SYBASE®

An SAP® Company

Deploying Applications and Components to .NET

PowerBuilder® 12.5

Contents

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.

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.

This table shows some of the advantages and disadvantages of Web Forms and Windows Forms applications.

Application type	Advantages	Disadvantages
Web Forms	• No installation • Easy to upgrade • Broader reach	• Slower response • Must be online
Windows Forms	• Rich user experience • Quicker response time • Availability of client-side resources, such as 3D animation • Offline capability	• Requires client-side installation • Difficult to upgrade
Windows Forms with smart client feature	• Same advantages as Windows Forms without smart client feature • Easy to deploy and upgrade	• Requires first time client-side installation

Note: The PowerBuilder® 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 *Intelligent Deployment and Update* on page 129.

Although PowerBuilder Classic continues to support traditional client-server as well as distributed applications, it also 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://msdn.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.

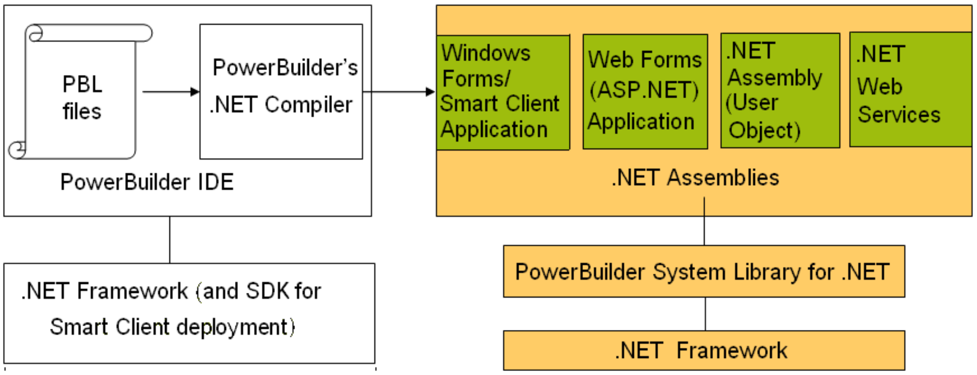
PowerBuilder Web Forms applications have a three-tier architecture, with the client running in a Web browser on the front end and PowerBuilder components running on the Microsoft IIS server using ASP.NET 4.0 technology. A session is created and is dedicated to processing each user request on the client 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 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.

Note: For PowerBuilder .NET applications and components, you must install the .NET Framework redistributable package on the deployment computer or server. The .NET Framework SDK (x86) is required on the deployment server for Windows Forms smart client applications. The x86 version of the SDK is required even for 64-bit computers. You cannot install the SDK without first installing the redistributable package.

he SDK and the redistributable package are available as separate downloads from the Microsoft .NET Framework Developer Center at *http://msdn.microsoft.com/en-us/netframework/aa731542.aspx*.

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


PBL files
PowerBuilder's .NET Compiler
PowerBuilder IDE
.NET Framework (and SDK for Smart Client deployment)
Windows Forms/ Smart Client Application
Web Forms (ASP.NET) Application
.NET Assembly (User Object)
.NET Web Services
.NET Assemblies
PowerBuilder System Library for .NET
.NET Framework

Security Settings

PowerBuilder applications and components can run in partial trust environments when they are constrained by .NET code access security (CAS) configurations.

PowerBuilder lets you configure CAS security zones (sandboxes) for .NET Web Forms, .NET Web Service, and Windows Forms smart client projects, to minimize the amount of trust required before application or component code is run by an end user.

For .NET Web Forms and Web Service projects, you can also modify the `Web.config` file to support security zones after you deploy the project. The .NET assemblies that you create by building and deploying .NET Assembly projects are run with the security permissions of the calling application or component.

However, Microsoft .NET Framework 4.0 does not support the machine security policy and zone settings used by Windows Forms applications. To continue to use the CAS policy system in a Windows Forms application, modify its application.config files as as follows:,

```
<configuration>
  <runtime>
    <NetFx40_LegacySecurityPolicy enabled="true"/>
</runtime>
</configuration>
```

Trust options

A radio button group field on the Project painter's Security tab allows you to select full trust or a customized trust option. For Windows Forms applications, you can also select local intranet trust or internet trust. A list box below the radio button group allows you to select or display the permissions you want to include or exclude when you select local intranet trust, internet trust, or the custom option. (If you select full trust, the list box is disabled.)

For Windows Forms applications, if you modify a permission after selecting the local intranet or internet trust options, PowerBuilder automatically treats the selected permissions as custom selections, but does not modify the selected radio button option. This allows you to click the Reset button above the list box to change back to the default local intranet or internet permission settings. Clicking the Detail button (to the left of the Reset button) opens the Custom Permissions dialog box that allows you to enter custom permissions in XML format. The Reset and Detail buttons are disabled only when you select the Full Trust radio button option.

For smart client applications, the permission information is stored in the manifest file that you deploy with your application. When users run a smart client application, the application loader process loads the manifest file and creates a sandbox where the application is hosted.

For standard Windows Forms applications, the sandbox allows you to run the application with the permissions you define on the Project painter Security tab when the applications are run from the PowerBuilder IDE. When a user starts Windows Forms applications from a file explorer or by entering a UNC address (such as \\server\myapp\myapp.exe), the security policies set by the current user's system are applied and the Security tab permission settings are ignored.

Note: For information on custom security permissions, see *Custom Permission Settings* on page 231 and the Microsoft Web site at *http://msdn.microsoft.com/en-us/library/system.security.permissions.aspx*.

Permission error messages

If your .NET application attempts to perform an operation that is not allowed by the security policy, the Microsoft .NET Framework throws a runtime exception. For example, the default local intranet trust level has no file input or output (File IO) permissions. If your application runs with this security setting and tries to perform a File IO operation, the .NET Framework issues a File Operation exception.

You can catch .NET security exceptions using .NET interoperability code blocks in your PowerScript code:

```
#if defined PBDOTNET then
      try
      ClassDefinition cd_windef
      cd_windef = FindClassDefinition("w_1")
      messagebox("w_1's class
      definition",cd_windef.DataTypeOf)
       catch(System.Security.SecurityException ex)
         messagebox("",ex.Message)
       end try
#end if
```

All .NET targets must include a reference to the `mscorlib.dll` .NET Framework assembly in order to catch a System.Security.SecurityException exception. The `PBTrace.log` files that PowerBuilder Windows Forms and Web Forms applications generate by default contain detailed descriptions of any security exceptions that occur while the applications are running. PowerBuilder .NET Web Service components also generate `PBTrace.log` files that log critical security exceptions by default.

If you do not catch the exception when it is thrown, the PowerScript SystemError event is triggered. For Windows Forms applications, a default .NET exception message displays if you do not catch the exception or include code to handle the SystemError event. The exception message includes buttons enabling the user to show details of the error, continue running the application, or immediately quit the application. Under the same conditions for Web Forms applications, a system error dialog box displays instead of the .NET exception dialog box, and the application is terminated.

For more information about handling .NET exceptions, see *Handling Exceptions in the .NET Environment* on page 188.

Debugging and tracing with specified security settings

You can debug and run .NET applications and components from the PowerBuilder IDE with specified security settings. To support this capability in Windows Forms applications, PowerBuilder creates a hosting process in the same directory as the application executable. The hosting process creates a domain with the CAS setting before it loads the application assemblies. (The CAS settings generated in the `Web.config` file determine the security permissions used by .NET Web Forms applications and .NET Web Service components.)

If your .NET application attempts to perform an operation not allowed by the specified security setting, an exception is issued in the IDE.

For more information about debugging .NET applications and components, see *Debugging a .NET Application* on page 217.

Strong-Named Assemblies

PowerBuilder can generate strong-named assemblies from all .NET Project painters.

A strong name consists of an assembly's identity—its simple text name, version number, and culture information (when provided)—plus a public key and digital signature. It is generated from an assembly file using the corresponding private key. The assembly file contains the assembly manifest that includes the names and hashes of all the files that make up the assembly.

