smart client from a Web server, you must have access to a Web server. For information about configuring IIS on your local computer, see “Selecting the default ASP.NET version” on page 7.

You use the PowerBuilder .NET Windows Forms Application Wizard on the Target page in the New dialog box to create a Windows Forms application and target, and optionally a project. The project lets you deploy the PowerBuilder application to the file system or, if you select the smart client option, to publish it to a server. For more about publishing options, see Chapter 10, “Intelligent Deployment and Update.”

If you have an existing PowerBuilder application or target that you want to deploy as a .NET Windows Forms application, you can select either in the wizard. If you choose to start from scratch, the wizard creates a new library and application object

- ❖ **To build a .NET Windows Forms application and target from scratch:**
 - 1 Select Start from scratch on the Create the Application page in the wizard.
 - 2 Specify the name of the .NET Windows Forms application and the name and location of the PowerBuilder library (PBL) and target (PBT). By default, the application name is used for the library and target.
 - 3 Specify project information as described in “Creating a .NET Windows Forms project” next.
- ❖ **To build a .NET Windows Forms application from an existing application and library:**
 - 1 Select Use an existing library and application object on the Create the Application page in the wizard.
 - 2 On the Choose Library and Application page, expand the tree view and select an existing application.
 - 3 On the Set Library Search Path page, click the ellipsis (...) button to navigate to and select additional libraries.

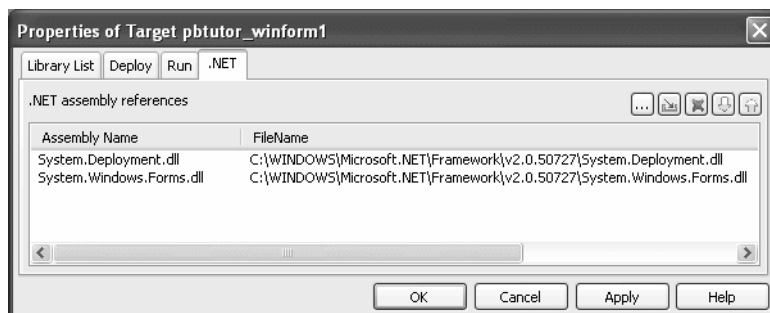
- 4 On the Specify Target File page, specify the name of the new target file.
- 5 Specify project information as described in “Creating a .NET Windows Forms project” next.

❖ **To build a .NET Windows Forms application from an existing target:**

- 1 Select Use the library list and application object of an existing target on the Create the Application page in the wizard.
- 2 On the Choose a Target page, select a target from the current workspace.
- 3 On the Specify Target File page, specify the name of the new target file.
- 4 Specify project information as described in “Creating a .NET Windows Forms project” next.

Importing .NET assemblies into the target

If your application uses .NET assemblies, specify them on the .NET tab page in the target’s Properties dialog box. Before you deploy a PowerBuilder .NET Windows Forms application, make sure the *System.Windows.Forms.dll* and *System.Deployment.dll* assemblies are listed on this page.



Creating a .NET Windows Forms project

You can create a project to deploy the application in the target wizard or by using the .NET Windows Forms wizard on the Project page of the New dialog box.

❖ **To build a .NET Windows Forms project object:**

- 1 On the Specify Project Information page, specify the name of the project and the library in which the project object will be saved.

- 2 On the Specify Application General Information page, optionally specify a product name for the application.

This can be different from the name of the application and is used as the name of the product on the General page in the Project painter.

You can also specify the name of the .NET Windows Forms executable file (by default, this is the name of the application object with the extension *.exe*) and the major and minor versions and build and revision numbers for the current build (the default is 1.0.0.0).

- 3 On the Specify Win32 Dynamic Library Files page, click the Add button to specify the names of any dynamic libraries required by your application.

The list is prepopulated with the names of libraries referenced in the application's code.

- 4 On the Specify Support for Smart Client page, select the check box if you want to publish the application as a smart client. Otherwise, click Next and then Finish.

If you select this check box, the wizard displays additional pages on which you set publish and update options. For more information about completing these wizard pages, see Chapter 10, "Intelligent Deployment and Update."

Setting properties for a .NET Windows Forms project

After you click Finish in the wizard, PowerBuilder creates a .NET Windows Forms project in the target library that you selected and opens the project in the Project painter. The painter displays all the values you entered in the wizard and allows you to modify them. It also displays additional properties that you can set only in the painter.

Table 9-1: Properties in the Project painter

Tab page	Properties
General	<p>The output path is where the application is deployed in the file system. This is not the same as the location where the application is published if you choose to publish the application as a smart client application.</p> <p>The build type determines whether the project is deployed as a debug build (default selection) or a release build. You use debug builds for debugging purposes. If you select Release, no PDB files are generated. Release builds have better performance, but when you run a release build in the debugger, the debugger does not stop at breakpoints.</p> <p>Clear the Enable DEBUG Symbol check box if you do not want any DEBUG preprocessor statements you have added to your code to be included in your deployed application. This selection does not affect and is not affected by the project's debug build or release build setting. For more information about using preprocessor statements, see "About conditional compilation" on page 187.</p>
Resource Files	<p>PowerBuilder .NET Windows Forms do not support PBR files, and they are unable to locate images embedded in PBD files.</p> <p>You can, however, search a PBR file for images required by the application.</p> <p>All resource files must be relative to the path of the .NET Windows Forms target. If the files your application requires are in another directory, copy them into the target's directory structure and click the Search PBR, Add Files, or Add Directory button again.</p> <p>Clear the check box in the Recursive column for a directory to deploy only the files in the directory, or select it to deploy files in its subdirectories as well.</p> <p>The Publish Type column indicates whether the file is a static file that should be installed in the Application directory, or application-managed data that should be installed in a Data directory. For more information, see "Resource files and publish type" on page 133.</p>

Tab page	Properties
Library Files	<p data-bbox="548 230 1189 317">Use the Library Files tab page to make sure all the PowerBuilder library files (PBLs or PBDs) that contain DataWindow or query objects used by the application are deployed with the application.</p> <hr/> <p data-bbox="548 361 1189 413">PBD files can contain only DataWindow and Query objects</p> <p data-bbox="548 413 1189 526">PBD files that you deploy to .NET as resource files can contain only DataWindow and Query objects. If your application references a PBD file that contains other PowerBuilder objects, such as functions or user objects, you must deploy it as a PBL file.</p> <hr/> <p data-bbox="548 569 1189 713">If your application uses external functions, use the Add button to include the DLL files in which they reside to the list of files to be deployed. You can also add PowerBuilder runtime files, including <i>pbshr110.dll</i> and <i>pbdwe110.dll</i> (if the project uses DataWindows), on this page, or you can add them on the Prerequisites page.</p>
Version	<p data-bbox="548 722 1189 835">Use the Version tab page to specify information that displays in the generated executable file's Properties dialog box in Windows Explorer. The company name is used if you publish the application. For more information, see "Publish the application" on page 140.</p>
Post-build	<p data-bbox="548 843 1189 1052">Use the Post-build page to specify a set of commands to be executed after building the application, but before the deployment process starts. A command can be the name of a stand-alone executable file or an operating system command such as copy or move. You can save a separate processing sequence for debug builds and release builds. (You change the build type of a project deployment on the General tab of the Project painter.)</p>
Run	<p data-bbox="548 1060 1189 1142">Use the Run page to specify any command line arguments that the application requires, as well as the name of the working directory in which the application starts.</p>

Intelligent update pages

The remaining pages in the Project painter are enabled if you checked the smart client check box in the wizard or on the General page. Check this box if you want to publish the application to a server so that users can download it and install updates as you make them available. For more information, see Chapter 10, "Intelligent Deployment and Update."

Resource files and publish type

Click the Add Files button on the Resource Files page to select image files that your application requires. PowerBuilder .NET Windows Forms do not support PBR files, and they are unable to locate images embedded in PBD files. All resource files must be relative to the path of the .NET Windows Forms target. If the files your application requires are not in the directory structure accessible from the Choose Required Resource Files dialog box, copy them into the directory structure, then reopen the dialog box.

Image files are designated as Include files. They are installed in the same directory as the application's executable files, libraries, and other static files. You can also specify that a file's Publish Type is "Data File." Files of this type are installed to a data directory. When an update to the application occurs, a data file might be migrated by the application.

The data directory is intended for application-managed data—data that the application explicitly stores and maintains. To read from and write to the data directory, you can use code enclosed in a conditional compilation block to obtain its path:

```
string is_datafilename
long li_datafileid

is_datafilename="datafile.txt"

#if defined PBWINFORM Then
  if System.Deployment.Application.
    ApplicationDeployment.IsNetworkDeployed=true then
    is_datafilename=System.Windows.Forms.
      Application.LocalUserAppDataPath+
        "\\ "+is_datafilename
    end if
  #end if

  li_datafileid = FileOpen (is_datafilename, linemode!,
    write!, lockwrite!, append!)
```

For more information about using preprocessor symbols and conditional compilation, see Chapter 13, "Referencing .NET Classes in PowerScript."

Other files, such as database drivers and PowerBuilder DLLs, should be included on the Prerequisites page if you are publishing a smart client application, or on the Library Files page.

Deploying the project from PowerBuilder

When a .NET Windows Forms project is open in the Project painter, you can select Design>Deploy Project or the Deploy icon on the PainterBar to deploy the project. When all painters are closed, including the Project painter, you can right-click a .NET Windows Forms target or project in the System Tree and select Deploy from its pop-up menu. If the target has more than one project, specify which of them to deploy when you select Deploy from the target's pop-up menu on the Deploy tab page in the target's Properties dialog box.

The Output window displays the progress of the deployment. Any PBLs referenced in the application are compiled into PBD files. The application and its supporting files are deployed to the location specified in the Output Path field on the General page.

Among the files deployed is a file with the name *appname.exe.config*, where *appname* is the name of your application. This file is a .NET configuration file that defines application settings. For a sample configuration file that includes database configuration settings for an ADO.NET connection, see the chapter on ADO.NET in *Connecting to Your Database*. The sample shows how to configure tracing in the *appname.exe.config* file, as shown in “Runtime errors” on page 224.

If there are any unsupported properties, functions, or events that are used in the application that are not supported in PowerBuilder .NET Windows Forms applications, they display on the Unsupported Features tab page in the Output view. For more information, see Chapter 11, “Unsupported Features in Windows Forms Projects.”

If the application uses features that might cause it to run incorrectly, they display on the Warnings tab page in the Output view. For a list of restrictions, most of which apply to both Windows and Web Forms applications, see Chapter 14, “Best Practices for .NET Projects.”

Running the project from PowerBuilder

After you deploy the application, you can run it by selecting Design>Run Project from the Project painter menu or selecting the Run Project toolbar icon from the Project painter toolbar. The pop-up menus for the .NET Windows Forms target and project in the System Tree also have a Run menu item. If the target has more than one project, specify which of them to run when you select Run from the target's pop-up menu on the Run tab page in the target's Properties dialog box. Run Project starts running the deployed executable file from the location it was deployed to.

For information on debugging .NET Windows Forms targets, see “Debugging a .NET application” on page 213.

Intelligent Deployment and Update

About this chapter

This chapter describes how to deploy a PowerBuilder application as a Windows Forms application that can use the smart client intelligent update feature.

Contents

Topic	Page
About intelligent deployment and update	137
Publishing an application for the first time	138
Installing the application on the user's computer	142
Updating the application	143
Using the bootstrapper	146
Rolling back	148

About intelligent deployment and update

One of the features of .NET smart client applications is that they can be deployed and updated from a file or Web server using Microsoft .NET ClickOnce technology, making it easier for users to get and run the latest version of an application and easier for administrators to deploy it. PowerBuilder .NET Windows Forms applications can use this “intelligent update” feature.

As the developer of a PowerBuilder .NET Windows Forms application, you can specify:

- Whether the application is installed on the user's computer or run from a browser.
- When and how the application checks for updates.
- Where updates are made available.
- What files and resources need to be deployed with the application.
- What additional software needs to be installed on the user's computer.

All these properties can be set in the Project painter before you publish the application. Support for these features is built into the .NET Framework and runtime.

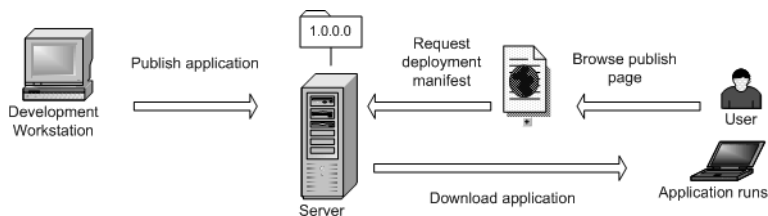
To support intelligent update, you (or a system administrator) need to set up a central HTTP, FTP, or UNC file server that supports file downloads. This is the server to which updates are published and from which they are deployed to the user's computer.

When the user clicks on a link, typically on a Web page or in an email, the application files are downloaded to a secure cache on the user's computer and executed. The application itself contains an updater component. If the application can only be run when the user is connected, the latest version is always downloaded. If the application can also be run offline, the updater component polls the server to check whether updates are available. If they are, the user can choose to download them.

Publishing an application for the first time

When you are ready to deploy an application to users, you publish it to the server. Users can then download the application, usually from a publish page that contains a link to the server.

Figure 10-1: Deploying an intelligent update application



You need to:

- Create a project and set publishing properties
- Publish the application

Create a project and set publishing properties

After you develop a PowerBuilder application that will be published as a .NET Windows Forms application with intelligent update capabilities, if you did not create a project when you built the application, select the .NET Windows Forms Application wizard or project icon on the Project page of the New dialog box to build a project.

To specify that the application uses intelligent update, select the check box on the Specify Support for Smart Client page in the wizard. Selecting this check box enables additional pages in the wizard:

- On the Specify Application Running Mode page, specify whether the application can be used both online and offline (the default), or online only.
- On the Specify How Application Will be Installed page, specify whether the user installs the application from a Web site, a shared path, or from a CD or DVD.
- On the Specify Application Update Mode page, specify whether the application checks for updates before starting, after starting, or neither. For more information, see “Updating the application” on page 143.

You can also select the Publish as a Smart Client Application check box on the General page in the Project painter. Selecting the check box enables the tab pages in the dialog box where you set publishing properties. You can set additional properties in the Project painter. For example, if you want to publish the application to an FTP site, select that option and specify details on the Publish page.

Locations for publish, install, and update

The publish location, specified on the Publish page in the Project painter, determines where the application files are generated or copied to when you publish the application. It can be an HTTP address, an FTP site, or a UNC address.

The install location, specified on the Install/Update page, determines where the end user obtains the initial version of the application. It can be an HTTP address or UNC address, by default the same address as the publish location specified in the wizard, or a CD or DVD. The install location does not need to be the same as the publish location. For example, you can publish the application to an FTP site, but specify that users get the application and updates from a Web site.

The update location, also specified on the Install/Update page, determines where the user obtains updated versions of the application. If the install location is an HTTP address or UNC address, the update location is always the same as the install location. If the application was installed from a CD or DVD, updates must be obtained from an HTTP or UNC address.

Security settings

Applications installed using intelligent update typically run with a limited set of permissions based on a security zone that depends on how the application was originally installed. Full Trust is used for applications installed from a CD or DVD, Local Intranet Trust is used when the application is installed from a network share, and Internet Trust is used when the application is run or installed from the Web. In PowerBuilder 11, Full Trust is always used.

Full trust permissions are required for local intranet deployment. When you deploy and run an application from a network path (either a path on a mapped drive or a UNC path), the .NET Framework on the computer must be configured to have Full Trust permissions at runtime. To set these permissions, select Administrative Tools>Microsoft .NET Framework 2.0 Configuration from the Windows control panel. In the .NET Framework Configuration tool, expand My Computer and select Runtime Security Policy>Machine>Code Groups>All_Code>LocalIntranet_Zone. Select Properties from the pop-up menu and select FullTrust in the Permission set drop-down list on the Permission Set tab page.

Publish the application

After you set publish properties, click the Publish button on the toolbar in the Project painter to publish the application to the server.

PowerBuilder checks whether your publish settings are valid and prompts you to correct them if necessary. If the application is not up to date, PowerBuilder rebuilds and redeploys it before publishing it to the server. The files that the application needs at runtime are then published to the server. If you select the defaults in the wizard, the application is deployed to a subdirectory of the IIS root directory on your local computer, usually *C:\inetpub\wwwroot*.

If you encounter problems when publishing the application, see “Troubleshooting tips for Windows Forms applications” on page 224.