Project painter Sign tab

PowerBuilder includes a Sign tab in the Project painters for all .NET application and component projects. The Assembly group box on the Sign tab allows you to attach strong name key files to the assemblies that the.NET projects generate. The Assembly group box contains the following fields:

Assembly group box field	Description
Sign the assembly	Select this check box to enable the "Choose a strong name key file" single line edit box, the browse and New buttons, and the "Delay sign only" check box.
Choose a strong name key file	Name of the key file you want to attach to the generated assembly. This field is associated with a browse (ellipsis) button and a New button. The browse button opens a Select File dialog box where you can select a key file with the .snk extension. The New button lets you create a key file with the .snk extension. PowerBuilder uses the `Sn.exe` tool from the .NET Framework to create the key file.
Delay sign only	Select this check box if your company's security considerations require the step of signing the assembly to be separate from the development process. When this check box is selected, the project will not run and cannot be debugged. However, you can use the strong name tool `Sn.exe` (in the .NET Framework) with the `-Vr` option to skip verification during development.

Assembly group box field	Description
Mark the assembly with AllowPartiallyTrusted-CallerAttribute (.NET Web Service and .NET Assembly projects only)	By default, a strong-named assembly does not allow its use by partially trusted code and can be called only by other assemblies that are granted full trust. However, you can select this check box to allow a strong-named assembly to be called by partially trusted code.

The Sign tab has additional fields for selecting certificate files that you publish with smart client applications.

For information about the Sign tab fields for smart client applications, see *Digital Certificates* on page 131.

Error messages

If you select a strong name key file in either the Assembly or Intelligent Updater group boxes, and the key file is invalid, PowerBuilder displays a message box telling you that the key file is invalid. If the key file you select is password protected, PowerBuilder prompts you to enter the password for the key file. If you enter an incorrect password, a message box informs you that the password you entered is invalid.

ASP.NET Configuration for a .NET Project

You can configure ASP.NET for a Web Forms or smart client 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 or a smart client application on a Web server must have appropriate ASPNET (IIS 5.0), IIS_WPG (IIS 6.0), or IIS_IUSRS (IIS 7.0 and 7.5) user permissions.

Note: You do not need to install IIS on the development computer for PowerBuilder applications or components unless you are using the same computer as a server for Web Forms or smart client applications, or for Web service components. IIS is also not required on end users' computers.

For an example of granting user permissions to a directory, see *Setting Up a SQL Anywhere Database Connection* on page 10.

When you deploy directly to a remote computer, system information about the deployment computer, including its OS and IIS versions, is passed to PowerBuilder through the Windows Management Instrumentation (WMI) interface. Deployment through the WMI interface requires administrator privileges. If you make any changes to administrator accounts on a remote computer, you will probably need to reboot that computer before you can deploy a .NET Web project from PowerBuilder.

If you deploy to an MSI setup file, and run the setup file on a deployment computer, PowerBuilder can use the Windows API to obtain information about the OS and IIS versions on that computer.

IIS Installation

You can install IIS from the Control Panel, but you might need a Windows operating system CD.

On Windows XP, 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. On Vista and Windows 7, go to the Programs and Features page in the Control Panel, select Turn Windows features on or off, and select Internet Information Services.

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

Selecting the Default ASP.NET Version

If you installed multiple versions of the .NET Framework on the target Web server, you should make sure that IIS uses a supported version for PowerBuilder .NET applications.

You can make this change globally, for all ASP.NET Web site applications, or for individual applications that you deploy to IIS.

The following procedure applies to IIS 5 and 6. In IIS 7 and later, set the .NET Framework version for the application pool your applications use. For more information, see *Configuration Requirements for Windows Vista and Later* on page 11.

1. Select **Start** > **Run** from the Windows Start menu.
2. Type `InetMgr` in the Run dialog box list.
3. In the left pane of the IIS Manager, expand the local computer node and its Web Sites sub-node.
4. One of the following:
   - For all new Web sites, right-click the Default Web Site node and select **Properties**.
   - For already deployed projects, expand the Web site node and right-click the .NET application that you deployed from PowerBuilder.
5. Specify the ASP.NET or .NET Framework version.
   - On Windows XP, open the ASP.NET tab and choose the ASP.NET version:
     - For PowerBuilder 12.0 and earlier: `2.0.50727`
     - For PowerBuilder 12.5: `4.0.30319`
   - On Windows 7, Windows Vista, and Windows 2008, set the .NET Framework version used by your Web Forms or NVO Web service deployment:

1. In the IIS Manager, open the Application Pools node underneath the machine node.
2. Right-click the PBDotNet4AppPool filter and choose **Advanced Settings**.
3. Set the .NET Framework Version to `4.0`.

Viewing and Modifying Global Properties in the 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 65.

1. In the left pane of the IIS Manager, expand the nodes 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 (for 12.0 and earlier) or 4.0.30319 (for 12.5) if necessary.
4. Click Edit Configuration.
   The ASP.NET Configuration dialog box for the current .NET Web Forms application appears. You can view its global properties in the list box at the bottom of the General tab.

   Note: 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 .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 48.

The `File\Icon` directory is used by the PowerBuilder 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

Full control permissions are required for directories containing databases that you need to access from your Web Forms applications.

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, or IIS_IUSRS (IIS 7 on Windows Vista and IIS 7.5 on Windows 7) default permissions for the Sybase\Shared and Sybase SQL Anywhere directories, making sure to replace permissions of all child objects in those directories.

Note: 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.

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_IUSRS user group has the right to access them.

1. In Windows Explorer, right-click the `Sybase`, `Sybase\Shared` or `Sybase SQL Anywhere` directory and select Properties from the context menu.
2. Select the Security tab of the Properties dialog box for the directory and click **Add**. On Vista and Windows 7, click **Edit** and then **Add**.

   Note: To show the Security ta of the Select Users, Computers, or Groups dialog box, 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**, choose the server computer name from the Locations dialog box, and click **OK**.
4. Type ASPNET (IIS 5), IIS_WPG (IIS 6), or IIS_IUSRS (IIS 7 and 7.5) 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 check boxes for the access permissions you need under the Allow column in the bottom list box.

   You must select the Full Control check box for a directory containing a database that you connect to from your application.
6. Click **Advanced**.
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 appears, and 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.
   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.

Telerik RadControls

PowerBuilder installs Telerik RadControls for ASP.NET and deploys these controls with your Web Forms applications.

RadControls provide enhanced functionality for Web Forms toolbars and menus, DatePicker and MonthCalendar controls, and TreeView controls.

Configuration Requirements for Windows Vista and Later

When you run PowerBuilder on Windows Vista or Windows 7 under a standard user account, and attempt to deploy Web Forms or Web Service projects, the User Account Control (UAC) dialog box appears. This dialog box allows you to elevate your privileges for the purpose of deployment.

Deploying .NET targets to a remote Windows Vista, Windows 2008, or Windows 7 computer might require changes to the Windows firewall, UAC, or the Distributed Component Object Model (DCOM) settings:

Settings for	Required changes
Windows firewall	Enable exceptions for WMI and file and printer sharing

Settings for	Required changes
UAC (When you are not running PowerBuilder with the built-in Administrator account)	If the development and deployment computers are in the same domain, connect to the remote computer using a domain account that is in its local Administrators group. Then UAC access token filtering does not affect the domain accounts in the local Administrators group. You should not use a local, nondomain account on the remote computer because of UAC filtering, even if the account is in the Administrators group. If the development and deployment computers are in the same workgroup, UAC filtering affects the connection to the remote computer even if the account is in the Administrators group. The only exception is the native “Administrator” account of the remote computer, but you should not use this account because of security issues. Instead, you can turn off UAC on the remote computer. and if the account you use has no remote DCOM access rights, you must explicitly grant those rights to the account.
DCOM	Grant remote DCOM access, activation, and launch rights to a nondomain user account in the local Administrators group of the remote computer if that is the type of account you are using to connect to the remote computer.

Deploying to the default application pool

Virtual directories in IIS 7 and later are hosted in an application pool. An application pool is the host process for one or more Web applications. When you deploy a PowerBuilder Web Forms application to IIS 7 or later, the application is deployed to a PowerBuilder-specific application pool named PBDotnet4AppPool. On 64-bit operating systems, the PBDotnet4AppPool pool is configured to run 32-bit applications.

To avoid compatibility issues with some features, Web Forms applications deployed from PowerBuilder to IIS 7 or IIS 7.5 must run in an application pool that uses the classic managed pipeline mode, where ASP.NET runs as an ISAPI extension. The PBDotnet4AppPool application pool uses the classic managed pipeline mode by default.

Changing Application Pool Identity for IIS 7.5

IIS7.5 includes a new identity type, ApplicationPoolIdentity, and sets it as the default identity for application pools.

On IIS 7.5, PBDotnet4AppPool also uses ApplicationPoolIdentity as its default identity value, but some Web Forms application features—such as creation of permanent user accounts, SSL authentication, and DataWindow **Print** and **SaveAs** commands—fail with this identity. You can avoid these issues by changing the PBDotnet4AppPool identity to NetworkService:

1. In IIS Manager, select Application Pools.
2. From the list of application pools, right-click PBDotnet4AppPool and select Advanced Settings.
3. In the Process Model section, change the identity property from ApplicationPoolIdentity to NetworkService, and click **OK**.
   Changing this setting affects all applications running in the PBDotnet4AppPool application pool.

Creating an Application Pool

Create and configure an application pool to host PowerBuilder Web Forms applications.

1. In IIS Manager, select Application Pools.
2. In the Actions pane, select Add Application Pool.
3. Provide a name, such as PBWebForms, for the application pool.
4. Set .NET Framework version to .NET Framework v4.0.30319.
5. If necessary, set Managed Pipeline Mode to Classic and click **OK**.

Enabling 32-bit Applications on 64-bit Operating Systems

On 64-bit operating systems, you must enable the application pool to run 32-bit applications.

1. In IIS Manager, select Application Pools.
2. In the list of Application Pools, select the application pool you have configured for use with PowerBuilder Web Forms.
3. In the Actions pane, select Advanced Settings under Edit Application Pool.
4. Expand the General settings, set Enable 32-bit Applications to true, and click **OK**.

Moving an Application into a Different Application Pool

If you have created and configured a new application pool for PowerBuilder, you must move your PowerBuilder Web Forms applications into the pool.

1. In IIS Manager, expand Web Sites and Default Web Site.
2. Right-click the virtual directory for your application and click **Advanced Settings**.
3. Select the drop-down list next to the Application Pool property, select the application pool you created, and click **OK**.
4. Reload the application.

Application Directory Permissions

When you deploy a new Web Forms target, a `temp` directory is created in the `Inetpub\wwwroot\`*`application_name`* directory, where *application_name* is the name of your application, and several subdirectories are created in the `Inetpub\wwwroot\application_name_root` directory.

Files are written to and deleted from these directories, therefore the IIS_IUSRS group must have full permissions on `temp` and `application_name_root`.

Checklist for Deployment

Verify that production servers and target computers meet all requirements for running the .NET targets that you deploy from PowerBuilder Classic.

Checklist for all .NET targets

For deployment of all .NET target types (Windows Forms, Web Forms, .NET Assembly, .NET Web Service), production servers or target computers must have:

- The Windows XP SP2, Windows Vista, Windows 2008, or Windows 7 operating system
- .NET Framework 4.0
- The Microsoft Visual C++ runtime libraries `msvcr71.dll`, `msvcp71.dll`, `msvcp100.dll`, `msvcr100.dll`, and the Microsoft .NET Active Template Library (ATL) module, `atl71.dll`
- PowerBuilder .NET assemblies in the global assembly cache (GAC)
- PowerBuilder runtime dynamic link libraries in the system path
  See *Deploying PowerBuilder runtime files* on page 14.

Checklist for .NET Web Forms and Web Service targets

For .NET Web Forms and Web Service targets, production servers must have:

- IIS 5 or later (See *IIS Installation* on page 8)
- ASP.NET (See *Selecting the Default ASP.NET Version* on page 8)
- 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 *ASP .NET user permissions* on page 27.

For information on different methods for deploying .NET Web Forms applications to a production server, see *Deploying to a production server* on page 26. These methods are also valid for deployment of .NET Web Service components.

Deploying PowerBuilder runtime files

The simplest way to deploy PowerBuilder runtime DLLs and .NET assemblies to production servers or target computers is to use the PowerBuilder Runtime Packager tool. The Runtime Packager creates an MSI file that installs the files you select, registers any self-registering DLLs, and installs the .NET assemblies into the global assembly cache (GAC).

Note: When you deploy any PowerBuilder application or component, always make sure that the version and build number of the PowerBuilder runtime files on the target computer or server is the same as the version and build number of the DLLs on the development computer. Mismatched DLLs can result in unexpected errors in all applications. If the development computer is updated with a new build, PowerBuilder .NET applications and components must be rebuilt and redeployed with the new runtime files.

For information on all the steps required to migrate .NET applications and components that you deployed with earlier releases of PowerBuilder, see *Release Bulletin > Migration Information*. PowerBuilder release bulletins are available from links on the Product Manuals Web site at *http://www.sybase.com/support/manuals/*.

For a list of base components deployed when you select PowerBuilder .NET Components in the Runtime Packager, see *Application Techniques > Deploying Applications and Components*. The Runtime Packager installs additional components depending on the options you select in its user interface.

You can also choose to use another tool to install the runtime files on the server or target computer:

File name	Required for

`pbshr125.dll`
`Sybase.PowerBuilder.ADO.dll`
`Sybase.PowerBuilder.Common.dll`
`Sybase.PowerBuilder.Core.dll`
`Sybase.PowerBuilder.Interop.dll`
`Sybase.PowerBuilder.Web.dll`
`Sybase.PowerBuilder.Win.dll`
	All .NET targets

`pbrth125.dll`	.NET Web Forms and ADO.NET

`pbdwm125.dll`
`Sybase.PowerBuilder.Datawindow.Web.dll`
`Sybase.PowerBuilder.DataWindow.Win.dll`
`Sybase.PowerBuilder.Datawindow.Interop.dll`
	DataWindows and DataStores

`pbdpl125.dll`	Data pipelines (Windows Forms only)
`Sybase.PowerBuilder.EditMask.Win.dll` `Sybase.PowerBuilder.EditMask.Interop.dll`	Edit masks

File name	Required for

`Sybase.PowerBuilder.Graph.Web.dll`
`Sybase.PowerBuilder.Graph.Win.dll`
`Sybase.PowerBuilder.Graph.Core.dll`
`Sybase.PowerBuilder.Graph.Interop.dll`
	Graphs

`pbrtc125.dll`
`Sybase.PowerBuilder.RTC.Win.dll`
`Sybase.PowerBuilder.RTC.Interop.dll`
`tp13.dll`
`tp13_bmp.flt`
`tp13_css.dll`
`tp13_doc.dll`
`tp13_gif.flt`
`tp13_htm.dll`
`tp13_ic.dll`
`tp13_ic.ini`
`tp13_jpg.flt`
`tp13_obj.dll`
`tp13_pdf.dll`
`tp13_png.flt`
`tp13_rtf.dll`
`tp13_tif.flt`
`tp13_tls.dll`
`tp13_wmf.flt`
`tp13_wnd.dll`
`tp4ole13.ocx`
	Rich text

`PBXerces125.dll`
`xerces-c_2_6.dll`
`xerces-depdom_2_6.dll`
	XML export and import

File name	Required for
`Sybase.PowerBuilder.WebService.Runtime.dll` `Sybase.PowerBuilder.WebService.RuntimeRemo-teLoader.dll`	Web service Data-Windows

`ExPat125.dll`
`libeay32.dll`
`ssleay32.dll`
`xerces-c_2_6.dll`
`xerces-depdom_2_6.dll`
`EasySoap125.dll`
`pbnetwsruntime125.dll`
`pbsoapclient125.pbx`
`pbwsclient125.pbx`
`Sybase.PowerBuilder.WebService.Runtime.dll`
`Sybase.PowerBuilder.WebService.RuntimeRemo-teLoader.dll`
	Web service clients

`pblab125.ini`	Label DataWindow presentation style
`pbtra125.dll` `pbtrs125.dll`	Database connection tracing

Sybase.PowerBuilder files are strong-named .NET assemblies that can be installed into the GAC. For more information about the GAC, see *Installing assemblies in the global assembly cache* on page 18.