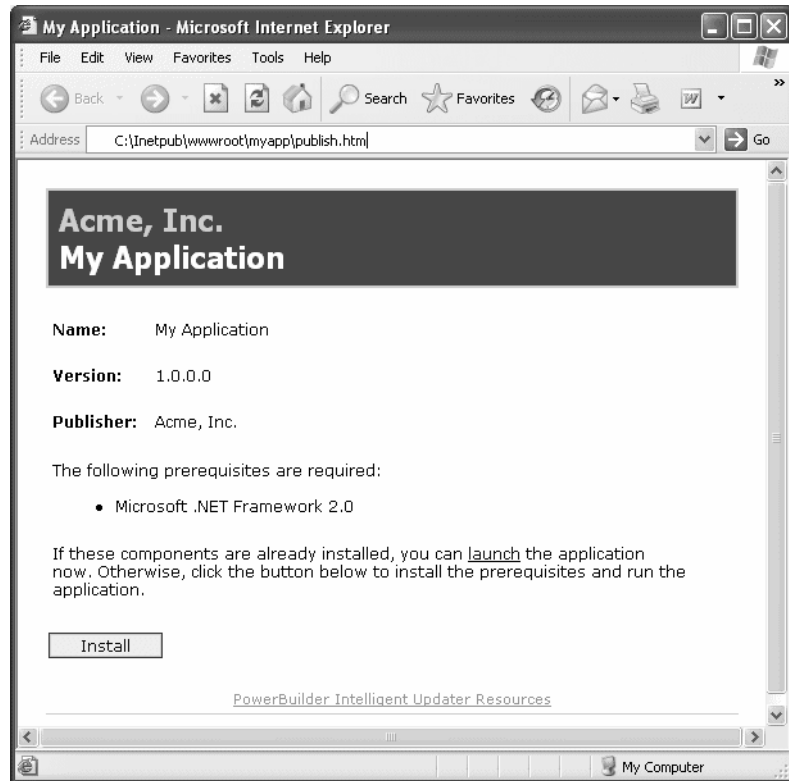
The following additional files are created on the server:

- The *application manifest* is an XML file that describes the deployed application, including all the files included in the deployment, and is specific to a single version of the application. The file is named *appname.exe.manifest*, where *appname* is the name of your Windows Forms application. This file is stored in a version-specific subdirectory of the application deployment directory.

- The *deployment manifest* is an XML file that describes an intelligent update deployment, including the current version and other deployment settings. The file is named *appname.application*, where *appname* is the name of your Windows Forms application. It references the correct application manifest for the current version of the application and must therefore be updated when you make a new version of the application available. The deployment manifest must be strongly named. It can contain certificates for publisher validation.
- If you specified any prerequisites for the application, such as the .NET Framework or database drivers, PowerBuilder uses a bootstrapper program to collect the details in a configuration file called *configuration.xml* and adds the prerequisites to a *setup.exe* program. For more information, see “Using the bootstrapper” on page 146.
- The *publish.htm* file is a Web page that is automatically generated and published along with the application. The default page contains the name of the application and links to install and run the application and, if you specified any, a button to install prerequisites.

By default, the application name is the same as the name of the target and the company name is Sybase, Inc. In this publish page, both have been changed by setting the Product Name and Company Name properties on the General page in the Project painter.

Figure 10-2: Publish page with prerequisites



Installing the application on the user's computer

Users can install the application from a CD or DVD or from a file server or Web site. The system administrator or release engineer is responsible for writing the files to the disk if a CD or DVD is used. If the files are available to the user on a server, the *publish.htm* file provides easy access to the application and its prerequisites. For more information about prerequisites, see "Using the bootstrapper" on page 146.

The application can be available both online and offline or online only. If you select online only, the application can be run only from the Web. Otherwise, the application is installed on the client. It can be run from the Windows Start menu and is added to the Add or Remove Programs page in the Windows control panel so that the user can roll back to the previous version or remove the application.

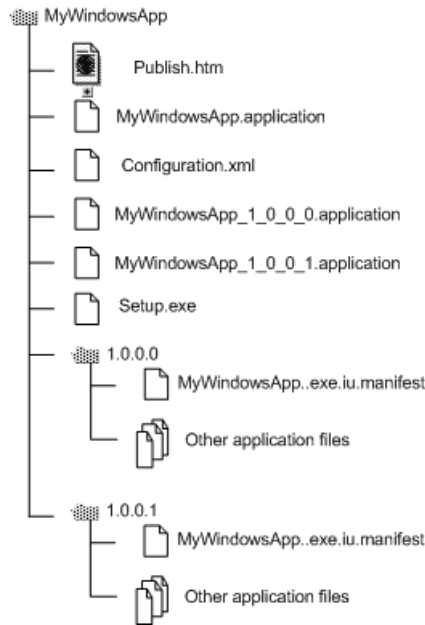
Whether the application is available online only or offline as well, all the files it needs except optional assemblies are downloaded to the client and stored in an application-specific secure cache in the user's Local Settings directory. Keeping the files in a separate cache enables the intelligent updater to manage updates to the physical files on the user's computer.

Updating the application

When you update an application and publish the updates, the revision number is incremented automatically unless you clear the check box in the Publish Version group box on the Publish page.

PowerBuilder creates a new directory on the server for the new version with a new application manifest file, and updates the deployment manifest file in the top-level directory. Figure 10-3 shows an overview of the directory structure for an application with one revision. The deployment manifest for each version is saved in a numbered file, which enables you to force a rollback from the server if you need to. For more information, see "Rolling back" on page 148.

Figure 10-3: Published file structure



Online only applications

If the application is available online only, the latest updates are always downloaded before the application runs.

Online and offline applications

If the application is available offline as well as online, the user is notified of new updates according to the update strategy you specified in the wizard or Project painter. Whether the application was originally installed from the Web, a file server, or a CD or DVD, the intelligent updater component always checks for updates on the Web.

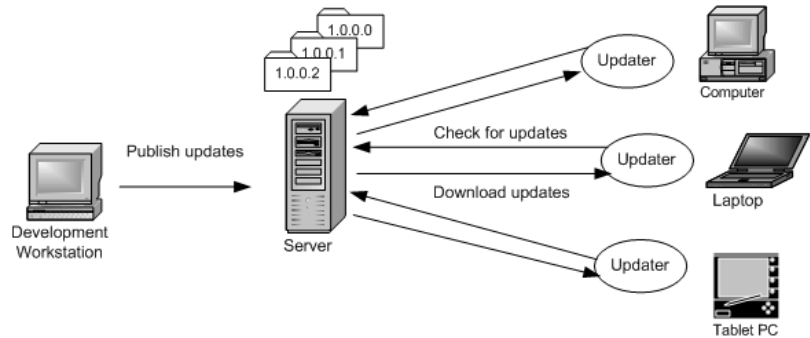
When to check for updates

You can specify that the application never checks for updates (if it does not require automatic updating or uses a custom update), or that it checks for updates either before or after it starts. If you specify a check after the application starts and an update is available, it can be installed the next time the application is run.

For high-bandwidth network connections, you might want to use the before startup option, and for low-bandwidth network connections or large applications, use the after startup option to avoid a delay in starting the application. If you specify that the intelligent updater performs the check after the application starts, you can choose to perform the check every time the application starts or only when a specified interval has elapsed since the last check.

If an update is available, a dialog box displays to inform the user, who can choose to download the update immediately or skip the current update and check again later. The user cannot skip the update if you have specified that it is mandatory. You set all these properties on the Install/Update page.

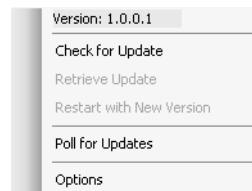
Figure 10-4: Checking for updates



Intelligent notifier

When you select either of the check for updates options for an application that is available offline, the Notify tab page is enabled. The notifier enables users to check for updates and download them manually while the application is running. When the application starts, a notifier icon displays in the task bar. By default, the icon is a PowerBuilder icon, but you can choose a custom icon in the Project painter.

The pop-up menu that displays when a user right-clicks the notifier icon displays the current version and contains Check for Update, Retrieve Update, Restart with New Version, Poll for Updates, and Options menu items.



Check for Update opens a pop-up window that contains information about the availability of updates. If any are available, the Retrieve Update item is enabled, and if the update is downloaded and installed, the Restart with New Version item is enabled.

Selecting the Poll for Updates item enables or disables polling for updates. When Poll for Updates is enabled, the notifier checks for updates at the interval specified in the dialog box that displays when the user selects the Options item. In this dialog box, the user can also specify the title of the pop-up window that displays when the user selects Check for Update.

Using the bootstrapper

To ensure that your application can be successfully installed and run, you must first make sure that all components on which it depends are already installed on the target computer. For example, most applications have a dependency on the .NET Framework. The correct version of the common language runtime must be present on the destination computer before the application is installed. You can use tools to help you install the .NET Framework and other redistributable packages as a part of your installation, a practice often referred to as bootstrapping.

Bootstrapper for intelligent update

The bootstrapper is a simple setup packager that can be used to install application prerequisites such as the .NET Framework, MDAC, database drivers, or PowerBuilder runtime files. You specify what prerequisites your application has and where they can be found. The bootstrapper downloads and installs the prerequisites.

If you select one or more prerequisites on the Prerequisites page, PowerBuilder generates a Windows executable program named *Setup.exe* that installs these dependencies before your application runs. The packages are copied to a *SupportFiles* directory on the server.

If a *Setup.exe* is generated, the *Publish.htm* page contains a link to install just the application, and a button to install both the application and the bootstrapped components, as shown in Figure 10-2.

The bootstrapper lets you provide users with a simple, automated way to detect, download, and install an application and its prerequisites. It serves as a single installer that integrates the separate installers for all the components making up an application.

How the bootstrapper works

When the user clicks the Install button on the *Publish.htm* page, the bootstrapper downloads and installs the application and the prerequisites you specified if they are not already installed on the user's computer.

For example, suppose you specified that the application required the .NET Framework and the PowerBuilder 11 runtime files. If neither of these components is already installed on the user's computer, they both display in the Installation dialog box. If both are already installed, they do not display. If the user clicks the Advanced button on the Installation dialog box, the Components List dialog box displays. This dialog box shows that both components are already installed.

The bootstrapper also detects whether a component is supported on the target computer's operating system. If the component cannot run on the target platform, the bootstrapper notifies the user and ends the installation before downloading the component.

Customizing the Prerequisites page

The selections available on the Prerequisites page can be customized by adding a new subdirectory to the *PowerBuilder 11.0\DotNET\pb11\BootStrapper\Packages* directory. To this subdirectory, you add the package you want to make available and an XML configuration file that specifies where to obtain the package and what to check on the user's system to determine whether the package needs to be installed.

PowerBuilder 11 does not supply a tool to customize prerequisites. You can use the PowerBuilder Runtime Packager tool to build an MSI file that contains the database drivers and other PowerBuilder runtime files that your application needs, and use the *configuration.xml* file in the *BootStrapper\Packages\1-PBRuntime* directory as an example when creating your own *configuration.xml* file.

You can use the InstallerEditor (dotnetInstaller) open source tool to set up your own customizations. It is described on the DevAge Web site at <http://www.devage.com/> and can be downloaded from the dotnetInstaller project page on the SourceForge Web site at <http://www.sourceforge.net>. Read the instructions on the DevAge Web site to learn how to use the tool.

Packages on the Prerequisites page

There are two packages available on the Prerequisites page: the .NET Framework 2.0 runtime files and the PowerBuilder 11 Runtime Library. If you look in the *BootStrapper\Packages* directory, you see two subdirectories, each of which contains a *configuration.xml* file.

To enable your application to deploy the .NET Framework 2.0 package, you need to copy the .NET Framework 2.0 redistributable package, *dotnetfx.exe*, to the *0-dotnetfx* directory. This file can be downloaded from the Microsoft Web site. You also need to edit the *configuration.xml* file to ensure that the application name and locations specified in the file are correct for your installation. The file uses *http://localhost/SampleApp* as the source URL for the package.

The PowerBuilder 11 Runtime Library package is in the *1-PBRuntime* subdirectory. The *PBRuntime.msi* file installs the same files as the PowerBuilder Runtime Packager (with .NET and all database interfaces and other options selected) into a directory on the target computer, and it installs the same .NET assemblies into the Global Assembly Cache (GAC). If you do not require all the files included in the package, you can create your own package. For more information, see “Customizing the Prerequisites page” on page 147.

For more information about the Runtime Packager, see the chapter on deployment in *Application Techniques*.

For more information about editing *configuration.xml* files, see the documentation for the available on the DevAge Web site at <http://www.devage.com/>.

Rolling back

You can roll back a version on the server by replacing the current deployment manifest with the deployment manifest of the version to which you want to roll back. As shown in Figure 10-3 on page 144, the deployment manifests for each version are saved in the application deployment folder.

Suppose the current *appname.application* file in the deployment folder is for version 1.0.0.2, but you have found a bug and you want all users to revert to version 1.0.0.1. You can delete the current *appname.application* file, which points to version 1.0.0.2, and save the *appname_1_0_0_1.application* file as *appname.application*.

Users on whose computers the application has been installed for use offline as well as online can roll back to the previous version or uninstall the application completely from the Windows control panel’s Add/Remove Programs dialog box. Users can roll back only one update.

Using MobiLink synchronization

You can use MobiLink synchronization with smart client applications to take advantage of the “occasionally connected” nature of a Windows Forms application that has been installed on a client so that it can be run from the Start menu as well as from a browser.

MobiLink is a session-based synchronization system that allows two-way synchronization between a main database, called the consolidated database, and many remote databases. The user on the client computer can make updates to a database when not connected, then synchronize changes with the consolidated database when connected.

You need to deploy the SQL Anywhere database driver and the MobiLink synchronization client file to the client computer. You can simplify this process by adding the required files to a package and adding the package to the Prerequisites page in the Project painter.

For more information, see “Using the ASA MobiLink synchronization wizard” in the *User’s Guide* and Chapter 13, “Using MobiLink Synchronization,” in *Application Techniques*.

Unsupported Features in Windows Forms Projects

About this chapter

This chapter lists controls, classes, and system functions that are not fully supported in Windows Forms applications in this release.

Contents

Topic	Page
About unsupported features	151
Unsupported nonvisual objects and structures in Windows Forms	153
Unsupported system functions in Windows Forms	157
Partially supported visual controls for Windows Forms	158
Unsupported functions for controls in Windows Forms	162
Unsupported events for controls in Windows Forms	163
Unsupported properties for controls in Windows Forms	164

About unsupported features

PowerBuilder .NET Windows Forms applications do not currently support some features. Some are not implemented in PowerBuilder 11.0 and some have been partially implemented.

The tables in this chapter provide detailed lists of all objects, controls, functions, events, and properties and indicate whether they are supported

The following list summarizes support in Windows Forms for features in this release:

- All DataWindow presentation styles are supported, but there are some restrictions on RichText and OLE presentation styles.
- External function calls are supported except when the function has a reference structure parameter.
- You cannot call functions on .NET primitive types that map to PowerBuilder primitive types. See Table 13-3 for the list of datatype mappings from .NET to PowerBuilder.

- You can use the built-in Web services client extension (*pbwsclient110.pbx*) in applications that you plan to deploy to .NET Windows Forms. You *cannot* use any other PBNI extensions in a .NET target.
- In-process OLE controls (controls with the extension *.ocx* or *.dll*) are partially supported. Most of the OLE control container's events are not supported, but events of the control in the container are supported with the exception of the Help event. Other OLE features are not supported. You cannot create an ActiveX control dynamically, and you must set the initial properties of an ActiveX control in code because the implementation does not support saving to or retrieving from structured storage.

Support for OLE controls requires the Microsoft ActiveX Control Importer (*aximp.exe*). This tool generates a wrapper class for an ActiveX control that can be hosted on a Windows Form. It imports the DLL or OCX and produces a set of assemblies that contain the common language runtime metadata and control implementation for the types defined in the original type library. When you deploy the application, you deploy these assemblies. You do not need to deploy *aximp.exe*.

The *aximp.exe* tool is part of the .NET Framework 2.0 SDK, which can be freely downloaded from the Microsoft Web site. For more information, see "System requirements" on page 127.

- The following features are not currently supported in .NET targets: shared objects, EAServer integration, COM/COM+ components, OLE automation server, data pipelines, tracing and profiling, DDE functions, and SSLCallback.
- The .NET Framework replaces fonts that are not TrueType fonts, such as MS Sans Serif, with TrueType fonts. This replacement can cause unexpected display issues. For example, the cursor does not display when you click in an EditMask control that does not use a TrueType font. To avoid such issues, always use a TrueType font such as Tahoma.

Unsupported nonvisual objects and structures in Windows Forms

This section contains two tables:

- Table 11-1 lists all PowerBuilder nonvisual objects and structures and indicates whether they are supported in this release.
- When there is an X in the partially supported column in Table 11-1, see Table 11-2 on page 155 for detailed information about what is supported. XX in the Unsupported column indicates that there are currently no plans to support the object.

Objects used for profiling and tracing, DDE, and OLE storage and streams are not supported.