You must also install the database interfaces your application uses:

File name	Required for
`pbin9125.dll`	Informix I-Net 9 native interface
`pbo84125.dll`	Oracle8i native interface
`pbo90125.dll`	Oracle9i native interface
`pbo10125.dll`	Oracle 10g native interface

File name	Required for
`pbsnc125.dll`	SQL Native Client for Microsoft SQL Server native interface
`pbdir125.dll`	Sybase DirectConnect™ native interface
`pbase125.dll`	Sybase Adaptive Server® Enterprise native interface (Version 15 and later)
`pbsyc125.dll`	Sybase Adaptive Server Enterprise native interface

`pbado125.dll`
`pbrth125.dll`
`Sybase.PowerBuilder.Db.dll`
`Sybase.PowerBuilder.DbExt.dll`
	ADO.NET standard interface

`pbjvm125.dll`
`pbjdb125.dll`
`pbjdbc12125.jar`
	JDBC standard interface

`pbodb125.dll` `pbodb125.ini`	ODBC standard interface
`pbole125.dll` `pbodb125.ini`	OLE DB standard interface

Installing assemblies in the global assembly cache

When the Common Language Runtime (CLR) is installed on a computer as part of the .NET Framework, a machine-wide code cache called the global assembly cache (GAC) is created. The GAC stores assemblies that can be shared by multiple applications. If you do not want or need to share an assembly, you can keep it private and place it in the same directory as the application.

If you do not want to use the Runtime Packager to deploy your application, you should use Windows Installer or another installation tool that is designed to work with the GAC. Windows Installer provides assembly reference counting and other features designed to maintain the cache.

On the development computer, you can use a tool provided with the .NET Framework SDK, `gacutil.exe`, to install assemblies into the GAC.

Some assemblies, like `RadCalendar.NET2.dll`, `RadInput.Net2.0.dll`, `RadMenu.Net2.Dll`, `RadToolBar.Net2.dll`, and `RadTreeView.Net2.dll`, are still installed into the .NET Framework 2.0 GAC `windows\assembly` directory.

Assemblies deployed in the global assembly cache must have a strong name. A strong name includes the assembly's identity as well as a public key and a digital signature. The GAC can contain multiple copies of an assembly with the same name but different versions, and it might also contain assemblies with the same name from different vendors, so strong names are used to ensure that the correct assembly and version is called.

For more information about assemblies and strong names, see the Microsoft library at *http://msdn.microsoft.com/en-us/library/wd40t7ad(VS.71).aspx*.

Web Forms Targets

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

PowerBuilder Web Forms Applications

The PowerBuilder 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.

System Requirements for PowerBuilder Web Forms

You must install the .NET Framework 4.0 on the same computer as PowerBuilder. The system PATH environment variable must include the location of the .NET Framework.

You must also install the .NET Framework on the IIS server where you deploy a Web Forms target.

For information about installation and configuration, see *ASP.NET Configuration for a .NET Project* on page 7. For information on migrating Web Forms targets from earlier releases of PowerBuilder, see *Release Bulletin > Migration Information*.

If you are deploying .NET applications as a standard user from a computer with the Vista operating system or later, the User Account Control dialog box prompts you for privilege elevation. This dialog box does not appear if you run PowerBuilder as a computer administrator.

Web Forms Targets

Use the .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.

Web Forms Projects

Whether you use the .NET Web Forms 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 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 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.

This table lists optional and required items in the .NET Web Forms Application project wizard:

Wizard field	Description
Project name	Name of the Web Forms project.
Project library	Library where you want to store the Web Forms project.
Web application name	Name of the Web Forms application. By default, this is the name of the application for the current PowerBuilder target.
Application URL preview	Address for starting the Web Forms application in a browser (minus the `default.aspx` or `default.htm` 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 `bin` directory in the application Web site under the virtual root folder.
JavaScript file list	Specifies JavaScript files you want to deploy with the project.

Wizard field	Description
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 Web Forms application.

The Web Forms project painter

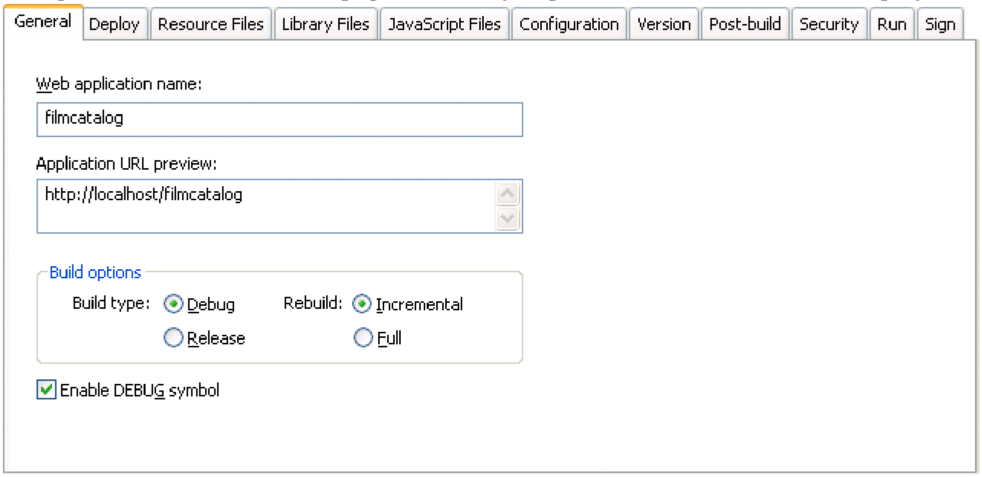
After you click Finish in the project wizard, PowerBuilder creates a Web Forms project and opens the project in the Project painter. The Project painter displays the values you entered in the wizard and allows you to modify them. The painter also includes functionality that is not available in the .NET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

Project tab page	Functionality not available in the .NET Web Forms wizard
General	Includes the following radio button build options: Build Type — Debug (default) or Release. Rebuild — Incremental (default) or Full. 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. For information on the rebuild scope, see *Rebuild Scope* on page 214. The Enable DEBUG Symbol check box enables code inside conditional compilation blocks using the DEBUG symbol. This selection does not affect and is not affected by the project's build type setting. It is selected by default.
Resource Files	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. You can also select a registry XML file that you want to deploy to the `File/Common` directory for your application. For more information on using registry files, see *Registry Functions for Web Forms Applications* on page 30.

Project tab page	Functionality not available in the .NET Web Forms wizard
Library Files	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 if they contain DataWindow or Query objects. 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. 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 context menu, rather than using the PBD file that contains the SoapConnection and SoapError classes.
Configuration	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. For more information about global properties, see *Global Web Configuration Properties* on page 65.
Version	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 appear 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.
Security	Lets you configure CAS security zones for your applications, minimizing the amount of trust required before application code is run by an end user. See *Security Settings* on page 3 and *Custom Permission Settings* on page 231.

Project tab page	Functionality not available in the .NET Web Forms wizard
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.
Sign	The Assembly group box on this tab page allows you to attach strong name key files to the assemblies that your project generates.

This picture shows the General page of the Project painter for a .NET Web Forms project.



Web application name: filmcatalog

Application URL preview: http://localhost/filmcatalog

Build type: [✓ Debug] Release

Rebuild: [✓ Incremental] Full

✓ Enable DEBUG symbol

Web Forms Deployment

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 Web Forms project in the System Tree and select Deploy from its context menu.