Table 11-1: Support for nonvisual objects in Windows forms

Class name	Supported	Partially supported	Unsupported
AdoResultSet	X		
Application		X	
ArrayBounds	X		
ClassDefinition *	X		
ClassDefinitionObject	X		
Connection	X		
ConnectionInfo	X		
ConnectObject	X		
ContextInformation	X		
ContextKeyword			XX
CorbaCurrent			X
CorbaObject			X
CorbaSystemException (and its descendants)		X	
CorbaUnion			X
CorbaUserException	X		
DataStore		X	
DataWindowChild		X	
DivideByZeroError	X		
DWObject	X		
DWRuntimeError	X		
DynamicDescriptionArea	X		
DynamicStagingArea	X		

Class name	Supported	Partially supported	Unsupported
EnumerationDefinition		X	
EnumerationItemDefinition	X		
Environment	X		
ErrorLogging	X		
Exception	X		
Graxis	X		
GrDispAttr	X		
Inet	X		
InternetResult	X		
JaguarOrb			XX
MailFileDescription	X		
MailMessage	X		
MailRecipient	X		
MailSession	X		
Message	X		
NonVisualObject	X		
NullObjectError		X	
OleObject	X		
OleRuntimeError		X	
OleStorage			XX
OleStream			XX
OleTxnObject			X
OmObject		X	
OmStorage			XX
OmStream			XX
Orb			X
PbxRuntimeError		X	
Pipeline			X
ProfileCall			XX
ProfileClass			XX
ProfileLine			XX
ProfileRoutine			XX
Profiling			XX
RemoteObject			XX
ResultSet	X		
ResultSets	X		
RuntimeError		X	

Class name	Supported	Partially supported	Unsupported
ScriptDefinition		X	
Service	X		
SimpleTypeDefinition		X	
SSLCallback			X
SSLServiceProvider			X
Throwable	X		
Timing	X		
TraceActivityNode			XX
TraceBeginEnd			XX
TraceError			XX
TraceESQL			XX
TraceFile			XX
TraceGarbageCollect			XX
TraceTreeLine			XX
TraceTreeNode			XX
TraceTreeObject			XX
TraceTreeRoutine			XX
TraceTreeUser			XX
TraceUser			XX
Transaction	X		
TransactionServer		X	
TypeDefinition		X	
VariableCardinalityDefinition	X		
VariableDefinition		X	
WSCConnection	X		

* The order of the array items in the VariableList property of the ClassDefinition object may not be the same in .NET applications as in standard PowerBuilder applications.

Table 11-2: Unsupported functions, events, and properties by class

Class name	Unsupported functions	Unsupported events	Unsupported properties
Application	SetLibraryList SetTransPool	None	ToolBarUserControl
CorbaSystemException (and its descendants)			Class Line Number

Class name	Unsupported functions	Unsupported events	Unsupported properties
DataStore	CopyRTF GenerateHTMLForm GenerateResultSet GetStateStatus InsertDocument PasteRTF	Destructor	None
DataWindowChild	DBErrorCode DBErrorMessage SetRedraw SetRowFocusIndicator	None	None
OmObject	GetAutomationNativePointer SetAutomationLocale SetAutomationTimeOut	None	None
RuntimeError (and its descendants)			Class Line Number
ScriptDefinition			AliasName ExternalUserFunction (supported for external functions only) LocalVariableList Source SystemFunction
SimpleTypeDefinition			LibraryName
TypeDefinition			LibraryName
VariableDefinition			InitialValue (supported for instance variables and primitive types) IsConstant (supported for instance variables) OverridesAncestorValue ReadAccess (supported for instance variables) WriteAccess (supported for instance variables)

Unsupported system functions in Windows Forms

Table 11-3 lists categories of system functions that are not supported in Windows Forms applications.

Table 11-3: *Unsupported system functions by category*

Category	Functions
DDE functions	CloseChannel, ExecRemote, GetCommandDDE, GetCommandDDEOrigin, GetDataDDE, GetDataDDEOrigin, GetRemote, OpenChannel, RespondRemote, SetDataDDE, SetRemote, StartHotLink, StartServerDDE, StopHotLink, StopServerDDE
Garbage collection functions	GarbageCollectGetTimeLimit, GarbageCollectSetTimeLimit
Input method functions	IMEGetCompositionText, IMEGetMode, IMESetMode
Profiling and tracing functions	TraceBegin, TraceClose, TraceDisableActivity, TraceDump, TraceEnableActivity, TraceEnd, TraceError, TraceOpen, TraceUser
Shared Object functions	SharedObjectDirectory, SharedObjectGet, SharedObjectRegister, SharedObjectUnRegister

Post function Post function calls with reference parameters are not supported.

IsNull function In .NET applications, if you call the IsNull function with a variable of a reference type (a type derived from the PowerObject base class) as the argument, IsNull returns true if the variable has not been initialized by assigning an instantiated object to it. To ensure consistent behavior between standard and .NET PowerBuilder applications, use the IsValid function to check whether the variable has been instantiated. For more information, see the description of the IsNull function.

Partially supported visual controls for Windows Forms

Table 11-4 lists all PowerBuilder visual controls and indicates whether they are fully or partially supported in this release.

For many visual controls, the only unsupported event is the Other event and the only unsupported property is IMEMode. If a control has no other unsupported events, properties, or functions, it is listed in the fully supported column in Table 11-4.

When there is an X in the partially supported column, see Table 11-5 on page 159 for detailed information about which functions, events, and properties are not supported.

Table 11-4: Support for visual controls

Class name	Supported	Partially supported
Animation	X	
Checkbox	X	
CommandButton	X	
DataWindow		X
DatePicker		X
DropDownListBox		X
DropDownPictureListBox	X	
EditMask	X	
Graph		X
GroupBox	X	
HProgressBar	X	
HScrollBar	X	
HTrackBar	X	
InkEdit	X	
InkPicture	X	
Line	X	
ListBox		X
Listview		X
ListViewItem		X
Menu		X
MenuCascade		X
MonthCalendar		X
MultiLineEdit		X
OleControl	X	
OleCustomControl		X

Class name	Supported	Partially supported
OmCustomControl		X
OmEmbeddedControl		X
Oval	X	
Picture	X	
PictureButton		X
PictureHyperLink	X	
PictureListBox	X	
RadioButton	X	
Rectangle	X	
RichTextEdit	X	
RoundRectangle	X	
SingleLineEdit	X	
StaticHyperLink		X
StaticText		X
Tab		X
TreeView		X
TreeViewItem		X
UserObject		X
VProgressBar	X	
VScrollBar	X	
VTrackBar	X	
Window		X

Table 11-5: *Unsupported functions, events, and properties by control*

Supported control	Unsupported functions	Unsupported events	Unsupported properties
DataWindow	DBErrorCode DBErrorMessage GenerateHTMLForm GetStateStatus	Other	RightToLeft
DatePicker	GetCalendar Resize (does not support changing height, otherwise supported)	DoubleClicked Other UserString	AllowEdit Border BorderStyle
DropDownListBox	None	Other	HScrollBar IMEMode

Supported control	Unsupported functions	Unsupported events	Unsupported properties
Graph	None AddData, GetDataValue, InsertData, ModifyData do not support string values	Other	BorderStyle
ListBox	None	Other	TabStop
ListView	SetOverlayPicture AddColumn, InsertColumn, SetColumn limitation: the alignment of the first column cannot be set to center or right	Other	IMEMode
ListViewItem	None	No events	OverlayPictureIndex
Menu	None	Help	MenuItemType MergeOption ToolBarAnimation ToolBarHighlightColor ToolBarItemSpace
MenuCascade	None	Help	Columns CurrentItem DropDown MenuItemType MergeOption ToolBarAnimation ToolBarHighlightColor ToolBarItemSpace
MonthCalendar	None	DoubleClicked Other	
MultiLineEdit	None	Other	IMEMode TabStop
OmCustomControl	None	None	Alignment Cancel Default

Supported control	Unsupported functions	Unsupported events	Unsupported properties
OmEmbeddedControl	_Get_DocFileName _Get_ObjectData _Set_ObjectData Drag InsertClass InsertFile InsertObject LinkTo Open PasteLink PasteSpecial SaveAs SelectObject UpdateLinksDialog	None	Activation ContentsAllowed DisplayType DocFileName LinkUpdateOptions ObjectData ParentStorage Resizable SizeMode
PictureButton	None	Other	Map3DColors
StaticHyperLink	None	Other	BorderColor FillPattern
StaticText	None	Other	BorderColor FillPattern
Tab	None	Other	BackColor RaggedRight (see “Tab properties” on page 165)
TreeView	SetOverlayPicture	Other	IMEMode StatePictureHeight StatePictureWidth
TreeViewItem	None	No events	OverlayPictureIndex
UserObject	AddItem DeleteItem EventParmDouble EventParmString InsertItem	Other	BackColor, TabTextColor (for tab pages—see “Tab properties” on page 165) Style

Supported control	Unsupported functions	Unsupported events	Unsupported properties
Window	CloseChannel ExecRemote GetCommandDDE GetCommandDDEOrigin GetDataDDE GetDataDDEOrigin GetRemote OpenChannel RespondRemote SetDataDDE SetRemote StartHotLink StartServerDDE StopHotLink StopServerDDE	Other	None

Unsupported functions for controls in Windows Forms

Table 11-6 is an alphabetical listing of unsupported functions, the controls on which they are not supported, and any notes that apply to specific controls. If your application uses these functions, rework it to avoid their use.

Table 11-6: Unsupported functions for Windows Forms deployment

Function	Controls
AddItem	UserObject
CloseChannel	Window
DBErrorCode	DataWindow
DBErrorMessage	DataWindow
DeleteItem	UserObject
Drag	OmEmbeddedControl
EventParmDouble	UserObject
EventParmString	UserObject
ExecRemote	Window
GenerateHTMLForm	DataWindow
_Get_DocFileName	OmEmbeddedControl
_Get_ObjectData	OmEmbeddedControl
GetStateStatus	DataWindow
GetCommandDDE	Window

Function	Controls
GetCommandDDEOrigin	Window
GetDataDDE	Window
GetDataDDEOrigin	Window
GetRemote	Window
InsertClass	OmEmbeddedControl
InsertFile	OmEmbeddedControl
InsertItem	UserObject
LinkTo	OmEmbeddedControl
Open	OmEmbeddedControl
OpenChannel	Window
PasteLink	OmEmbeddedControl
PasteSpecial	OmEmbeddedControl
Resize	DatePicker (only changing height is unsupported)
RespondRemote	Window
SaveAs	OmEmbeddedControl
SelectObject	OmEmbeddedControl
SetDataDDE	Window
_Set_ObjectData	OmEmbeddedControl
SetOverlayPicture	ListView, TreeView
SetRemote	Window
SetWSObject	DataWindow
StartHotLink	Window
StartServerDDE	Window
StopHotLink	Window
StopServerDDE	Window
UpdateLinksDialog	OmEmbeddedControl

Unsupported events for controls in Windows Forms

Table 11-7 is an alphabetical listing of unsupported events, the controls on which they are not supported, and any notes that apply to specific controls. If your application uses these events, rework it to avoid their use.

Table 11-7: Unsupported events for Windows Forms deployment

Event	Controls
DoubleClicked	DatePicker, MonthCalendar
Help	Menu, MenuCascade
Notify	TreeView
Other	All controls
Resize	DatePicker
UserString	DatePicker

Unsupported properties for controls in Windows Forms

Table 11-8 is an alphabetical listing of unsupported properties, the controls on which they are not supported, and any notes that apply to specific controls. If your application uses these properties, rework it to avoid their use.

Table 11-8: Unsupported properties for Windows Forms deployment

Property	Controls
Alignment	OmCustomControl
AllowEdit	DatePicker
Activation	OmEmbeddedControl
BackColor	Tab, UserObject (see “Tab properties”)
Border	DatePicker
BorderColor	StaticHyperLink, StaticText
BorderStyle	DatePicker, Graph
Cancel	OmCustomControl
Columns	MenuCascade
ColumnsPerPage	UserObject
ContentsAllowed	OmEmbeddedControl
CurrentItem	MenuCascade
Default	OmCustomControl
DisplayType	OmEmbeddedControl
DocFileName	OmEmbeddedControl
DropDown	MenuCascade
FillPattern	StaticHyperLink, StaticText
Height	DatePicker

Property	Controls
Help	Menu, MenuCascade
HScrollbar	DropDownListBox
IMEMode	All controls
LinkUpdateOptions	OmEmbeddedControl
Map3DColors	PictureButton
MenuItemType	Menu
MergeOption	Menu
ObjectData	OmEmbeddedControl
OverlayPictureIndex	ListViewItem, TreeViewItem
ParentStorage	OmEmbeddedControl
RaggedRight	Tab (see “Tab properties”)
RightToLeft	DataWindow, ListBox, ListView, TreeView
SizeMode	OmEmbeddedControl
StatePictureHeight	TreeView
StatePictureWidth	TreeView
Style	UserObject
TabStop	ListBox, MultiLineEdit
TabTextColor	UserObject (see “Tab properties”)
ToolbarAnimation	Menu
ToolbarHighLightColor	Menu
ToolbarItemSpace	Menu

FaceName property If you use a bitmap (screen) font such as MS Sans Serif instead of a TrueType font for the FaceName property, make sure you select a predefined font size from the TextSize drop-down list. PowerBuilder and .NET use different functions (CreateFontDirect and GdiCreateFont) to render bitmap fonts and they may display larger in the .NET application than in the development environment or a standard PowerBuilder application. For example, text that uses the MS Sans Serif type face and the undefined text size 16 looks the same as size 14 in PowerBuilder, but looks larger in .NET.

Tab properties The RaggedRight property for a Tab control works correctly if the sum of the widths of all the tab pages is greater than the width of the Tab control, and the MultiLine property is set to true.

The BackColor and TabTextColor properties for a tab page in a Tab control are not supported if the XP style is used.

PART 4

.NET Assembly and Web Service Targets

This part describes how to create and deploy PowerBuilder nonvisual objects as .NET assemblies and as Web services.

.NET Assembly and .NET Web Service Targets

About this chapter

PowerBuilder includes targets for creating .NET assemblies and .NET Web service applications from nonvisual custom class objects. This chapter describes how to create .NET Assembly and .NET Web Service targets “from scratch” or from objects in existing PowerBuilder libraries.

Contents

Topic	Page
The .NET Assembly target wizard	169
Modifying a .NET Assembly project	172
Supported datatypes	175
Deploying and running a .NET Assembly project	175
The .NET Web Service target wizard	176
Modifying a .NET Web Service project	178
Configuring ASP.NET for a .NET Web Service project	181
Deploying and running a .NET Web Service project	182

The .NET Assembly target wizard

You can create .NET assembly targets from scratch or by using PBLs from an existing target that contain at least one nonvisual custom class object.

Creating a target from scratch

When you use the .NET Assembly target wizard to create a target from scratch, the wizard also creates an Application object, a project object that allows you to deploy the assembly, and a nonvisual object. However, you must add and implement at least one public method in the wizard-created NVO before it can be used to create a .NET assembly.

For .NET Assembly targets you create from scratch, you must provide the information described in Table 12-1.

Table 12-1: Wizard fields for a .NET Assembly target created from scratch

Wizard field	Description
Project name	Name of the project object the wizard creates.
Library	Name of the library file the wizard creates. By default, this includes the current Workspace path and takes the name you enter for the project object with a PBL extension.
Target	Name of the target the wizard creates. By default, this includes the current Workspace path and takes the name you enter for the project object with a PBT extension.
Library search path	Lets you add PBLs and PBDs to the search path for the new target.
PowerBuilder object name	Name of the nonvisual object the wizard creates. By default this takes the name that you entered for a project object with an "n_" prefix.
Description	Lets you add a description for the project object the wizard creates.
Namespace	Provides a globally unique name to assembly elements and attributes, distinguishing them from elements and attributes of the same name but in different assemblies.
Assembly file name	Name of the assembly created by the wizard. By default, the assembly file name takes the namespace name with a DLL suffix.
Resource file and directory list	<p>List of resource files, or directories containing resource files, that you want to deploy with the project.</p> <p>You can use the Add Files, Add Directories, or Search PBR Files buttons to add files and directories to the list box. You can select a file or directory in the list and click the Delete button to remove that file or directory from the list.</p> <p>When you select a directory, the resource files in all of its subdirectories are also selected by default. However, you can use the Resource Files tab in the Project painter to prevent deployment of subdirectory files. For more information, see "Resource Files and Library Files tabs" on page 173.</p>
Win32 dynamic library file list	Specifies any Win32 DLLs you want to include with your project. Click the Add button to open a file selection dialog box and add a DLL to the list. Select a DLL in the list and click Delete to remove the DLL from the list.
Setup file name	Name of the setup file the wizard creates. You can copy this MSI file to client computers, then double-click the files to install the .NET assembly on those computers.

Creating a target from an existing target

If you select the option to use an existing target, the wizard creates only the .NET Assembly target and a .NET Assembly project. The target you select must include a PBL with at least one nonvisual object having at least one public method. The public method must be implemented by the nonvisual object or inherited from a parent. The `AutoInstantiate` property of the nonvisual object must be set to `false`.

System Tree display

All objects from an existing target display in the System Tree for the .NET Assembly target created from the existing target, except for any project objects that are incompatible with the new target. Although visual objects, as well as the application object, are not used in a .NET Assembly target, you can view them in the System Tree under the new target's PBLs.

When you use the wizard to create a .NET Assembly target from an existing target, the wizard prompts you for the same information as when you create a target from scratch, except that it omits the PowerBuilder object name and library search path fields. These fields are unnecessary because the existing target must have a usable nonvisual object and the library search path for the target is already set. The wizard does, however, present fields that are not available when you create a target from scratch. Table 12-2 describes these additional fields.

Table 12-2: Additional fields for the existing target wizard selection

Wizard field	Description
Choose a target	Select a target from the list of targets in the current workspace.
Specify a project name	Select a name for the project you want to create. You must create a project object to deploy nonvisual objects as .NET components.
Choose a project library	Specify a library from the list of target libraries where you want to store the new project object.
Choose NVO objects to be deployed	Expand the library node or nodes in the list box and select check boxes next to the nonvisual objects that you want to deploy.
Use .NET nullable types	Select this check box to map PowerBuilder standard datatypes to .NET nullable datatypes. Nullable datatypes are not Common Type System (CTS) compliant, but they can be used with .NET Generic classes if a component accepts or returns null arguments or if reference arguments are set to null.

After you create a .NET Assembly target, you can create as many .NET Assembly projects as you need. You start the .NET Assembly project wizard from the Project tab of the New dialog box. The fields in the wizard include all the fields in Table 12-1 except the “PowerBuilder object name” and “Description” fields and all the fields in Table 12-2 except the “Choose a target” field.

Whether you opt to build a new target from scratch or from an existing target, most of the project-related fields listed in Table 12-1 and Table 12-2 are available for modification in the Project painter. For more information, see “Modifying a .NET Assembly project” next.

Modifying a .NET Assembly project

You can modify a .NET Assembly project from the Project painter. In addition to the values for fields that you entered in the target and project wizards, you can also modify version, debug, and run settings from the Project painter. The Objects tab of the Project painter lets you select and rename functions of the nonvisual objects you deploy to a .NET assembly.

Each .NET Assembly project has seven tab pages: General, Objects, Resource Files, Library Files, Version, Post-build, and Run.

General tab

The General tab in the Project painter allows you to modify the namespace, assembly file name, and setup file name for a .NET Assembly project. It also has a check box you can select to use .NET nullable datatypes. These fields are described in Table 12-1 or Table 12-2.

In addition, the General tab has fields that are not available in the target or project wizards. Table 12-3 describes these additional fields.

Table 12-3: Additional fields available on the General tab

Project field	Description
Debug or Release	Radio button options that determine whether the project is deployed as a debug build (default selection) or a release build. You use debug builds for debugging purposes. Release builds have better performance, but when you debug a release build, the debugger does not stop at breakpoints.
Enable DEBUG symbol	Select to activate code inside conditional compilation blocks using the DEBUG symbol. This selection does not affect and is not affected by the project’s debug build or release build setting. This check box is selected by default.

Objects tab

The Objects tab in the Project painter lists all the nonvisual user objects available for deployment from the current .NET Assembly target. The Custom Class field lists all these objects even if you did not select them in the target or project wizard.

Objects that you selected in the wizard display with a user object icon in the Custom Class treeview. All methods for the objects selected in the wizard are also selected for deployment by default, but you can use the Objects tab to prevent deployment of some of these methods and to change the method names in the deployed component.

Table 12-4 describes all the fields available on the Objects tab.

Table 12-4: Fields available on the Objects tab

Project field	Description
Custom class	Select an object in this treeview list to edit its list of functions for inclusion in or exclusion from the assembly component. You can edit the list for all the objects you want to include in the assembly, but you must do this one object at a time.
Object name, Class name, and Namespace	You can change the object name only by selecting a different object in the Custom Class treeview. By default, the class name is the same as the object name, but it is editable. In the Project painter, the namespace is editable only on the General tab.
Method names and Function prototypes	Select the check box for each function of the selected custom class object you want to deploy to a .NET assembly. Clear the check box for each function you do not want to deploy. You can modify the method names in the Method Names column. The Function Prototype column is for descriptive purposes only.
Change method name and description	You enable these buttons by selecting a method in the list of method names. PowerBuilder allows overloaded functions, but each function you deploy in an assembly class must have a unique name. After you click the Change Method Name button, you can edit the selected method name in the Method Name column. The Change Method Description button lets you add or edit a method description.
Select All and Unselect All	Click the Select All button to select all the functions of the current custom class object for deployment. Click the Unselect All button to clear the check boxes of all functions of the current custom class object. Functions with unselected check boxes are not deployed to a .NET assembly.

Resource Files and Library Files tabs

The fields that you can edit on the Resource Files and Library Files tabs of the Project painter are the same as the fields available in the target and project wizards. These fields are described in Table 12-1.

The Resource Files page of the Project painter does have an additional field that is not included in the project or target wizard. The additional field is a Recursive check box next to each directory that you add to the Resource Files list. By default, this check box is selected for each directory when you add it to the list, but you can clear the check box to avoid deployment of unnecessary subdirectory files.

Version, Post-build,
and Run tabs

The fields on the Version, Post-build, and Run tabs of the Project painter are not available in the .NET Assembly target or project wizards. Table 12-5 describes these fields.

Table 12-5: Fields available on the Version, Post-build, and Run tabs

Project field	Description
Product name, Company, Description, and Copyright (Version tab)	You can specify identification, description, and copyright information that you want to associate with the assembly you generate for the project.
Product version, File version, and Assembly (Version tab)	You can enter major, minor, build, and revision version numbers for the product, file, and assembly.
Post-build command line list (Post-build tab)	You can use the Add button to include command lines that run immediately after you deploy the project. For example, you can include a command line to process the generated component in a code obfuscator program, keeping the component safe from reverse engineering. The command lines run in the order listed, from top to bottom. You can save separate sequences of command lines for debug and release build types.
Application (Run tab)	You use this text box to enter the name of an application with code that invokes the classes and methods of the generated assembly. If you do not enter an application name, you get an error message when you try to run or debug the deployed project from the PowerBuilder IDE.
Argument (Run tab)	You use this text box to enter any parameters for an application that invokes the classes and methods of the deployed project.
Start In (Run tab)	You use this text box to enter the starting directory for an application that invokes the classes and methods of the deployed project.

Supported datatypes

The PowerBuilder to .NET compiler converts PowerScript datatypes to .NET datatypes. Table 12-6 shows the datatype mapping between PowerScript and C#. Arrays are also supported for all standard datatypes.

Table 12-6: Datatype mapping between PowerScript and C#

PowerScript datatype	C# datatype
boolean	bool
blob	byte []
byte	byte
int, uint	short, ushort
long, ulong	int, uint
longlong	long
decimal	decimal
real	float
double	double
string	string
user-defined structure	struct
user-defined nonvisual object	class
Date	DateTime
Time	DateTime
DateTime	DateTime

Deploying and running a .NET Assembly project

After you create a .NET Assembly project, you can deploy it from the Project painter or from a pop-up menu on the project object in the System Tree.

When you deploy a .NET Assembly project, PowerBuilder creates an assembly DLL from the nonvisual user objects you selected in the wizard or project painter. If you also listed a setup file name, PowerBuilder creates an MSI file that includes the assembly DLL, PowerBuilder system libraries for .NET, and any resource files you listed in the wizard or project painter.

Deploying required PowerBuilder files

You can use the Runtime Packager to copy required PowerBuilder runtime files to deployment machines.

For information on required runtime files and the Runtime Packager, see “Deploying Applications and Components” in *Application Techniques*.

You can run or debug an assembly project from the PowerBuilder UI if you fill in the Application field (and optionally, the Argument and Start In fields) on the project Run tab in the Project painter.

For more information about debugging .NET targets, including .NET assembly components, see Chapter 15, “Debugging and Troubleshooting.”

The .NET Web Service target wizard

Creating a target from scratch

The .NET Web Service target wizard gives you the option of creating a target from scratch or from an existing PowerBuilder target.

The .NET Web Service target wizard shares the following fields in common with the .NET Assembly target: Project Name, Target Name, Library, Library Search Path, PowerBuilder Object Name, Description, Resource Files, and Win32 Dynamic DLLs. (Table 12-1 provides descriptions for these fields.) It does not include the Namespace and Assembly File Name fields, which are specific to the .NET Assembly wizard. Although it does include a Setup File Name field, in the .NET Web Service target wizard, this field is part of a deployment option selection.

Table 12-7 shows wizard fields that are unique to the .NET Web Service target with the “from scratch” option selected.

Table 12-7: Wizard fields specific to .NET Web Service targets and projects

Wizard field	Description
Web service virtual directory name	The directory path you want to use as the current directory in the virtual file system on the server. By default, this is the full path name for the current PowerBuilder target. This field is similar to the “Initial current directory” field in the Web Forms wizard.
Web service URL preview	Address for accessing the .NET Web service from an application.
Generate setup file	Select this option to deploy the Web service in an MSI file. When you select this option, you must provide a name for the setup file.
Directly deploy to IIS	Select this option to deploy the Web service directly to an IIS server. When you select this option, you must provide an IIS server address. By default, the server address is “localhost”.

When you click Finish in the wizard after selecting the option to create a target from scratch, the wizard generates an Application object, a project object, a target, and a nonvisual object. You must add and implement a public method in the nonvisual object generated by the wizard before you can deploy it as a Web service.

Creating a target from an existing target

As with the other .NET target wizards (.NET Web Forms, .NET Windows Forms, and .NET Assembly), you can use the .NET Web Service target wizard to a target from an existing PowerBuilder target. The existing target must be added to the current workspace and must include a PBL with at least one nonvisual object having at least one public method. The public method must be implemented by the nonvisual object or inherited from a parent. The AutoInstantiate property of the nonvisual object must be set to false.

When you click Finish in the .NET Web Service target wizard, the wizard creates a .NET Web Service target and a .NET Web Service project. The .NET Web Service target uses the same library list as the existing target from which you select nonvisual user objects.

As with the .NET Assembly target wizard, the .NET Web Service target wizard has additional fields for selecting nonvisual user objects when you use the existing target option. Table 12-8 lists these additional fields.

Table 12-8: Additional fields for the existing target wizard selection

Wizard field	Description
Choose a target	Select a target from the list of targets in the current workspace.
Specify a project name	Select a name for the project you want to create. You must create a project object to deploy nonvisual objects as .NET components.
Choose a project library	Specify a library from the list of target libraries where you want to store the new project object.
Choose NVO objects to be deployed	Expand the library node or nodes in the list box and select check boxes next to the nonvisual objects that you want to deploy.
Use .NET nullable types	Select this check box to map PowerBuilder standard datatypes to .NET nullable datatypes. Nullable datatypes are not Common Type System (CTS) compliant, but they can be used with .NET Generic classes if a component accepts or returns null arguments or if reference arguments are set to null.

Modifying a .NET Web Service project

You can modify a .NET Web Service project from the Project painter. The Project painter displays all the values you selected in .NET Web Service target or project wizards. However, you can also modify version, debug, and run settings from the Project painter. The Objects tab of the Project painter lets you select and rename functions of the nonvisual objects that you deploy to a .NET Web Service component.

.NET Web Service project tab pages

Each .NET Web Service project has eight tab pages: General, Deploy, Objects, Resource Files, Library Files, Version, Post-build, and Run.

General and Deploy tabs See Table 12-3 for a description of the debug fields available on the General tab of the Project painter. The fields on the Deploy tab are all available in the .NET Web Service project wizard. For descriptions of fields available on the Deploy tab, see Table 12-7.

Resource Files tab The Resource Files tab fields in the Project painter are the same as those in the project wizard. However, as for the .NET Assembly project, there is one additional field that is not included in the project or target wizard. This field is a Recursive check box next to each directory you add to the Resource Files list. By default, this check box is selected for each directory when you add it to the list, but you can clear the check box to avoid deployment of unnecessary subdirectory files.

Library Files tab The Library Files tab of the Project painter includes fields for the Win 32 dynamic libraries you want to deploy with your project. These fields are described in Table 12-1. However, the Project painter tab also includes a list of PBL files for the target. A check box next to each of these PBL files indicates that they deploy as PBD files with your project.

You can clear the check box next to each PBL that you do not want to deploy with your project. You need to deploy only PBLs containing DataWindow or Query objects used by the custom class objects you include in a Web service component. You can use the Select All button to select all the check boxes for the PBLs or the Unselect All button to clear all the check boxes.

Objects tab The Objects tab allows you to select the methods you want to make available for each nonvisual object you deploy as a Web service. You can rename the methods as Web service messages. Table 12-9 describes the the Objects tab fields for a .NET Web Service project.

Table 12-9: Fields on the .NET Web Service Objects tab

Objects tab field	Description
Custom class	Select an object in this treeview list to edit its list of methods for inclusion in or exclusion from the Web service component. You can edit the list for all the objects you want to include in the component, but you must do this for one object at a time.
Object name	You can change the object name only by selecting a different object in the Custom Class treeview.
Web service name	Specifies the name for the Web service. By default, this takes the name of the current custom class user object.
Target namespace	Specifies the target namespace. The default namespace for an IIS Web service is: <code>http://tempurl.org</code> . Typically you change this to a company domain name.
Web service URL	Specifies the deployment location for the current custom class user object. This is a read-only field. The location combines selections on the General, Deploy, and Objects tabs for the current project.

Objects tab field	Description
Web service WSDL	Specifies the WSDL file created for the project. This is a read-only field. It appends the “?WSDL” suffix to the Web service URL.
Browse Web Service	If you have previously deployed the project to the named IIS server on the Deploy tab of the current project, you can click this button to display a test page for the existing Web service. If a Web service has not been deployed yet for the current custom class object, a browser error message displays. The button is disabled if you selected the option to deploy the current project to a setup file.
View WSDL	If you previously deployed the project to the named IIS server on the Deploy tab of the current project, you can click this button to display the existing WSDL file. If a Web service has not been deployed yet for the current custom class object, a browser error message displays. The button is disabled if you selected the option to deploy the current project to a setup file.
Message names and Function prototypes	Select the check box for each function of the selected custom class object that you want to deploy in a .NET Web service component. Clear the check box for each function you do not want to deploy. You can modify the message names in the Message Names column. The Function Prototype column is for descriptive purposes only.
Change message name	You enable this button by selecting a function in the list of message names. PowerBuilder allows overloaded functions, but each function you deploy in a component class must have a unique name. After you click the Change Message Name button, you can edit the selected function name in the Message Name column.
Select All and Unselect All	Click the Select All button to select all the functions of the current custom class object for deployment. Click the Unselect All button to clear the check boxes of all functions of the current custom class object. Functions with unselected check boxes are not deployed as messages for a Web service component.

Version, Post-build, and Run tabs See Table 12-5 for a description of the version, post-build, and run settings fields. These are the same fields that are available with .NET Assembly projects. They cannot be set in the target or project wizards.

The Run tab settings for a Web Service project typically have default values for the Application and Arguments fields. The Application field default is the path to the Internet Explorer browser on the development computer and the Arguments field default is the URL for a Web Service project test page that is created when you deploy the project.

Configuring ASP.NET for a .NET Web Service project

IIS and ASP.NET You configure ASP.NET for .NET Web Service projects the same way you configure ASP.NET for .NET Web Forms projects. This includes making sure the Web server has a compatible version of IIS and the 2.0 version of ASP.NET is selected for your Web service components. .NET Web Service projects also use the same directory structure on the server as .NET Web Forms projects.

For information on installing IIS and setting the default version of ASP.NET, see “Configuring ASP.NET for a .NET Web project” on page 6. For information on the directory structure for deployed projects, see “Directory structure on the server” on page 9.

SQL Anywhere database connection If you set up a database connection for your Web service components, you configure the connection in the same way as for a Web Forms application. For information on configuring a SQL Anywhere database connection, see “Setting up a SQL Anywhere database connection” on page 10.

Global properties Most of the global properties for Web Forms applications that do not involve visual controls also apply to Web service components. The following global properties can be used by Web service projects: LogFolder, FileFolder, PrintFolder, PBWebFileProcessMode, PBCurrentDir, PBTempDir, PBLibDir, PBDenyDownloadFolders, PBCommandParm, PBTrace, PBTraceTarget, PBTraceFileName, PBMaxSession, PBEventLogID, and PBDeleteTempFileInterval.

For information on viewing and modifying global properties, see “Viewing and modifying global properties in IIS Manager” on page 8. For descriptions of the global properties, see “Global Web configuration properties” on page 74.

Deploying and running a .NET Web Service project

After you create a .NET Web Service project, you can deploy it from the Project painter or from a pop-up menu on the project object in the System Tree.

When you deploy directly to an IIS server, PowerBuilder creates an application directory under the IIS virtual root and creates an ASMX file in the application directory. The ASMX file created by the project is an ASP.NET executable file rather than a true WSDL file, so you might need to add the “?WSDL” suffix to the URL when you try to access this Web service from certain types of applications.

In addition to the application directory and the ASMX file, deploying the project creates a directory structure that is substantially the same as that created by a .NET Web Forms project. For more information on the directory structure, see “Directory structure on the server” on page 9.

Access permissions for Web service components

In some versions of IIS for the Windows XP platform, ASP.NET Web services use the Temp system directory during method processing. If the ASP.NET user (IIS 5), the IIS_WPG user group (IIS 6), or the IIS_USRS user group (IIS 7) does not have read or write access to the Temp directory on the server, applications invoking methods on those services receive an error message stating that temporary classes cannot be generated. You can prevent this error by granting appropriate user or user group permissions to the Temp directory in the same way you grant permissions for the Sybase and database directories.

For more information on granting ASP.NET user permissions, see “Setting up a SQL Anywhere database connection” on page 10.

When you deploy to a setup file in a .NET Web Service project, the project builds an MSI file that includes the ASMX file, PowerBuilder system libraries for .NET, and any resource files you listed in the project wizard or painter.