The Output window shows 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 a supported version of the Microsoft .NET Framework is the only version of the .NET Framework installed on the server, or if you configured the server to use a supported version

(2.0, 3.0, or 3.5) 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 context menu for the Web Forms project also has a **Run Project** menu item.

Deployment 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 appear 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.

Deployment 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 computer 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 *ASP.NET Configuration for a .NET Project* on page 7. You must install all database clients and have access to all data sources on the production computer. 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 `PBODB125.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`, `msvcp100.dll`, `msvcr100.dll`, `pbshr125.dll`, and if your application uses DataWindow objects, `pbdwm125.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 *Application Techniques > Deploying Applications and Components*.

Deployment 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 computer user's Windows login ID as a member of the Administrators group on the remote computer hosting the IIS server.

If you copy a Web Forms application from a development computer 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.

Note: For information on the directory file structure of a deployed Web Forms application under the IIS virtual root directory (`\inetpub\wwwroot`), see *Web Forms File Manager* on page 56.

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 the version of IIS that your server is running:

IIS version	Commands for granting appropriate user permissions

5	`cacls applicationName\temp /t /e /c /g ASPNET:f`
	`cacls applicationName_root /t /e /c /g ASPNET:f`

6	`cacls applicationName\temp /t /e /c /g IIS_WPG:f`
	`cacls applicationName_root /t /e /c /g IIS_WPG:f`

7 and 7.5	`cacls applicationName\temp /t /e /c /g IIS_IUSRS:f`
	`cacls applicationName_root /t /e /c /g IIS_IUSRS:f`

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 computer, 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 65.

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

When you debug or run the project from PowerBuilder, a system option setting can cause a message box to appear if the application has been modified since it was last deployed. The message box prompts you to redeploy the application, although you can select **No** to debug or run the older application, and you can set the system option to prevent the message box from appearing.

For information about the message box, see *Triggering Build and Deploy Operations* on page 215. For information about the system option, see *System Option* on page 215.

The Web browser that opens when you run a Web Forms project from PowerBuilder 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.

Note: 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 65.

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

You can share the data from primary, delete, and filter buffers of read-only DataWindow objects across Web Forms application sessions.

Sharing DataWindow objects

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 *Viewing and Modifying Global Properties in the IIS Manager* on page 9.

These 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.
- 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.

Sharing DDDW objects

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.

Note: 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 *DataWindow objects and controls*.

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.

Registry Functions for Web Forms Applications

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

Searching and setting registry entries

The **RegistryGet**, **RegistryKeys**, and **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 48.

Rules for registry searching and setting

These 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.

Deploying a `registry.xml` file

A text box at the bottom of the Resource Files tab of a Web Forms project allows you to select a `registry.xml` file for deployment 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.

This 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.

Client-Side Programming

The use of client-side events can improve application performance because they do not require round trips to the server.

Interrupting default event handlers

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.

Note: JavaScript event handlers can enhance performance by allowing users to make modifications without a postback to the server.

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

This example of a customized, client-side JavaScript event handler for the ItemChanged event of a DataWindow 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 *XMLClientSide!*, 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.

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.

This table describes the logic followed by the default handlers that attach to each event, and indicates whether the handler causes a postback, a delayed postback, or no postback.

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 • Clicked, DoubleClicked, and ButtonClicking are handled

Client-side Event	Default JavaScript handler (postback action)	Used under the following conditions for server-side events:
	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
DoubleClicked	PBDataWindow_DoubleClicked (postback)	DoubleClicked 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

Client-side Event	Default JavaScript handler (postback action)	Used under the following conditions for server-side events:
ItemChanged	PBDataWindow_ItemChangedReject (no postback)	ItemChanged is handled
RowFocusChanged	PBDataWindow_RowFocusChanged (postback)	• RowFocusChanging is handled, but ItemFocusChanged is not • RowFocusChanged is handled, but ItemFocusChanged is not

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 Liist of Web DataWindow Client-Side Events* on page 37.

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 this table:

Event	Arguments	Return Codes
ButtonClicked	sender, rowNumber, buttonName	0 – Continue processing

Event	Arguments	Return Codes
ButtonClicking	sender, rowNumber, buttonName	0 – Execute action assigned to button, then trigger ButtonClicked 1 – Do not execute action or trigger ButtonClicked
Clicked	sender, rowNumber, objectName	0 – Continue processing 1 – Prevent focus change
DoubleClicked	sender, rowNumber, objectName	0 – Continue processing 1 – Prevent focus change
ItemChanged	sender, rowNumber, columnName, newValue	0 – Accept data value 1 – Reject data value and prevent focus change 2 – Reject data value but allow focus change
ItemError	sender, rowNumber, columnName, newValue	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	sender, rowNumber, columnName	0 – Continue processing
RButtonDown	sender, rowNumber, objectName	0 – Continue processing 1 – Prevent focus change
RowFocusChanged	sender, newRowNumber	0 – Continue processing
RowFocusChanging	sender, currentRowNumber, newRowNumber	0 – Continue processing 1 – Prevent focus change

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 Liist of Web DataWindow Client-Side Events* on page 37.

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, rowNumber,
   objectName)
   {
    alert ("You clicked the " + objectName +
          " column in row " + rowNumber)
   }
```

Note: Showing an Alert message box for all clicked objects in a DataWindow can prevent data entry or modification.

Alphabetical Liist 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 *Client-Side Programming* on page 31.

ButtonClicked

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

Applies to

Web DataWindow client control

Arguments

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.

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.

Note: The server-side event that posts back to the ButtonClicked client-side event can be triggered by these default event handlers: PBDataWindow_ButtonClicked, PBDataWindow_ButtonClicking, PBDataWindow_Clicked, and PBDataWindow_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.

ButtonClicking

Occurs when the user clicks a button inside a DataWindow object. This event occurs before the ButtonClicked event.

Applies to
Web DataWindow client control

Arguments

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 Data-Window under the pointer when the user clicked.

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.

Note: The server-side event that posts back to the ButtonClicking client-side event can be triggered by these default event handlers: PBDataWindow_ButtonClicked, PBDataWindow_ButtonClicking, PBDataWindow_Clicked, and PBDataWindow_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.

Clicked

Occurs when the user clicks anywhere in a DataWindow control.

Applies to

Web DataWindow client control

Arguments

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row the user clicked.

Argument	Description
objectName	String. The name of the control within the DataWindow under the pointer when the user clicked.

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.

Note: The server-side event that posts back to the Clicked client-side event can be triggered by these default event handlers: PBDataWindow_Clicked and PBDataWindow_DelayedClicked.

Examples

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

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

DoubleClicked

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

Applies to

Web DataWindow client control

Arguments

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row the user double-clicked.

Argument	Description
objectName	String. The name of the control within the Data-Window under the pointer when the user double-clicked.

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.

Note: The server-side event that posts back to the DoubleClicked client-side event can be triggered by these default event handlers: PBDataWindow_DoubleClicked, PBDataWindow_DelayedClicked, and PBDataWindow_DelayedClickedDifferentRow.

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();
}
```

ItemChanged

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.

Applies to

Web DataWindow client control

Arguments

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.

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.

Note: The server-side event that posts back to the ItemChanged client-side event can be triggered by these default event handlers only after a cascade of events occurs: ItemChanged, Clicked, RowFocusChanging, RowFocusChanged, and ItemFocusChanged.

Default postback scripts for this event are PBDataWindow_ItemFocusChanged_AND_ItemChanged and PBDataWindow_ItemFocusChanged_AND_ItemChanged_OR_ItemError. The default event handler PBDataWindow_ItemChangedReject does not cause a postback, and rejects the changed value entered by the user.

ItemError

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.

Applies to

Web DataWindow client control

Arguments

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

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.

Note: 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 PBDataWindow_ItemFocusChanged_AND_ItemError and PBDataWindow_ItemFocusChanged_AND_ItemChanged_OR_ItemError. The default event handler PBDataWindow_ItemError does not cause a postback, and rejects the value entered by the user.

ItemFocusChanged
Occurs when the current item in the control changes.

Applies to
Web DataWindow client control

Arguments

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.

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.

Note: The server-side event that posts back to the ItemFocusChanged client-side event can be triggered by this event handler: PBDataWindow_ItemFocusChanged.

RButtonDown

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

Applies to

Web DataWindow client control

Arguments

Argument	Description
sender	String. Identifier for the client-side control.
row	Number. The number of the row the user right-clicked.

Argument	Description
objectName	String. The name of the control within the Data-Window under the pointer when the user right-clicked.

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.

Note: The server-side event that posts back to the RButtonDown client-side event can be triggered by this event handler: PBDataWindow_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();
}
```

RowFocusChanged

Occurs when the current row changes in the DataWindow.

Applies to

Web DataWindow client control

Arguments

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

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.

Note: The server-side event that posts back to the RowFocusChanged client-side event can be triggered by these default event handlers: PBDataWindow_RowFocusChanged, PBDataWindow_ClickedDifferentRow, and PBDataWindow_DelayedClickedDifferentRow.

Examples

This script in an `.aspx` file shows an alert message when the row focus changes:

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

RowFocusChanging

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.

Applies to

Web DataWindow client control

Arguments

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, indicating 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, indicating that it does not yet exist.

Return codes

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

0 — continue processing.
1 — prevent the focus 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.

Note: The server-side event that posts back to the RowFocusChanging client-side event can be triggered by these default event handlers: PBDataWindow_RowFocusChanged, PBDataWindow_ClickedDifferentRow, and PBDataWindow_DelayedClickedDifferentRow.

Permanent User Accounts

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.

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*.

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.

If you are not using SQL Server Express
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>
```

This 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 this procedure to set up permanent user accounts.

It uses the default ASP.NET membership provider and data source assignment. (A note in the procedure describes a required change to the `web.config` file when you use a nondefault connection string.)

1. Start the ASP.NET SQL Server Setup wizard by entering **aspnet_regsql** from a DOS command prompt window.

   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 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=""/>
```

Note: 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. Verify that the appropriate user or user group (ASPNET for IIS 5, IIS_WPG for IIS 6, or IIS_IUSRS for IIS 7 and IIS 7.5) 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, or the IIS_IUSRS user group (Windows Vista and later) does not have write authority for the Web Forms virtual root directory.

   If you proceed to the next step without modifying write permissions, you may 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

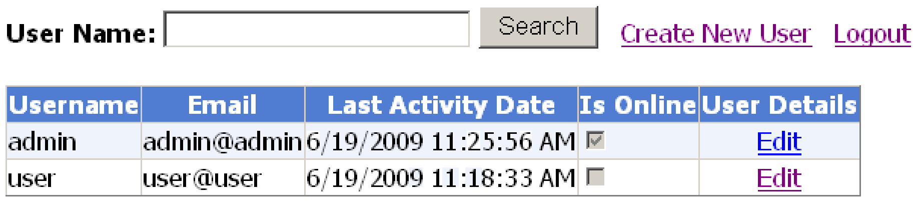
   The `UsersInit.aspx` returns a success page after the above user accounts are created.

   Note: 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 opens.

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, in the page to manage users, click **Search**.
   The page for managing users displays the application users in the permanent user database.



Username	Email	Last Activity Date	Is Online	User Details
admin	admin@admin	6/19/2009 11:25:56 AM	✓	Edit
user	user@user	6/19/2009 11:18:33 AM	☐	Edit

9. Click the **Create New User** hyperlink.
   The page for adding users opens.
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. If the new user should have administrative privileges, select the Admin Role check box and click **Finish.**
    When you return to the page for managing users and click **Search** 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.

Note: 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.

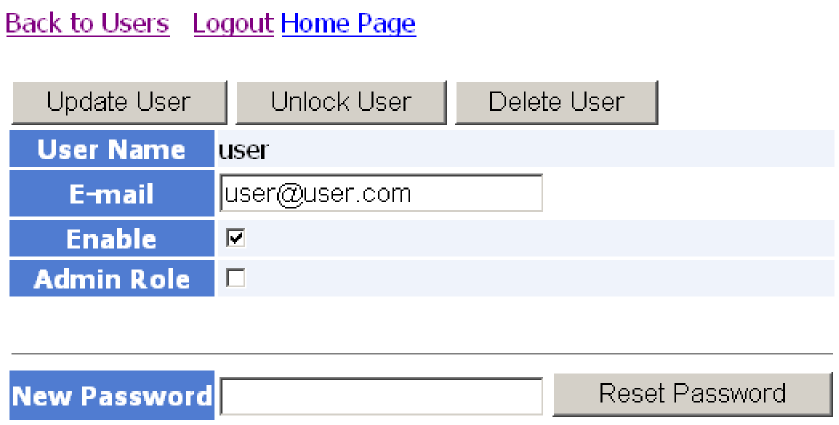
Managing Permanent User Accounts

In an 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 48.

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 48.
3. Click the **Users** hyperlink, then in the page to manage users, click **Search**.
   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 appears. This page also includes an Unlock User button 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.



User Name	user
E-mail	user@user.com
Enable	✓
Admin Role	☐
New 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.

Managers in Web Forms Applications

You can enable print, file, mail profile, and theme managers in a Web Forms application.

The managers provide application users with the ability to print documents, perform file operations, send e-mail, or change the appearance of controls in the application.

Web Forms Print Manager

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

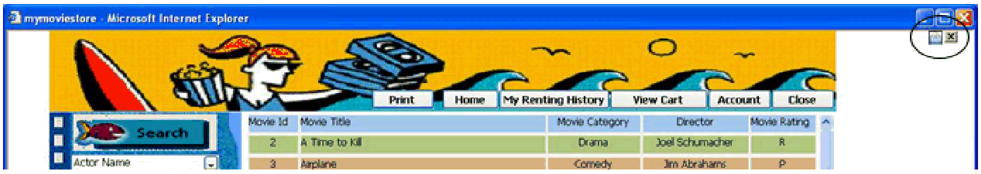
You can use the DataWindow control’s **Print** method to print a DataWindow object to a PDF file. Application users can open the PDF file in a separate browser instance by selecting the print result in the Print Manager. They can then print the PDF file using the **File > Print** menu of the browser.

You can also use the **SaveAs** method to print DataWindows and their data as PDF or XSL files. These files are not visible in the Print Manager. However, you can call the **DownloadFile** function (in a conditional compilation block) to download these files, or application users can download them from the server using the Web Forms File Manager and then print them from a local browser or Adobe Reader application.

See *DownloadFile* on page 78. For information on the File Manager, see *Web Forms File Manager* on page 56.

Print Manager icon

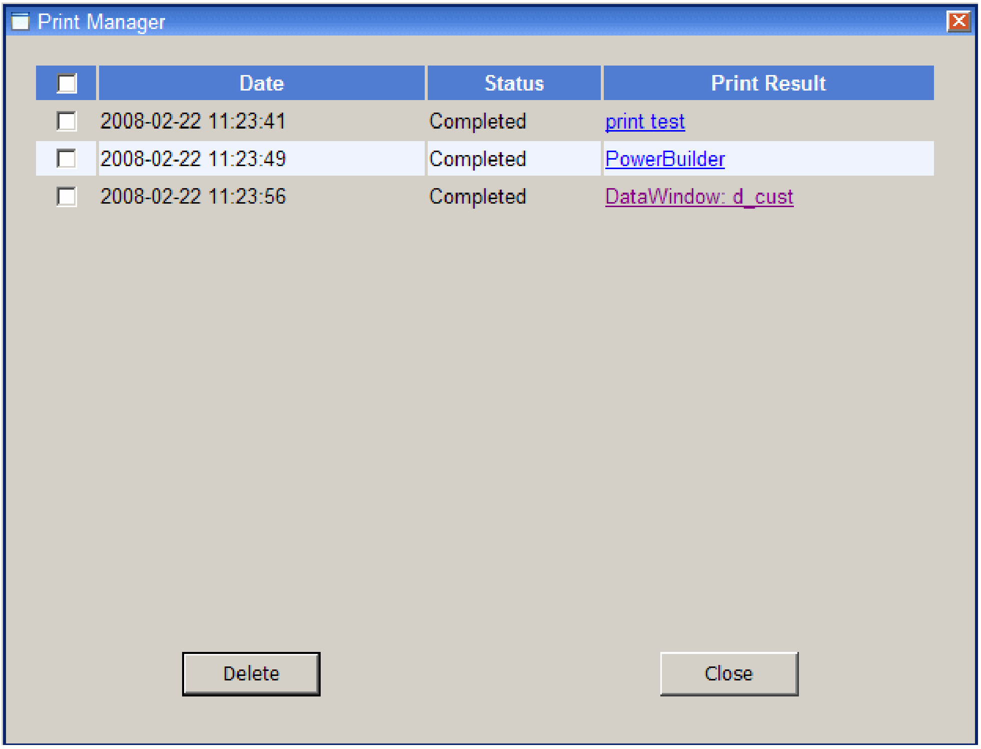
When supported print functions are used to print text in a Web Forms application, a printer icon appears in the right-top corner of the main browser window.



Movie Id	Movie Title	Movie Category	Director	Movie Rating
2	A Time to Kill	Drama	Joel Schumacher	R
3	Airplane	Comedy	Jim Abrahams	P

The application user can 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.

This picture shows the Print Manager with hyperlinks to printed files:



Print Manager

☐	Date	Status	Print Result
☐	2008-02-22 11:23:41	Completed	print test
☐	2008-02-22 11:23:49	Completed	PowerBuilder
☐	2008-02-22 11:23:56	Completed	DataWindow: d_cust

If you do not want the Print Manager icon to appear 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.

See *HasPrintManager* on page 77and *OpenPrintManager* on page 83.

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 cdxgel554rkxxsbn1221uh55. 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_IUSRS 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 these ways:

- Click **Have Disk** 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.

See *Installing GPL Ghostscript* on page 55.

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 `pbvm125.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.

These 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.

Note: 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.

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*.

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://pages.cs.wisc.edu/~ghost*.
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.

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

- The Windows registry.

- The relative path of the `pbdwm125.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 `pbdwm125.dll` file, they must be installed in this directory structure:

```
dirname\pbdwm125.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 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.

The generated files are saved 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.

Note: 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.

Web Forms File Manager

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.

Virtual file system

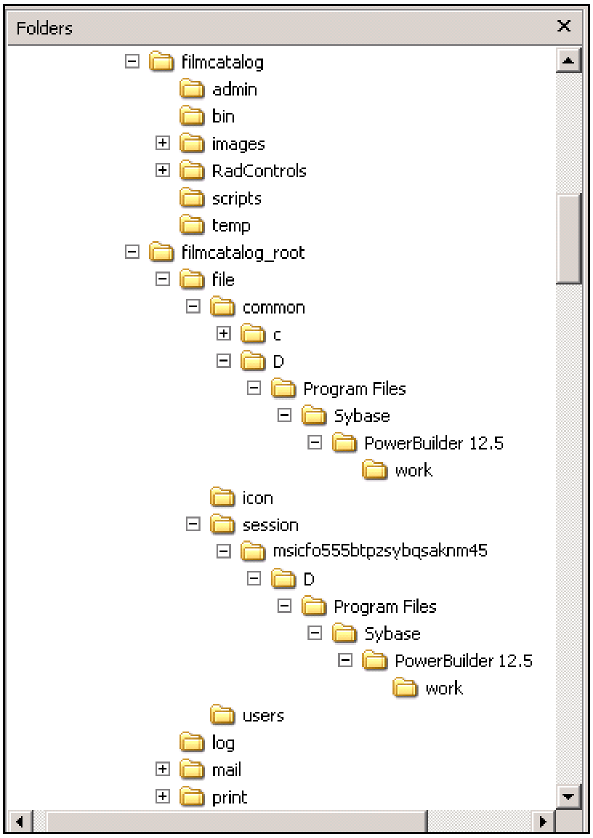
The virtual file system is maintained on the server. Application users can read from and write to files in the virtual file system as long as user permissions for file operations are not restricted.

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

The virtual file system for a PowerBuilder 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.

This figure 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:



```
Folders
⊟ filmcatalog
     admin
     bin
  ⊞ images
  ⊞ RadControls
     scripts
     temp
⊟ filmcatalog_root
  ⊟ file
    ⊟ common
      ⊞ c
      ⊟ D
        ⊟ Program Files
          ⊟ Sybase
            ⊟ PowerBuilder 12.5
                 work
       icon
    ⊟ session
      ⊟ msicfo555btpzsybqsaknm45
        ⊟ D
          ⊟ Program Files
            ⊟ Sybase
              ⊟ PowerBuilder 12.5
                   work
       users
     log
  ⊞ mail
  ⊞ print
```

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.

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 this 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.

File Manager icon

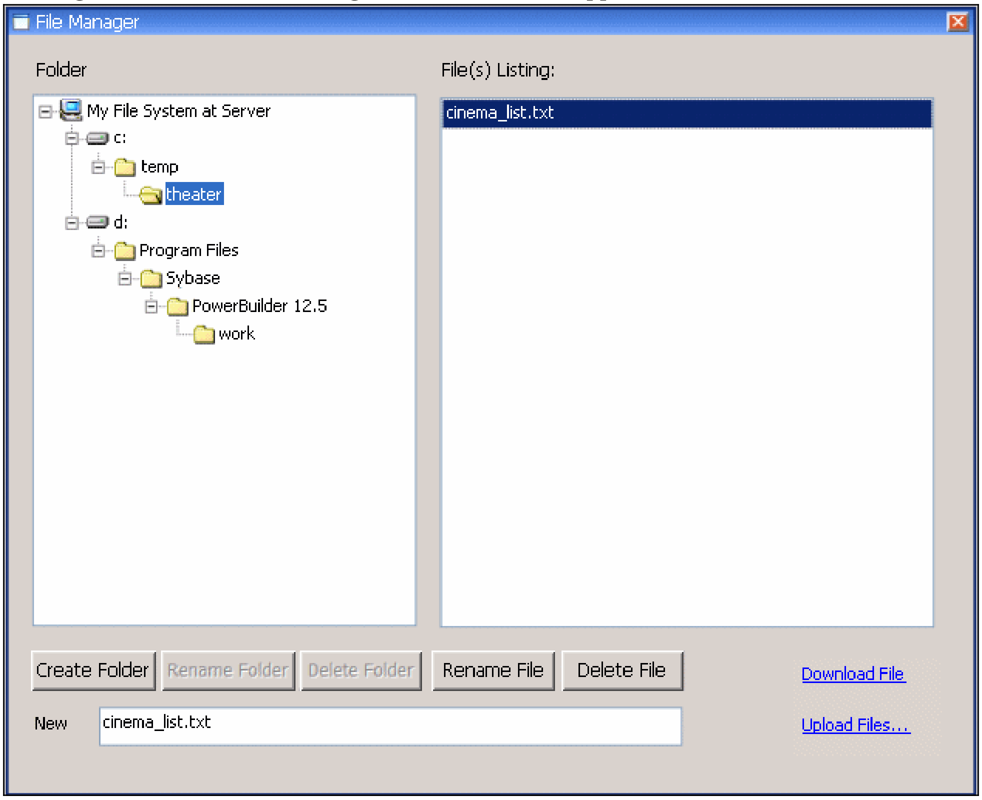
When you set the PBFileManager property to true, the File Manager icon normally displays in every window of your Web Forms application. Users can open the File Manager at any time by clicking the File Manager icon. You can also code an application event to open the File Manager by calling the **OpenFileManager**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

See *HasFileManager* on page 75 and *OpenFileManager* on page 82.

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.