Deploying required PowerBuilder files

You can use the Runtime Packager to copy required PowerBuilder runtime files to deployment servers. After you install the package created by the runtime packager, you must restart the server.

For information on required runtime files and the Runtime Packager, see “Deploying Applications and Components” in *Application Techniques*.

You can run or debug a .NET Web Service project from the PowerBuilder UI if you fill in the Application field (and optionally, the Argument and Start In fields) on the project Run tab in the Project painter. The Application field is typically filled in automatically with the name of the Internet Explorer executable on the development machine.

For more information about debugging .NET targets, including .NET Web Service components, see Chapter 15, “Debugging and Troubleshooting.”

.NET Language Interoperability

This part describes how to use conditional compilation blocks in PowerScript code. These coding blocks allow you to reference .NET objects and methods in PowerScript without triggering error messages from the PowerScript compiler.

It also includes suggestions for enhancing the .NET applications you build in PowerBuilder, information about debugging .NET targets, and troubleshooting tips.

Referencing .NET Classes in PowerScript

About this chapter

This chapter describes the special syntax you can use to add processing code for PowerBuilder .NET projects.

Contents

Topic	Page
About conditional compilation	187
Writing code inside a .NET block	190
PowerScript syntax for .NET calls	191
Calling assembly methods from PowerScript	193
Support for .NET language features	194
Handling exceptions in the .NET environment	198

About conditional compilation

In PowerBuilder 11.0, you can use the number sign (#) at the start of a line or block of code to mark the code for specialized processing prior to PowerScript compilation. Each line of code or block of conditional code set off by a leading number sign is automatically parsed by a PowerBuilder preprocessor before it is passed to the design-time PowerScript compiler or the PowerBuilder-to-C# (pb2cs) compiler.

Preprocessing symbols

There are six default code-processing symbols that affect the code passed to the PowerScript compiler at design time. Four of these symbols correspond to different PowerBuilder target types, one applies to all .NET target types, and one applies to both PowerScript and .NET target types.

The preprocessor enables PowerBuilder to compile project code specific to a particular deployment target without hindering the compiler's ability to handle the same code when a different deployment target is selected.

The preprocessor substitutes blank lines for all declarative statements and conditional block delimiters having leading number sign characters before passing the code to the PowerScript compiler or the pb2cs compiler. The contents of the conditional blocks are converted to blank lines depending on which compiler is called upon to process the code handed off by the preprocessor.

Table 13-1 displays the default preprocessing symbols, the project types to which they correspond, and their effects on the code passed to the PowerScript compiler engine or the pb2cs compiler.

Table 13-1: Default preprocessing symbols for conditional compilation

Preprocessing symbols	Project type	Code in this processing block
PBNATIVE	PowerBuilder client-server or distributed applications	Fully parsed by the PowerScript compiler. It is converted to blank lines for the pb2cs compiler.
PBWEBFORM	.NET Web Forms application	Fully parsed by the pb2cs compiler for .NET Web Forms targets only. It is converted to blank lines for the PowerScript compiler and all other types of .NET targets.
PBWIFORM	.NET Windows Forms applications	Fully parsed by the pb2cs compiler for .NET Windows Forms targets only. It is converted to blank lines for the PowerScript compiler and all other types of .NET targets.
PBWEBSERVICE	.NET Web Service component targets	Fully parsed by the pb2cs compiler for .NET Web Service targets only. It is converted to blank lines for the PowerScript compiler and all other types of .NET targets.
PBDOTNET	.NET Web Forms and Windows Forms applications, and .NET Assembly and .NET Web Service components	Fully parsed by the pb2cs compiler for all .NET target types. It is converted to blank lines for the PowerScript compiler.

Preprocessing symbols	Project type	Code in this processing block
DEBUG	PowerScript targets and all .NET application and component targets	When a project's Enable DEBUG Symbol check box is selected, code is fully parsed in deployed applications by the PowerScript compiler, or for .NET targets, by the pb2cs compiler. Code is converted to blank lines when the check box is cleared.

Conditional syntax

You indicate a conditional block of code with a statement of the following type, where *symbolType* is any of the symbols defined by PowerBuilder:

```
#IF defined symbolType then
```

You can use the NOT operator to include code for all target types that are not of the symbol type that you designate. For example, the following code is parsed for all targets that are not of the type PBNative:

```
#IF NOT defined PBNATIVE then
```

You can also use #ELSE statements inside the code block to include code for all target types other than the one defined at the start of the code block. You can use #ELSEIF defined *symbolType* then statements to include code for a specific target type that is different from the one defined at the start of the code block.

The closing statement for a conditional block is always:

```
#END IF
```

Comments can be added to conditional code if they are preceded by double slash marks (//) in the same line of code. Although you cannot use the PowerScript line continuation character (&) in a conditional code statement, you must use it in code that you embed in the conditional block when you use more than one line for a single line of code.

Limitations and error messages

Conditional compilation is not supported in DataWindow syntax, in structure or menu objects, or in JSP targets. You cannot edit the source code for an object to include conditional compilation blocks that span function, event, or variable definition boundaries.

You must rebuild your application after you add a DEBUG conditional block.

Table 13-2 shows the types of error messages displayed for incorrect conditional compilation code.

Table 13-2: Types of error messages returned by the preprocessor

Error message	Description
Invalid if statement	#if statement without a defined symbol, with an incorrectly defined symbol, or without a then clause
#end if directive expected	#if statement without an #end if statement
Unexpected preprocessor directive	Caused by an #else, #elseif, or #end if statement when not preceded by an #if statement
Preprocessor syntax error	Caused by including text after an #else or #end if statement when the text is not preceded by comment characters (//)

Writing code inside a .NET block

Because the main PowerBuilder compiler does not recognize the classes imported from .NET assemblies, you must surround the code referencing those classes in a conditional compilation block for a .NET application. For example, to reference the .NET message box Show function, you must surround the function call with preprocessor statements that hide the code from the main PowerBuilder compiler:

```
#IF Defined PBDOTNET Then
    System.Windows.Forms.MessageBox.Show ("This "&
        + "message box is from .NET, not "&
        + "PowerBuilder Native.")
#END IF
```

The PBDOTNET symbol can be used for all types of .NET targets supported by PowerBuilder. You can also use the following symbols for specific types of .NET targets: PBWEBFORM, PBWINFORM, and PBWEBSERVICE.

For more information on conditional compilation symbols, see “About conditional compilation” on page 187.

PowerScript syntax for .NET calls

When you make calls to .NET assemblies or their methods or properties from PowerBuilder, you must follow PowerScript syntax rules. The following syntax rules are especially important for C# developers to keep in mind:

Instantiating a class To instantiate a class, use “create”, not “new”. Even when you are referencing a .NET type in a .NET conditional block, you must use the PowerScript create syntax. The following line instantiates a .NET type from the logger namespace:

```
ls = create logger.LogServer
```

Note that a single dot (.) is used as a namespace separator in .NET conditional blocks.

Compound statements You must use PowerScript syntax for compound statements, such as “if”, “for”, or “switch”. The preprocessors for .NET applications signal an error if C# compound statements are used. For example, you cannot use the following C# statement, even inside a .NET conditional block: `for (int I=0; I<10; I++)`. The following script shows the PowerScript equivalent, with looping calls to the .NET WriteLine method, inside a PBDOTNET conditional block:

```
#IF Defined PBDOTNET THEN
    int i
    for I = 1 to 10
        System.Console.WriteLine(i)
    next
#END IF
```

PowerScript keywords The .NET Framework uses certain PowerBuilder keywords such as “System” and “type”. To distinguish the .NET Framework usage from the PowerBuilder keyword, you can prepend the @ symbol. For example, you can instantiate a class in the .NET System namespace as follows:

```
#IF Defined PBDOTNET THEN
    @System.Collections.ArrayList myList = create &
        + @System.Collections.ArrayList
#END IF
```

The PowerBuilder preprocessor includes logic to distinguish the .NET System namespace from the PowerBuilder System keyword, therefore the use of the @ prefix is optional as a namespace identifier in the above example. However, you must append the @ identifier if you reference the .NET type class in PowerScript code or if you use other PowerBuilder keywords as namespace names.

Line continuation and termination You must use PowerScript rules when your script extends beyond a single line. The line return character indicates the end of a line of script except when it is preceded by the ampersand (&) character. Semicolons are not used to indicate the end of a PowerScript line.

Rules for arrays To declare an array, use square brackets after the variable name, not after the array datatype. You cannot initialize an array before making array index assignments. PowerBuilder provides automatic support for negative index identifiers. (In C#, you can have negative index identifiers only if you use the `System.Array.CreateInstance` method.) The following example illustrates PowerScript coding for an array that can hold seven index values. The code is included inside a conditional compilation block for the .NET environment:

```
#IF Defined PBDOTNET THEN
    int myArray[-2 to 5]
    //in C#, you would have to initialize array
    //with code like: int[] myArray = new int[7]
    myArray[-1]=10 //assigning a value to 2nd array index
#END IF
```

In PowerBuilder, unbounded arrays can have one dimension only. The default start index for all PowerBuilder arrays is 1. The `GetValue` method on a C# array returns 0 for a default start index identifier, so you would call `array_foo.GetValue (0)` to return the first element of the array `array_foo`. However, after a C# array is assigned to a PowerBuilder array, you access the elements of the array with the PowerBuilder index identifier. In this example, you identify the first element in PowerScript as `array_foo[1]`.

Case sensitivity .NET is case sensitive, but PowerBuilder is not. The .NET Framework does provide a way to treat lowercase and uppercase letters as equivalent, and the PowerBuilder to .NET compiler takes advantage of this feature. However, if the .NET resources you are accessing have or contain names that differ only by the case of their constituent characters, PowerBuilder cannot correctly compile .NET code for these resources.

Cross-language data exchange Code inside a .NET conditional compilation block is not visible to the main PowerBuilder compiler. If you use variables to hold data from the .NET environment that you want to access from outside the conditional block, you must declare the variables outside the conditional block. Variables you declare outside a .NET conditional block can remain in scope both inside and outside the conditional block.

Declaring enumeration constants You use a terminal exclamation mark (!) to access enumeration constants in PowerShell. For more information about using enumeration constants in the .NET environment, see “User-defined enumerations” on page 196.

Calling assembly methods from PowerShell

When you call methods from managed assemblies in PowerShell, you must use PowerShell datatypes in any method arguments or return values. Table 13-3 shows the mappings between .NET, C#, and PowerShell datatypes.

Table 13-3: Datatype mappings in managed assembly methods

.NET datatype	C# datatype	PowerShell datatype
System.Boolean	boolean	Boolean
System.Byte	Byte	Byte
System.Sbyte	Sbyte	Sbyte
System.Int16	short	Int
System.UInt16	ushort	UInt
System.Int32	int	Long
System.UInt32	uint	Ulong
System.Int64	long	Longlong
System.UInt64	ulong	Unsignedlonglong
System.Single	float	Real
System.Double	Double	Double
System.Decimal	Decimal	Decimal
System.Char	Char	Char
System.String	String	String
System.DateTime	System.Datetime	Datetime

For example, suppose you want to reference a method `foo` with arguments that require separate `int` and `long` datatype values when you call the method in C# script. The class containing this method is defined in an assembly in the following manner:

```
public class MyClass
{
    public int foo(int a, long b);
    {
        return a + b
    }
}
```

```
}  
}
```

In PowerScript code, you must replace the foo method datatypes with their PowerBuilder datatype equivalents (long for int, longlong for long):

```
long p1, returnValue  
longlong p2  
#IF Defined PBWINFORM Then  
    MyClass instanceOfMyClass  
    instanceOfMyClass = create MyClass  
    returnValue = instanceOfMyClass.foo(p1, p2)  
#END IF
```

Unidirectional cross-language calls

Although you can call .NET methods in PowerScript, you cannot currently call PowerBuilder functions from .NET methods. Some PowerBuilder datatypes such as Blob, Date, and Time do not have equivalent datatypes in the C# language. Because you cannot make calls to PowerBuilder functions, there is no need to map these PowerBuilder datatypes to C# datatypes.

Support for .NET language features

When you use conditional blocks of code for the .NET environment, you can take advantage of the following features that are not available in the standard PowerBuilder application environment:

- **Support for sbyte and ulonglong** Sbyte is the signed format of the byte datatype and ulonglong is the unsigned format of the longlong datatype.
- **Bitwise operators** See “Bitwise operator support” next.
- **Parameterized constructors** Arguments are not permitted in constructors for standard PowerBuilder applications, but they are supported in conditional code blocks for the .NET environment.
- **Static fields and methods** Static fields and methods are not permitted in standard PowerBuilder applications, but they are supported in conditional code blocks for the .NET environment.

- **Namespaces, interfaces, and user-defined enumerations** You can reference namespaces and .NET interfaces and enumerations in conditional code blocks for the .NET environment. Namespaces are not available and you cannot declare an interface or enumeration in standard PowerScript code.

For more information on .NET enumerations, see “User-defined enumerations” on page 196.

- **.NET index access** See “Accessing indexes for .NET classes” on page 198.

Bitwise operator support

Standard PowerBuilder applications allow the use of the logical operators AND, OR, and NOT to evaluate boolean expressions. In .NET applications and components, in addition to evaluating boolean expressions, you can use these same operators to perform bitwise evaluations. For the AND and OR operators, a bitwise evaluation compares the bits of one operand with the bits of a second operand. For the NOT operator, a bitwise evaluation assigns the complementary bit of the single operand to a result bit.

The operands in a bitwise comparison must have integral data types, such as integer, uint, long, ulong, and longlong. However, if either of the operands (or the sole operand in the case of a NOT operation) has an any datatype, the .NET application or component treats the operation as a standard logical evaluation rather than as a bitwise comparison.

You can perform a bitwise comparison only inside a .NET conditional compilation block. If you try to evaluate operands with integral datatypes in a standard PowerBuilder application, you will get a compiler error.

For .NET applications and components, you can also use the bitwise operator XOR. If you use this operator to evaluate a boolean expression in the .NET environment, the return result is true only when one of the operands is true and the other is false. If both operands are true, or both are false, the return result for the XOR operator is false.

Table 13-4 describes the result of using the bitwise operators.

Table 13-4: Bitwise operators in the .NET environment

Operator	Description
AND	The bitwise “AND” operator compares each bit of its first operand to the corresponding bit of its second operand. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
OR	The bitwise “inclusive OR” operator compares each bit of its first operand to the corresponding bit of its second operand. If either bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
XOR	The bitwise “exclusive OR” operator compares each bit of its first operand to the corresponding bit of its second operand. If one bit is 0 and the other bit is 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
NOT	This is a unary operator. It produces the bitwise complement of its sole operand. If one bit is 1, the corresponding result bit is set to 0. Otherwise, the corresponding result bit is set to 1.

User-defined enumerations

Declaring .NET enumerations in PowerShell

To use enumerations that you import from a .NET assembly, you must surround the enumeration references in a conditional compilation block that is valid for your .NET target environment. You must also append an exclamation mark (“!”) to each of the enumeration’s constant strings that you declare in the conditional code block.

For example, the following code defines the .NET enumeration class `TimeOfDay`:

```
Public enum TimeOfDay
{
    Morning = 0,
    AfterNoon,
    Evening
}
```

In PowerShell, you reference a .NET enumeration constant string as follows, when `TimeOfDay` is an enumeration class in the `ns_1.ns_2` namespace:

```
#if defined PBDOTNET THEN
    ns_1.ns_2.TimeOfDay a
    a=ns_1.ns_2.TimeOfDay.Morning!
#end if
```

Scope of enumeration constant

When you set a system-defined enumeration constant in standard PowerBuilder applications, there is no issue regarding the scope of the constant definition, since all system enumeration constants are uniquely defined. However, for .NET enumerations, you must define a scope for the constant using the syntax:

```
enumerationType.enumerationEntryName!
```

If the enumeration class is declared under a namespace, you must include the namespace when you set an enumeration constant:

```
namespaceName.enumerationType.enumerationEntryName!
```

If there is no *enumerationType* enumeration class prefacing the declaration of a constant in a .NET conditional code block, PowerBuilder assumes the enumeration is a system-defined type and returns an error if the system-defined type is not found.

The syntax for a PowerBuilder system enumeration constant in the .NET environment is:

```
[enumerationType.] enumerationEntryName!
```

Although you cannot use dot notation in a constant declaration for a system-defined enumeration in standard PowerScript, the pb2cs compiler must let you use dot notation for constant declarations that you make in a conditional compilation block for the .NET environment. Prefixing a constant declaration in the .NET environment with a PowerBuilder system enumeration name is equivalent to making the same declaration without a prefix.

The VM initially checks whether the *enumerationType* is a declared .NET enumeration class. If it does not find the enumeration class, it checks whether the *enumerationType* is a PowerBuilder system enumeration. When the *enumerationType* matches the name of a PowerBuilder system enumeration, the VM sets the constant for your .NET application or component.

Therefore, for the system Alignment enumeration, the constant declaration `Alignment.Left!` produces the same result as the `Left!` declaration inside a .NET conditional code block. Outside such a code block, the `Alignment.Left!` declaration causes a compiler error.

Accessing indexes for .NET classes

You can access the indexes of .NET classes in the same way you access PowerBuilder array elements. However, in standard PowerBuilder applications, you can reference indexes only using integral datatypes, such as integer, short, long, and so on. In the .NET environment, you are not restricted to referencing indexes as integral types; you can reference the indexes using any datatypes as parameters.

The following example shows how to use a string datatype to access the index of the .NET hashtable class, countries:

```
#IF Defined PBDOTNET then
system.collections.hashtable countries
countries = create system.collections.hashtable
//Assign value to hashtable
countries["Singapore"] = 6
countries["China"] = 1300
countries["United States"] = 200
//Obtain value from hashtable
int singaporePopulation, USAPopulation
singaporePopulation = countries["Singapore"]
USAPopulation = countries["United States"]
#END IF
```

Handling exceptions in the .NET environment

Modified exception hierarchy

The PowerBuilder to .NET compiler changes the exception hierarchy used by the native PowerScript compiler. In the native PowerBuilder environment, Throwable is the root datatype for all user-defined exception and system error types. Two other system object types, RuntimeError and Exception, inherit directly from Throwable.

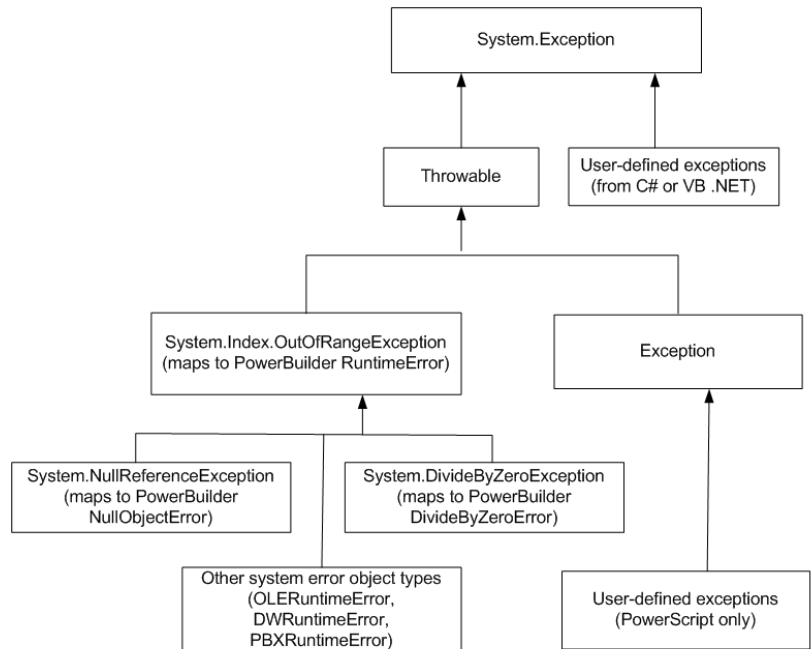
In the .NET environment, System.Exception is the root datatype. The PowerBuilder to .NET compiler redefines the Throwable object type as a subtype of the System.Exception class, and maps the .NET System.IndexOutOfRangeException class to the PowerBuilder RuntimeError object type with the error message "Array boundary exceeded." The PowerBuilder to .NET compiler also maps the following .NET exceptions to PowerBuilder error objects:

- System.NullReferenceException class to the NullObjectError object type

- System.DivideByZeroException class to the DivideByZeroError object type

Figure 13-1 shows the exception hierarchy for PowerBuilder applications in the .NET environment.

Figure 13-1: Exception hierarchy for PowerBuilder in the .NET environment



Example using a .NET system exception class

Even though a .NET exception class is mapped to a PowerBuilder object type, you must use the PowerBuilder object type in your PowerScript code. For example, suppose you define a .NET test class to test for division by zero errors as follows:

```

public class Test
{
    public int division_test (int a)
    {
        return a/0;
        //pops a System.DivideByZero exception
    }
}
  
```

To catch the error in PowerScript, you can use the `DivideByZeroError` object type or either of its ancestors, `RuntimeError` or `Throwable`. The following PowerScript code catches the error caused by the call to the `.NET Test` class method for invoking division by zero errors:

```
int i = 10
string ls_error
try
    #IF Defined PBDOTNET Then
        Test t = create Test
        i = t.division_test(i)
    #END IF
catch (DivideByZeroError e)
//the following lines would also work:
//catch (RuntimeError e)
//catch (Throwable e)
    ls_error = e.getMessage ( )
end try
```

Example using a custom .NET exception class

Suppose the `.NET Test` class is modified to catch a custom `.NET` exception:

```
public class Test
{
    public int second_test (int a)
    {
        a = a/2;
        throw new MyUserException();
    }
}
```

Because `MyUserException` is a user-defined exception in the `.NET` environment, it cannot be caught by either the `PowerBuilder Exception` or `Throwable` object types. It must be handled inside a `.NET` conditional compilation block:

```
int i = 10
string ls_error
#IF Defined PBDOTNET Then
    try
        Test t = create Test
        i = t.second_test
        catch (MyUserException e)
//this will also work: catch (System.Exception e)
            ls_error = e.getMessage()
        end try
    #END IF
```

About this chapter

This chapter provides tips on design and coding for the .NET environment.

Contents

Topic	Page
Coding restrictions	201
Design-level considerations	204
Take advantage of global configuration properties	207
Use client-side events to delay postbacks	209

The changes required to transform a PowerBuilder application into a .NET application depend on the nature of the application, the scripting practices used to encode the application functionality, and the number of methods the application uses that are not supported in the .NET environment.

Coding restrictions

Although PowerScript is essentially a compiled language, it is quite tolerant. For the sake of performance, the PowerBuilder .NET compiler is not designed to be as tolerant as the PowerBuilder native compiler. To be able to compile your applications with .NET, you should avoid certain practices in your PowerScript code.

The following language-level items apply when you plan to transform a PowerBuilder application to a Windows Forms or Web Forms application.

Syntax issues

Avoid the GoTo statement Jumping into a branch of a compound statement is legal in PowerBuilder, because the concept of scope inside a function does not exist in PowerScript. For example, the following code works well in PowerBuilder:

```

if b = 0 then
    label: ...
else

```

```
...
end if
goto label
```

This PowerScript translates conceptually into the following C# code:

```
if (b == 0)
{ // opening a new scope
  label: ...
}
else
{
  ...
}
goto label;
```

Since a GoTo statement is not allowed to jump to a label within a different scope in .NET, the C# code would not compile. For this reason, avoid using GoTo statements.

Avoid calling an indirect ancestor event in an override event Suppose that there are three classes, W1, W2, and W3. W1 inherits from Window, W2 inherits from W1, and W3 inherits from W2. Each of these classes handles the clicked event. In the Clicked event of W3, it is legal to code the following in PowerScript:

```
call w1::clicked
```

However, in C#, calling the base method of an indirect base class from an override method is not allowed. The previous statement translates into the following C# code, which might produce different behavior:

```
base.clicked();
```

In this example, a possible workaround is to move code from the Clicked event of the indirect ancestor window to a window function, and then call the function, rather than the original Clicked event, from the descendant window.

Semantic issues

Do not use the This keyword in global functions A global function is essentially a static method of a class. Although the PowerBuilder compiler does not prevent you from using the This pronoun in a global function, the C# compiler does not allow this.

Do not change an event's signature The PowerBuilder compiler does not prevent you from changing the signature of an event defined by its super class, but .NET does not allow this. For example, suppose the w_main class contains the following event:

```
Event type integer ue_update(int e)
```

The subclasses of the `w_main` class should not change the parameters or the return type of the event.

Do not change the access modifier of an inherited function to public If your application contains a class that inherits from another class, do not change to public access the access modifiers of functions whose access level in the parent class was protected or private. The PowerBuilder compiler does not prevent you from changing the access modifier of a function in an inherited class from protected or private to public, but if you attempt to deploy a .NET target that contains such a function, you receive an error indicating that a private or protected function cannot be accessed.

Do not code Return statements in Finally clauses PowerBuilder allows you to code a Return statement in the Finally clause of a Try-Catch-Finally-End-Try statement, but C# does not support Return statements in Finally clauses. If your code includes such statements, the compiler returns the error “Return statement cannot be used in finally clause.”

External functions

Differences in passing a structure by reference PowerBuilder allows you to declare an external function that has a parameter of type Structure passed by reference. For example:

```
Subroutine CopyMemory(ref structure s, int size)
    library "abc.dll"
```

The `s` parameter can accept any datatype that is a pointer to something.

A PowerBuilder external function is mapped to the .NET Platform Invoke functionality. This functionality requires that the structure passed into the external function be exactly of the type declared. Therefore, when compiling the following PowerScript code, the PowerBuilder .NET compiler issues an error, because the parameter, `li`, references a `LogInfo` structure, which is different from the function’s declared structure class.

```
LogInfo li
CopyMemory(ref li, 20) // error!
```

To solve this problem, you can declare an additional external function as follows:

```
Subroutine CopyMemory(ref LogInfo li, int size) library
    "abc.dll"
```

Structures as parameters in .NET Applications External functions that have structures for parameters must be passed by reference rather than value if you call them in a .NET Windows Forms or .NET Web Forms application when the parameter is a const pointer. For example, a PowerScript call to the `SystemTimeToFileTime` function in *kernel32.dll* could use the following declaration, with the first parameter being passed by value and the second parameter by reference:

```
Function boolean SystemTimeToFileTime(os_systemtime
lpSystemTime, ref os_filedatettime lpFileTime) library
"KERNEL32.DLL"
```

For .NET Windows Forms or Web Forms applications, you must modify the declaration to pass both parameters by reference:

```
Function boolean SystemTimeToFileTime(ref os_systemtime
lpSystemTime, ref os_filedatettime lpFileTime) library
"KERNEL32.DLL"
```

The `SystemTimeToFileTime` function is declared as a local external function and used in `pfc_n_cst_filesrvunicode`, `pfc_n_cst_filesrvwin32`, and other operating-system-specific classes in the *pfcapsrv.pbl* in the PFC library. If you use this library in a .NET Windows Forms or Web Forms application, you must change the declaration as described above.

Allocate space before passing a string by reference Before passing a string to an external function by reference in PowerBuilder, you should allocate memory for the string by calling the `Space` system function. In subsequent calls to the function, if you pass the same string to the function, PowerBuilder continues to work well even if the string becomes empty, because memory allocated for the string is not yet freed by the PowerBuilder VM.

This is not the case in the .NET environment. If the string passed to an external function by reference is empty, and if the external function writes something to the string, an exception is thrown. Therefore, you must make sure to allocate enough space for a string before passing it to an external function by reference.

Design-level considerations

Although stricter compiler enforcement for the .NET environment can catch coding errors typically tolerated by the PowerScript compiler, the .NET environment might also require changes in application design that are not necessarily caught by the compiler.

Use PowerBuilder system functions

For a PowerBuilder .NET Web Forms application, use PowerBuilder system functions instead of external functions whenever possible. Some system functions, such as the functions for file operations, are implemented differently for Windows Forms and Web Forms. If you always use PowerBuilder system functions, you do not need to worry about these differences.

Use GetCurrentDirectory Some applications use external DLL functions to get the current directory. For PowerBuilder Web Forms applications, you must use the GetCurrentDirectory standard system function instead.

PowerBuilder Web Forms use a virtual file system to emulate a file system on the server for each client. The virtual file system is actually a folder on the server computer to which the ASPNET user (IIS 5), the IIS_WPG user group (IIS 6), or the IIS_USRS user group (IIS 7) has write permission. Calling an external function to get the current directory from the virtual file system fails.

Use regional formats based on client or server settings

The PBCultureSource global property determines whether a .NET Web Forms application uses client or server regional settings. Client regional settings are specified by the first language listed in the Language Preference dialog box of the Internet Explorer browser. However, if you set PBCultureSource to “client” and no language is listed in the Language Preference dialog box, server-side regional settings are used instead.

Server regional settings are those set for the ASP.NET user or user group on the server machine. You can use the IIS Manager to change the default regional settings in the Globalization section of the *Web.config* files for your Web Forms applications, or you can modify the *Web.config* files manually after you deploy your applications.

The regional settings specify formats for the following items:

- numeric separators (decimals or commas)
- number of digits per group to the left of a separator
- currency symbol location when a specific EditMask is not used
- date and time values

The regional settings apply to DataWindow columns of relevant datatypes and to the following PowerScript controls and functions:

- DatePicker control using the DtfLongDate!, DtfShortDate! or DtfTime! format
- EditMask control when the mask contains a [DATE], [TIME], or [CURRENCY [{digit number}]] format
- MonthCalendar control

- System String (v) function when the data argument datatype is Date, Time, or DateTime (the formats for these datatypes are [SHORTDATE], [TIME], or [SHORTDATE][TIME], respectively)
- System String (v, f) function when the format argument is [SHORTDATE], [LONGDATE], [DATE], [TIME], or [DATETIME]

The regional settings selection can also apply to objects you include in .NET conditional compilation blocks. It does not apply to button labels in message boxes or other system dialog boxes.

You can set the PBCultureSource global property on the Configurations tab in the Web Forms Project painter before you deploy a project. By default, applications use the regional settings specified by the Web Forms server.

Work around unsupported features

Avoid using Handle Some applications call the Handle function to get the window handle of a control and pass it to an external function. This does not work in a Web Forms application.

Restrict impact of unsupported events Since unsupported events are never triggered, do not allow the logic in unsupported events to affect the logic flow of other events or functions. For example, if the code in an unsupported event changes the value of an instance variable, it can affect the logic flow in a supported event that uses that variable. Remove this type of coding from unsupported events.

Avoid name conflicts PowerBuilder allows two objects to have the same name if they are of different types. For example, you can use the name *s_address* to define a structure and a static text control or a nonvisual object in the same PowerBuilder application. The .NET environment does not allow two classes to have the same name. To enable your application to compile in .NET, you must not give the same name to multiple objects, even if they are of different types.

Using structures in inherited objects Using local structures in inherited objects can prevent deployment of a .NET project. To deploy the project, replace all local structures defined in inherited objects with global structures.

AcceptText is redundant In the Web Forms deployment version of the DataWindow, explicit invocations of AcceptText are redundant but harmless. Any loss of focus of a DataWindow implicitly invokes AcceptText.

Avoid hindrances to application performance

Some functions and features that are fully supported can hinder application performance. Use these functions and features sparingly and avoid them where possible.

Response windows and message boxes Although response windows and message boxes are supported in Web Forms, use them only when absolutely necessary. Response windows and message boxes require more server-side resources than other kinds of windows.

Hiding a response window in a Web Forms application does not work properly and can cause the application to fail. Instead of hiding a response window, always close it when the user has finished with it.

Yield Although the Yield function works in a Web Forms application, avoid it whenever possible, because it requires additional server-side resources.

Timers Timers are supported in Web Forms applications, but they periodically generate postbacks and can impede data entry. Use them sparingly and avoid including them on forms that require data entry. When you use them, delay the postbacks by appropriate scripting of client-side events.

PFC The DataWindow service in PFC handles many DataWindow events. Each event causes a postback for each mouse-click, which adversely affects application performance. Delay postbacks by scripting client-side events or cache DataWindow data in the client browser by setting the paging method property for the DataWindow object to `XMLClient!`.

Take advantage of global configuration properties

Properties have been added to standard PowerBuilder controls to enhance the application presentation in the .NET environment and to improve application performance. These properties are listed in “Global Web configuration properties” on page 74. You can set them on the Configuration tab in the .NET Web Forms project painter.

The global properties are generated in the *Web.config* file in the main folder for your PowerBuilder .NET Web Forms project under the IIS server root. After deployment, you can edit the file directly, or you can modify the global properties using the IIS Manager.

For information on how to modify global properties in the IIS Manager, see “Viewing and modifying global properties in IIS Manager” on page 8.

Global properties also allow you to share data across application sessions. For information on sharing DataWindows, see “Sharing data across sessions” on page 26.

DataWindow pagination If the `HTMLGen.PageSize` property (`RowsPerPage` property in `DataWindow .NET` and `Rows Per Page` in the `DataWindow` painter) of a `DataWindow` object is not set, the `Web.config` file property `PBDataWindowRowsPerPage` limits the number of rows per page for a `Web DataWindow` control to 20 rows by default. Because this renders only the specified number of rows at a time, the `PBDataWindowRowsPerPage` helps reduce the size of the HTML response and thereby enhances performance. This property is global, since it applies to all `DataWindows` in the application for which `HTMLGen.PageSize` is not set.

Pagination and DataWindow presentation style

The `PBDataWindowRowsPerPage` setting has no effect on the number of rows in a `DataWindow` object with the `Label` presentation style. `Composite` and `Crosstab` presentation styles do not support pagination.

To disable pagination of `Web Forms DataWindow` objects, set the `PBDataWindowRowsPerPage` property to `-1`. To disable pagination for a specific `DataWindow` object, set its `HTMLGen.PageSize` property to `-1`.

DataWindow page navigation There are several global properties related to `DataWindow` page navigation. You can set the navigation bar at the top or the bottom of a `DataWindow` page by modifying the `PBDataWindowNavigationBarPosition` property. You can edit labels for the “`QuickGo`” navigation bar and the text for the current and total page counts by modifying the `PBDataWindowGoToDescription`, `PBDataWindowGoToButtonText`, and `PBDataWindowStatusInfoFormat` properties.


The `PBDataWindowPageNavigatorType` property lets you select the type of navigation bar you want to use: `NextPrev`, `Numeric`, `QuickGo`, or combined types. Figure 14-1 shows the default “`NextPrev`” navigation bar. It also displays page status information with the default text for the current and total page count. You can set the text in the `PBDataWindowStatusInfoFormat` property.

Figure 14-1: “NextPrev” navigation bar with page count display


|<< ≤ ≥ >>| Page 2 of 7

The NextPrev navigation bar includes the “>” symbol for navigating to the next page, and the “<” symbol for navigating to the previous page. Doubled symbols are controls for navigating to the first page (“<<”) or last page (“>>”). The navigation bar folds up to display only symbols that are functional when a user displays the first or last page of a DataWindow. For example, the user cannot navigate to a previous page from the first page, and navigating to the first page is unnecessary, so the “<” and “<<” symbols do not display on the first page.

Figure 14-2 displays the “NumericWithQuickGo” navigation bar. The numeric portion of the navigation bar lists each page by its page number. You can set the PBDDataWindowPageNavigatorType to “Numeric” or to “QuickGo” if you want to use these styles separately. You can also combine the NextPrev style with the QuickGo style by setting the PBDDataWindowPageNavigatorType property to “NextPrevWithQuickGo”.

Figure 14-2: “NumericWithQuickGo” navigation bar and page count


[1] [2] [3] [4] [5] [6] [7] Go To: 7 Page 7 of 7

Although the QuickGo navigation control defaults to a drop-down list, you can change this to a text box with an associated command button by setting the PBDDataWindowQuickGoPageNavigatorType property to “Button”. You can edit the button label by setting the PBDDataWindowGoToButtonText property. You set the label for the text box or the drop-down list by modifying the PBDDataWindowGoToDescription property.

Use client-side events to delay postbacks

Before the .NET target is deployed, you can code client-side events in JavaScript and set properties to reference the JavaScript code that handles client-side events. You must set the properties in #IF DEFINED -#END IF conditional compilation code blocks for .NET targets. The beginning and end tags for these code blocks signal the PowerBuilder native compiler to ignore the code contained inside them.

For more information, see “About conditional compilation” on page 187.

The code inside the conditional compilation code blocks is passed to the Web browser client from the server at runtime. You use this code to designate JavaScript functions that handle events on client-side objects. Most events on client-side objects cause a postback to controls on the server side, because the events have server-side analogs that are written originally in PowerShell, then transformed to run in the .NET environment.

If you write any JavaScript code for the client-side events, the postback to the server is interrupted. To resume a postback, you can call the submit method for Web Forms or one of the postback methods generated in the *PBDataWindow.JS* file. The *PBDataWindow.JS* file is generated in the Scripts subdirectory of the main project directory under the IIS virtual root.

The postback methods of the *PBDataWindow.JS* file are described in Chapter 3, “Client-Side Events and Default Event Handlers.”

DataWindow property for setting a customized event handler Properties of the DataWindow class allow you to handle client-side events in JavaScript code. The JavaScriptFile property specifies the JS file that contains JavaScript functions for handling individual client-side events. Make sure to deploy the JavaScript file that contains your customized event handling code. You assign the JavaScriptFile property in an #IF DEFINED -#END IF code block:

```
#IF Defined PBWEBFORM THEN
    dw_1.JavaScriptFile = "D:\Scripts\MyScriptFile.js"
#end if
```

DataWindow properties for calling client-side events The following DataWindow events can be handled on the client side in JavaScript code:

- Clicked
- ButtonClicking
- ButtonClicked
- DoubleClicked
- ItemChanged
- ItemError
- ItemFocusChanged
- RButtonDown
- RowFocusChanged
- RowFocusChanging

For more information on client-side events, see Chapter 3, “Client-Side Events and Default Event Handlers.”

To specify a JavaScript function for handling a client-side event, you must indicate the function to call in the corresponding Web DataWindow property. The name of the corresponding property consists of the name of the client-side event with an “OnClient” prefix. For example, the property corresponding to the ItemChanged event is OnClientItemChanged.

The following example references a script called MyDwClickedEventHandler for the client-side DataWindow Clicked event. The script for the MyDwClickedEventHandler event handler must use the syntax for the client-side Clicked event described in Chapter 3, “Client-Side Events and Default Event Handlers.”

```
#IF Defined PBDOTNET THEN
    dw_1.JavaScriptFile = "D:\Scripts\MyScriptFile.js"
    dw_1.OnClientClicked = "MyDwClickedEventHandler"
#END IF
```

Client-Side CommandButton property The OnClientClick CommandButton property specifies a snippet of JavaScript code that executes when a command button is clicked.

AutoPostBack You can reduce postbacks and increase performance by setting the AutoPostBack property for CheckBox and RadioButton controls to false.

```
#IF DEFINED PBWEBFORM THEN
    cbx_1.AutoPostBack = false
#END IF
```

For more information on the built-in Web Forms control properties, see Chapter 6, “Properties for .NET Web Forms.”

About this chapter

This chapter explains how to debug PowerBuilder .NET applications and components and provides troubleshooting tips.

Contents

Topic	Page
Debugging a .NET application	213
Debugging a .NET component	217
Troubleshooting deployment errors	219
Troubleshooting tips for Web Forms applications	219
Troubleshooting tips for Windows Forms applications	224

Debugging a .NET application

After you have deployed a PowerBuilder Web Forms or Windows Forms application, you can debug it. To launch the application and open the debugger, right-click the target or project in the System Tree and select Debug from its pop-up menu. You can also select the Debug Project button or the Design>Debug Project menu item in the Project painter, or the Debug button in the PainterBar.

.NET debugger restrictions

The .NET debugger supports most features of the debugger for standard PowerBuilder applications, including expression evaluation and conditional breakpoints. It does not support the Objects in Memory view or variable breakpoints, which are not supported in .NET. Local variables that are declared but not used do not display in the Local Variables view in .NET targets.

Additional debugging restrictions include the following:

- **Debugger icon display in .NET Web Forms projects** When you close a .NET Web Forms application that is being debugged, the Stop Debugging icon remains enabled in the debugger, and the StartDebugging icon is disabled.

- **Single-stepping between events** In the .NET debugger, when you step into a statement or function in an event script, the debugger displays the next line of code. However, if you step into a different event script, the debugger continues execution to the next breakpoint. You should add a breakpoint to each event that you want to debug.

For example, if you have set a breakpoint in the application's Open event, and the script opens a window, the debugger does not step into the window's Open event. You should set a breakpoint in the window's Open event or in a user-defined event that is called from the Open event.

- **Server support restrictions for .NET Web Forms projects** The .NET debugger does not support IIS 6 if the maximum number of worker processes is set to greater than one. This is because it cannot determine whether the process to be debugged is newly created or is recycled from a pool of worker processes. (The debugger must attach to the worker process in Web garden mode.) It also does not support the Cassini Web server that ships with .NET Framework 2.0.
- **Multiple applications using the same PBLs** When you run or debug a Web Forms application, its PBLs can remain cached in the ASP.NET process. If you then try to debug a second Web Forms application that shares a PBL with the first application, the ASP.NET process lets the debugger know that the first module is loaded and the debugger binds to breakpoints in that module. In this case, the debugger never binds to breakpoints in the second application. You can avoid this issue by not sharing PBLs among Web Forms projects or by restarting IIS before you begin debugging.
- **Remote debugging** Debugging of Web Forms or Web Service targets is not supported for applications or components deployed to remote IIS servers.

For information about standard PowerBuilder debugger features, see “Debugging an application” in the *User’s Guide*.

Release and Debug builds

On the General page in the Project painter, you can choose to build a release build or a debug build. If you choose a debug build, an extra file with the extension .PDB is generated in the output directory and additional information displays in the Output window. If you want to stop at breakpoints in your code, you must use a debug build. Select a release build when your application is ready to distribute to users.

DEBUG preprocessor symbol

You can also enable or disable the DEBUG preprocessor symbol. This is useful if you want to add code to your application to help you debug while testing the application. Although you do not typically enable the DEBUG symbol in a release build, if a problem is reported in a production application, you can redeploy the release build with the DEBUG symbol enabled to help determine the nature or location of the problem.

When the DEBUG symbol is enabled, code that is enclosed in a code block with the following format is parsed by the pb2cs code emitter:

```
#if defined DEBUG then
    /*debugging code*/
#else
    /* other action*/
#endif if
```

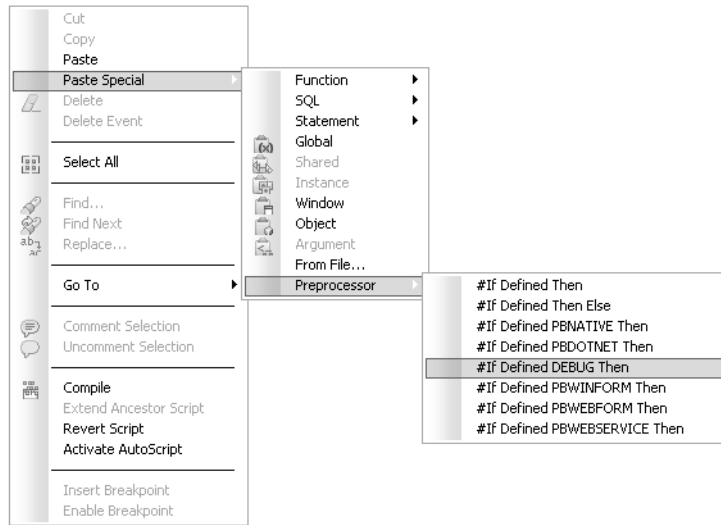
Adding breakpoints in a DEBUG block

When you use the DEBUG symbol, you can add breakpoints in the DEBUG block only for lines of code that are not in an ELSE clause that removes the DEBUG condition. If you attempt to add a breakpoint in the ELSE clause, the debugger automatically switches the breakpoint to the last line of the clause defining the DEBUG condition.

In the previous pseudocode example, if you add a breakpoint to the comment line “/* other action*/”, the breakpoint automatically switches to the “/*debugging code*/” comment line.

Figure 15-1 shows the pop-up menu item that you can use to paste the #If Defined DEBUG Then template statement in the Script view.

Figure 15-1: Menu cascade for pasting a template into a script



For more information about using preprocessor symbols, see “About conditional compilation” on page 187.

Attaching to a running Windows Forms process

For Windows Forms projects, you can start your deployed application from its executable file before starting the debugger, and then attach to the running process from the debugger. To attach to a process that is already running, select Run>Attach to .NET Process in the Project painter to open a dialog box from which you can select the process.

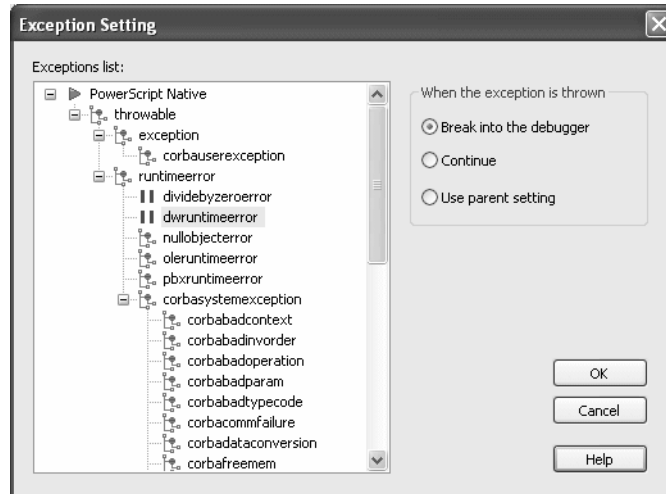
After you attach to the process, it starts running in the debugger and you can set breakpoints as you normally do. Select Run>Detach to detach from the process. This gives you more flexibility than simply using just-in-time (JIT) debugging.

Breaking into the debugger when an exception is thrown

When an application throws an exception while it is being debugged, the debugger sees the exception before the program has a chance to handle it. The debugger can allow the program to continue, or it can handle the exception. This is usually referred to as the debugger’s first chance to handle the exception. If the debugger does not handle the exception, the program sees the exception. If the program does not handle the exception, the debugger gets a second chance to handle it.

You can control whether the debugger handles first-chance exceptions in the Exception Setting dialog box. To open the dialog box, open the Debugger and select Exceptions from the Debug menu. By default, all exceptions inherit from their parent and all are set to Continue. Figure 15-2 shows the DWRuntimeError exception has been set to “Break into the debugger.”

Figure 15-2: Exception Setting dialog box



When this exception is thrown, a dialog box displays so that you can choose whether to open the debugger or pass the exception to the program.

Debugging a .NET component

.NET Assembly component You can run or debug an assembly project from the PowerBuilder UI if you fill in the Application field (and optionally, the Argument and Start In fields) on the project Run tab in the Project painter. Table 12-3 describes the Run tab fields for a .NET Assembly project.

.NET Web Service component When you start the debugger and Internet Explorer is listed as the application to run a Web Service project, a browser test page opens with links to the Web services deployed from your project.

Using the DEBUG symbol If you used the DEBUG conditional compilation symbol in code for the nonvisual objects you deploy as a Web service and you want this code to run, you must make sure that the enable DEBUG symbol check box is selected before you deploy the project. If you plan to debug the assembly or Web service, you should make sure the project is deployed as a debug build.

Troubleshooting deployment errors

The deployment process has two steps: the PowerBuilder to C# emitter (pb2cs) runs, then the project is compiled. Errors are written to the output window, and the progress of the deployment process is written to the *DeployLog.txt* file.

PB2CS errors

If pb2cs fails, make sure that:

- The PBNET_HOME system environment variable is set to the location of your *PowerBuilder 11.0\DotNET* directory.
- The *pb2cs.exe* file is present in the *PowerBuilder 11.0\DotNET\bin* directory and is the version distributed with the current PowerBuilder release.

If pb2cs fails and your application has any objects or controls whose names include dashes, open a painter with a Script view and select Design>Options from the menu bar. Make sure the Allow Dashes in Identifiers option is selected on the Script page in the Design Options dialog box.

If your application uses local structures in inherited objects, the .NET project might fail to deploy. To deploy the project successfully, replace all local structures defined in inherited objects with global structures. Also, your application must not include calls to functions, such as ToString, on primitive .NET datatypes, such as System.String, that map to PowerBuilder datatypes. See Table 13-3 for the list of datatype mappings from .NET to PowerBuilder.

Build errors

If there is a build failure, make sure the 2.0 version of the .NET Framework is installed and is listed in your PATH environment variable before any other versions of the .NET Framework.

Errors that display in the Output window with a CS prefix, such as error CS0161, are generated by the Microsoft C# compiler. There is no link from these errors back to the source code in PowerBuilder painters. Explanations for C# compiler errors can be found at the Microsoft Web site at [http://msdn2.microsoft.com/en-us/library/ms228296\(VS.80\).aspx](http://msdn2.microsoft.com/en-us/library/ms228296(VS.80).aspx).

Troubleshooting tips for Web Forms applications

After successfully deploying a PowerBuilder .NET project, you might encounter some of the following common, easy-to-fix issues at runtime.

Failure to deploy to local machine alias

You can deploy a Web Forms project to a local IIS server using “localhost” or one of the following aliases:

- Machine name
- Machine IP address
- 127.0.0.1 (the generic DNS address for the local machine)

However, in order to use the machine IP address or the generic DNS address for the local machine, you must share the wwwroot directory as “wwwroot\$” and enable write permissions for this directory.

If you are deploying .NET applications from a computer with the Vista operating system, you must run PowerBuilder as the computer administrator.

Browser error messages

Null reference exception The “Object reference not set to an instance of an object” error message might display in a client browser for a Web Forms application if the application uses an unsupported version of the .NET Framework. The error message description indicates that “an unhandled exception occurred during the execution of the current web request,” and the exception details display a “System.NullReferenceException”.

You can resolve this type of error by opening a command window on the server, changing directories to the Microsoft.NET\Framework\v2.0.50727 directory in the Windows system path, and typing the following command:

```
aspnet_regiis -i.
```

This upgrades all IIS scriptmaps to use the 2.0 release version of ASP.NET. After running this command and restarting the server, the error message should no longer display.

Could not load <Global.app> This error is usually due to an incorrect ASP.NET or IIS configuration setup. To resolve this issue, make sure .NET Framework 2.0 is installed on the server machine, register ASP.NET 2.0 with IIS by running `aspnet_regiis.exe -I` from the .NET Framework 2.0 directory, and make sure ASP.NET 2.0 is the version set for your Web application in the IIS Manager. You might also need to restart IIS.

Exception from HRESULT: 0x8007007E This error can be caused by different versions of PowerBuilder 11.0 .NET assemblies in the server environment. To resolve this issue, remove extra copies of PowerBuilder 11 .NET assemblies from the Global Assembly Cache (GAC), leaving only the latest copies of each assembly.

Page cannot be displayed This error is also known as the “404 file not found” error. If you see this error, make sure all the application files and folders have been generated under the `wwwroot` directory on the IIS server machine. If you are using a TCP port number other than 80 (the default port number), you must include the port number in the URL for the Web Forms application.

If you are trying to open the page from a remote client, ping the server to make sure it is accessible. If the firewall is on for the server you are accessing, turn it off and open the page again.

File not found exception After successfully deploying a Web Forms application, you might see an error such as the following when you try to run the application: `System.IO.FileNotFoundException: The specified module could not be found.` This is typically because IIS cannot locate PowerBuilder runtime DLLs, such as `pbdwm110.dll` or `pbshr110.dll`. To resolve this issue, make sure the directory where the runtime DLLs are located is included in the system path on the server machine.

Problem with toolbars and tab controls

If the application toolbar and tab controls are not working properly, this is most likely due to the improper installation of the Microsoft IE Web controls. For more information, see “Setting up IE Web Controls on the server” on page 12.

Failure to connect to database

DSN Due to limited access rights of ASP.NET user and user group accounts, data sources created as User DSNs cannot be loaded. You must create the data sources for your Web Forms application as System DSNs.

Oracle The appropriate user or user group must be granted full control rights to the Oracle Client directory. For example, if the Oracle client is installed in the c:\oracle\ora9 directory, the ASPNET user (IIS 5), the IIS_WPG user group (IIS 6), or the IIS_USRS user group (IIS 7) must have full control rights to this directory.

SQL Anywhere To launch a SQL Anywhere database automatically from a Web Forms application, the appropriate user or user group must be granted at least read and execution rights to the directory indicated by the SQLANY10 or ASANY9 environment variable. The ASPNET user, the IIS_WPG user group, or the IIS_USRS user group must also have full control privileges to the directory that contains the database.

Database connections using an INI file If your application uses an INI file to get database connection information, make sure to add the INI file to the resource file list of your .NET Web Forms project before you deploy it.

JDBC connections If an error message indicates that the Java VM cannot be initialized, make sure that the system CLASSPATH and JAVA_HOME environment variables have been set correctly. If an error message indicates that you are attempting to read from or write to protected memory, make sure the ASPNET user, the IIS_WPG user group, or the IIS_USRS user group has at least read, execute, and list folder contents permissions for the vendor's JDBC directory.

After making any changes to the directory permissions or system environment variables, restart the IIS service and either ASPNET_WP.EXE (IIS 5) or W3WP.EXE (IIS 6 and IIS 7). Alternatively, you can restart the IIS server machine to make sure that the changes take effect.

DataWindows do not display

Make sure the PBL files that contain the DataWindows you want to display are copied to the directory you assigned to the PBLibDir global property. By default, the PBLibDir global property assigns C:\~PL_ as the directory for application PBL files. This corresponds to the *File\Common\C\~PL_* subdirectory of the *applicationName_root* directory in the server's virtual file system path.

Pictures do not display

Before you deploy a .NET Web Forms project, make sure you add all picture files used by the application to the resource file list for the project.

Resource files might not be accessible if you change the default value for the initial current directory of the virtual file system for the Web Forms project. The default value in the .NET Web Forms Application wizard is the current target path. Modifying the `PBCurrentDirectory` global property in the project's ASP.NET configuration settings or directly in the *Web.config* file might also make the resource files inaccessible.

Excessive flickering on Web page

A Web Forms application user might encounter excessive flickering in an application if a default browser setting has been changed. When this occurs, the user must select the “Enable page transitions” check box on the Advanced tab of the Internet Options dialog box to minimize or eliminate the flickering problem. The user can open the Internet Options dialog box from the Tools menu of Internet Explorer.

Posted events are not executed

If you post an event in a response window that closes the response window, and call posted events in the Open event for a main window that displays when the response window is closed, the posted events in the Open event are not executed. This is due to a limitation of the threading model in Web Forms applications.

To make sure that the posted events of the main window are executed, close the response window directly in a triggered event rather than in a posted event. Alternatively, move the code from posted events in the main window to events that are triggered directly by the user.

External DLLs cannot be loaded

Make sure the DLLs you want to load are copied to the *bin* subdirectory of the main Web Forms application directory in the server's virtual file system path.

Print failure

Some of the PowerScript print functions are not supported in the current release. If your applications saves or exports DataWindows as PDF or XSL-FO files, make sure you read the instructions for installing the appropriate printing software on the Web Forms server.

For more information, see “Requirements for saving files in PDF or XSL format” on page 57.

Log files

Log.txt A PowerBuilder application that compiles successfully with the PowerBuilder native compiler might not compile successfully with the PowerBuilder to .NET compiler. At deployment time, PowerBuilder logs all compilation errors and warnings into the application’s *log.txt* file. The PowerBuilder to .NET compiler is stricter than the PowerBuilder native compiler, as described in Chapter 14, “Best Practices for .NET Projects.” If deployment fails, or if issues occur at runtime, review the errors and warnings in the *log.txt* file.

Pbtrace.log At runtime, a Web application logs all exceptions in the *pbtrace.log* file located in the *applicationName_root\log* directory. You can look into the call stack when an exception is thrown and map the call stack back to PowerScript code, from which you might find the root cause of any runtime errors.

Problems on Windows 2003

Enabling ASP.NET services By default, IIS is installed on Windows Server 2003 in a secure mode that prohibits ASP.NET. This means that only static pages can be served. To enable ASP.NET, do the following:

- 1 From the Start menu, click Administrative Tools and then choose the Internet Information Services (IIS) Manager.
- 2 Select Web Service Extensions in the left pane.

In the right pane, you should see a list of allowed and prohibited Web service extensions.
- 3 Click “ASP.NET v2.0.50727” in the right pane, then click Allow to enable ASP.NET services.

Enabling ActiveX controls and plug-ins You might also need to enable running and scripting ActiveX controls and plug-ins. To enable ActiveX controls and plug-ins:

- 1 Open Internet Explorer.
- 2 Select Internet Options from the Tools menu.
- 3 On the Security tab, select the Internet zone and click the Custom Level button.
- 4 Under the “ActiveX controls and plug-ins” settings, change the “Run ActiveX controls and plug-ins” and the “Script ActiveX controls marked safe for scripting” settings to Enable.

Troubleshooting tips for Windows Forms applications

If you experience difficulty deploying, running, publishing, or updating an application, make sure you have installed the .NET Framework and SDK as described in “System requirements” on page 127, then review the suggestions in this section. Also review the known issues listed in the *pb110readme.txt* file distributed with this beta release.

Runtime errors

The application might not run correctly when you select Design>Run Project in the Project painter, when you run the executable file in the deployment folder, or when a user runs the installed application. When you or a user runs the executable file, PowerBuilder creates a file called *PBTrace.log* in the same directory as the executable. This file can help you trace runtime errors. It can be configured by editing the *appname.exe.config* file, where *appname* is the name of the executable file:

```
<appSettings>
  <!-- The value could be "enabled" or "disabled"-->
  <add key ="PBTrace" value ="enabled"/>
  <!-- The target can be File, EventLog or File|EventLog -->
  <add key ="PBTraceTarget" value="File"/>
  <!-- If the Target is File, PBTraceFileName should also be
  specified.-->
  <add key ="PBTraceFileName" value ="PBTrace.log"/>
</appSettings>
```

```

<!-- EventLogId is optional(0 is default), and it only
      works when EventLog is enabled-->
<add key ="PBEventLogID" value ="1101"/>
...

```

The following problems might also occur:

- If the application cannot be launched from another computer, make sure the required PowerBuilder runtime files, *pbshr110.dll* and *pbdwm110.dll*, and the Microsoft runtime files on which they depend, *at71.dll*, *msvcp71.dll*, and *msvcr71.dll*, are available on the other computer and in the application's path.

If the executable file is located on a network path, the .NET Framework must be configured to have Full Trust permissions at runtime. See "Security settings" on page 140.

- If the application cannot connect to a database, make sure that the required PowerBuilder database interface, such as *pbodb110.dll*, has been added to the Win32 dynamic library files section of the Library Files tab page and that the required client software is available on the target computer. If the application uses a configuration file, such as *myapp.ini*, select it on the Resource Files tab page. For ODBC connections, make sure that the DSN file is created on the client.
- If no data displays in DataWindow objects, select the PBLs that contain them on the Library Files tab page.
- If graphics fail to display, select them on the Resource Files tab page.

Publish errors

There are two steps in the publication process. First, publish files are generated, and then they are transferred to the publish location. Publish errors are displayed in the Output window and recorded in a file called *pbiupub.log* in the output directory.

The following errors might be reported during file generation:

- **Failure to create local folder structure** Check that you have permission to create a folder in the specified directory.
- **Failure to generate application manifest file** Check that the .NET Framework 2.0 SDK *bin* directory is in your PATH environment variable. If a certificate file is specified, check that it exists in the specified location and is a valid certificate.

Use different output paths for multiple projects

If you create more than one Windows Forms project for a single application, make sure you specify a different output path on the General page for each project. If you do not, the application manifest files generated for each project conflict with each other.

The following errors might be reported during file transfer:

- **Publish location is a Web server: http://servername/appname**
Check that *servername* and the development computer are in the same network domain and that you are in the administrators group of *servername* or have write access to the *wwwroot* directory on *servername*.
- **Publish location is a file share: \\servername\appname** Check that *servername* and the development computer are on the same network and that you have write access to the *appname* directory on *\\servername*.
- **Publish location is an FTP site: ftp://servername/appname** Check that *servername* can be accessed using the specified user name and password and that you have write access to the *appname* directory on *\\servername*.

You should also check that the publish location name is typed correctly, that the PBNET_HOME environment variable is set correctly, and that network connections are working correctly.

Installation errors

If installation on the client computer fails, make sure that:

- The files exist in the location specified on the server.
- The link on the publish page matches the location where the files have been published.
- The user has access rights to the publish server.
- There is sufficient space on the user's computer.
- The network connection to the publish server is working correctly.
- You have not used *localhost* as the publish or install location.

If the publish page fails to open on the client, check the firewall settings on the publish server. The firewall must be turned off on the server.

If the *setup.exe* file is not downloaded when a prerequisite is selected, open the Properties dialog box for the HTTP directory in IIS Manager and make sure the script source access permission is enabled. If the Execute Permissions property is not set to Scripts only, select Scripts only from the drop-down list and refresh the server.

Update errors

If update fails, make sure that the update mode has been set as you intended and that the update files are in the specified location.

Index

A

- access permissions, ASP.NET 10, 182
- application
 - directory structure 143
 - installing 139, 142
 - manifest 140
 - publishing 138
 - running mode 139, 143
 - update mode 139, 144
 - updating 143
- ASP.NET
 - configuring 6
 - setting user permissions 10
 - version 7
- AutoPostBack property 80, 211

B

- best practices 126
- bootstrapper
 - about 146
 - customizing 147
- ButtonClicked event 36
- ButtonClicking event 37

C

- Clicked event 38
- ClickOnce technology 137
- client-side, event handlers 30
- command line parameters 26
- company name, setting 132, 142
- components
 - .NET Assembly target 169
 - .NET Web Service target 176

- conditional compilation, about 187
- configuring
 - ASP.NET 6, 181
 - IE Web Controls 12
 - SQL Anywhere database connection 10
- controls, supported 158
- conventions x
- copy mode, global property 62

D

- data, synchronizing 149
- datatype mapping 175, 193
- DataWindow
 - page navigation 208
 - pagination 208
 - saving as PDF 57
 - saving as XSL 58
- Debug builds 214
- DEBUG preprocessor symbol 215
- debugging
 - .NET applications 213
 - .NET components 217
- deploy
 - .NET Assembly project 175
 - .NET Web Service project 182
 - checklist for production servers 12
 - troubleshooting 218
 - Web Forms project 22
 - Windows Forms project 134
- deployment manifest 141
- directory structure, on server 143
- DLLs, deploying 132
- DoubleClick event 39
- DownloadFile Web Forms function 86
- downloading files 66

E

- Embedded property 80
- event handlers
 - and postback events 30
 - client-side 30
 - default 31
- events
 - ClientEvent properties 33
 - Web DataWindow client control 33
- exceptions, handling in .NET environment 198

F

- File Manager
 - creating a folder 64
 - downloading a file 66
 - uploading a file 65
 - virtual file system 60
- file process mode 62
- file server, setting up 138
- fonts, using TrueType in controls in Windows Forms 152
- FTP server, setting up 138
- full trust 140

G

- GetConfigSettings Web Forms function 87
- GetDownloadFileURL Web Forms function 88
- Ghostscript, installing 59
- global properties
 - and .NET Web Service targets 181
 - creating 79
 - descriptions of 74
 - list of 74
- global properties, taking advantage of 207

H

- handling exceptions, in .NET environment 198
- HasFileManager property 81
- HasMailManager property 82
- HasPrintManager property 82
- HasThemeManager property 83

- HTMLGen.PagingMethod property 31

I

- IE Web Controls
 - configuring 12
 - problem with toolbars and tab controls 220
- IIS, directory structure 9
- images, deploying 133
- intelligent notifier 145
- intelligent update 137, 144
- Internet trust 140
- interop. *See* interoperability
- interoperability
 - datatype mappings 193
 - referencing .NET classes 187
 - support for .NET language features 194
 - writing code in a .NET block 190
- Intranet trust 140
- ItemChanged event 40
- ItemError event 41
- ItemFocusChanged event 42

L

- library files 132
- log file, pbtrace 11

M

- Mail Profile Manager 67
- mail, sending 70
- mandatory updates 145
- manifest
 - application 140
 - deployment 141
- MobiLink synchronization, for smart clients 149

N

- navigation controls 208
- .NET Assembly project 172

- .NET Assembly target wizard 169
- .NET calls, PowerShell syntax for 191
- .NET compiler 17, 125
- .NET environment
 - debugging 213, 217
 - handling exceptions 198
 - support for language features 194
- .NET Framework 2.0 127
- .NET Framework 2.0 SDK 127
- .NET language features, support for 194
- .NET Web Forms
 - application wizard 18, 127
 - code block 29, 190
 - coding restrictions 201
 - configuring for 6
 - global properties 8, 74
- .NET Web Service project 178
- .NET Web Service target wizard 176
- .NET Windows Forms Application project 127
- .NET Windows Forms Application wizard 126
- notifier
 - icon 145
 - options 145

O

- OnClient event prefix 33
- online only 143
- OpenFileManager Web Forms function 89
- OpenMailManager Web Forms function 89
- OpenPrintManager Web Forms function 90
- OpenThemeManager Web Forms function 90

P

- PATH environment variable 128
- PBDataWindow.JS file 31
- PBDs, deploying 132
- PBLs, deploying 132
- PBTrace.Log file 11
- PDF
 - Apache FO printing method 58
 - postscript printing method 57

- permanent user accounts 47
- postbacks
 - and client-side events 209
 - avoiding 31
 - from default event handlers 31
- post-build commands 132
- PowerBuilder runtime files, deploying 132
- PowerScript
 - registry functions 53
 - unsupported events 115, 163
 - unsupported functions 111, 162
 - unsupported properties 117, 164
- preprocessor symbols
 - about 187
 - DEBUG 215
 - list of 188
- prerequisites
 - for application 141, 146
 - for development 127
- printing
 - Apache FO software processing 58
 - DataWindow as PDF 57
 - DataWindow as XSL 58
 - GNU Ghostscript software processing 59
 - output location 57, 60
- properties, global 8, 74
- publish page
 - link to server 138
 - prerequisites 146
 - view of 142
- publishing an application 138

R

- RButtonDown event 43
- registry functions, in Web Forms applications 53
- Release builds 214
- requirements, system 18, 127
- resources, deploying 133
- RowFocusChanged event 44
- RowFocusChanging event 45
- running an application 135

S

- script
 - .NET code block 29
 - client-side events 29
- security settings 140
- share mode, global property 62
- sharing data
 - across sessions 26
 - DropDownDataWindows 27
- smart client
 - intelligent update feature 137
 - rolling back 148
- SQL Anywhere database connection, setting up 10
- Start menu, adding to 143
- structures, supported 153
- supported features 151
- system objects, supported 153
- system requirements 18, 127

T

- troubleshooting
 - deployment errors 218
 - tips for Web Forms applications 219
 - tips for Windows Forms applications 224
- TrueType fonts, using in controls in Windows Forms 152
- trust, security settings 140
- typographical conventions x

U

- unsupported features 151
- updates
 - checking for 144, 145
 - mandatory 145
 - online and offline 144
 - online only 144
 - polling for 146
- UploadFiles Web Forms function 90
- uploading files 65
- users, permanent accounts 47

V

- visual controls, supported 158

W

- Web browser
 - command line parameters 26
 - default start page 25
- Web DataWindow
 - client-side scripts 33
 - events for client control 33
- Web Forms applications
 - advantages 3
 - directory structure 9
 - global properties 8
 - sending mail 70
 - start page 26
 - supported controls 98
 - unsupported features 93
 - using registry functions 53
 - virtual file system 60
- Web Forms function
 - DownloadFile 86
 - GetConfigSettings 87
 - GetDownloadFileURL 88
 - OpenFileManager 89
 - OpenMailManager 89
 - OpenPrintManager 90
 - OpenThemeManager 90
 - UploadFiles 90
- Web server, setting up 138
- Web service components, access permissions 182
- Windows and Web Forms applications, advantages of 125
- Windows Forms Application project 127
- Windows Forms Application wizard 126
- Windows Forms applications
 - supported controls 159
 - supported objects 155