This figure shows the File Manager for a Web Forms application:



Creating a Directory with the File Manager

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.

1. In the File Manager, 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.)

Note: 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.

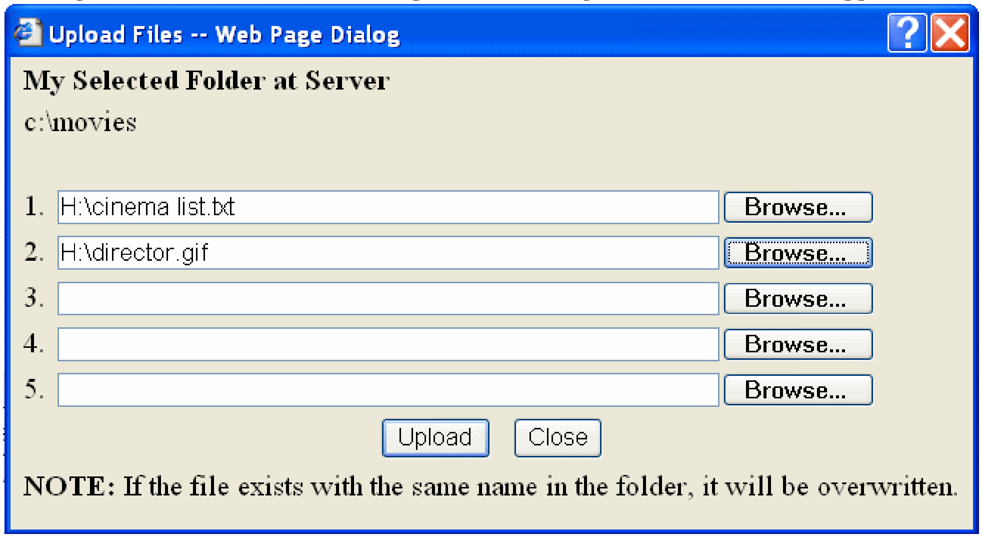
Users can close the File Manager and return to the current Web Forms window by clicking the close (**x**) button at the upper right corner of the manager frame.

Uploading Files with the File Manager

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.

This figure shows the PowerBuilder Upload File dialog box for a Web Forms application:



My Selected Folder at Server

c:\movies

1. H:\cinema list.txt
2. H:\director.gif
3.
4.
5.

NOTE: If the file exists with the same name in the folder, it will be overwritten.

1. In the left pane of the File Manager, select the directory where you want to copy a file.
2. Click the **Upload File** link.
3. In the PowerBuilder Upload File dialog box, type the file name or browse to the file or files you want to upload.
4. Click **Upload**.
   A message in the dialog box indicates if 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 Files with the File Manager

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.

1. From the right pane of the File Manager, select the file you want to download.
   The Download File link appears 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**.
4. In the Save As dialog box, browse to the path on the local computer where you want to save the file, and click **Save**.
   The Download Complete dialog box appears. Its appearance depends on the client operating system. It typically prompts the user to open the downloaded file, open the folder where the file was saved, or close the dialog box.
5. Click **Close** to close the Download Complete dialog box and return to the File Manager.

Web Forms 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 appears 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.

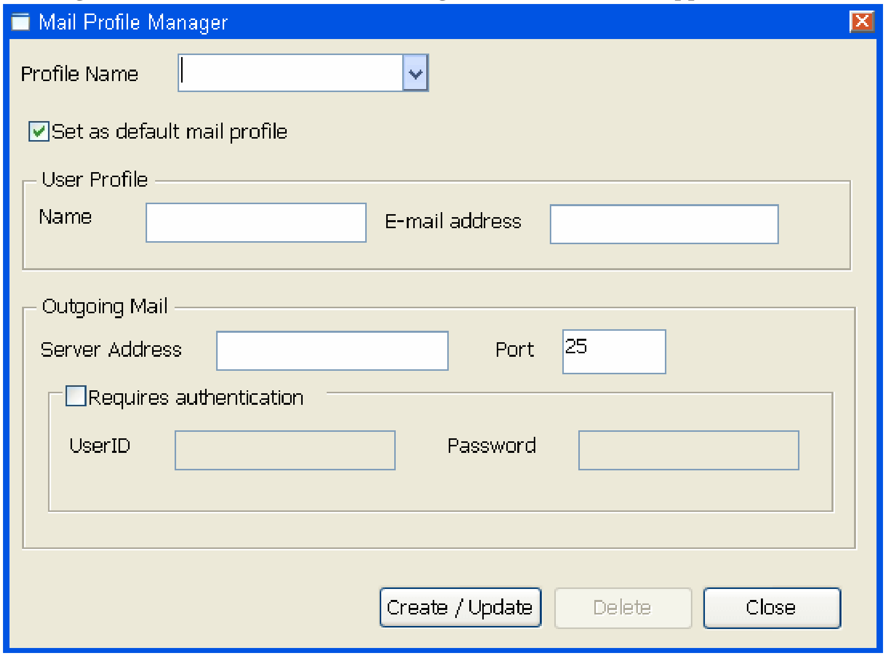
You can also code an application event to open the Mail Profile Manager by calling the **OpenMailManager** function.

See *HasMailManager* on page 76 and *OpenMailManager* on page 82.

The Mail Manager icon appears 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.

Note: The Mail Profile Manager appears 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.

This figure shows the Mail Profile Manager for a Web Forms application:



Port: 25

Set as default mail profile [✓]

The Mail Profile Manager is divided into sections for user profile information and outgoing mail parameters, incoming mail parameters, and account type. 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.

This table lists and describes the fields in the Web Forms Mail Profile Manager:

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.

Section	Field	Description
Outgoing Mail	Server address	Address for the outgoing mail server, such as smtp.sybase.com.
	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.

Mail profile management

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.

Note: 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 *Permanent User Accounts* on page 47.

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.

Required modifications 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 *Restrictions on Supported Controls* on page 89.

Note: 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.

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 appear 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 65.

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 appear on a specific window in your application, you can set the HasThemeManager property for that window to false.

See *HasThemeManager* on page 77 and *OpenThemeManager* on page 83.

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.

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, and several built-in control properties that are not valid for other types of PowerBuilder targets.

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.

Global Web Configuration Properties

Global properties are properties that you set at design time on the Configuration tab of the Project painter, or after deployment in the generated `Web.Config` file for your application.

You cannot set global properties in script.

This table lists global properties of a PowerBuilder .NET Web Forms application that you can set on the Configuration tab before you deploy the application:

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.
PBCachedAndSharedDDDWs	—	A comma-delimited set of names for DataWindow objects that you want to use in DropDownDataWindow edit style controls for sharing across application sessions. See *Sharing Data Across Sessions* on page 29.

Property	Default value	Description
PBCachedAndSharedDWs	—	A comma-delimited set of names for DataWindow objects that you want to share across application sessions. See *Sharing Data Across Sessions* on page 29.
PBCommandParm	—	Sets command line parameters for your application. Users can override the default by setting this property in a URL.
PBCultureSource	Server	Enumeration that specifies the source of regional settings for data formats. Values are Server or Client. See *Use regional formats based on client or server settings* on page 205 .
PBDataWindowEnableDDDW	False	Indicates whether to render a Drop-DownDataWindow (DDDW) or a DropDownListBox 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.
PBDataWindowGoToButtonText	Go	Sets the label for a navigation bar button that takes a user to a designated DataWindow page.
PBDataWindowGoToDescription	Go To:	Sets the label for the navigation control that takes a user to a designated DataWindow page.

Property	Default value	Description
PBDataWindowNavigationBarPosition	PBDWBottom	Sets the position where the page navigation controls appear. Values are PBDWBottom to show them at the bottom of the DataWindow, PBDWTop to show them at the top of the DataWindow, or PBDWTopAndBottom to show them at the top and bottom of the DataWindow.
PBDataWindowPageCountPerGroup	10	Sets a limit to the number of pages that can appear for the Numeric or NumericWithQuickGo style navigation bars.
PBDataWindowPageNavigatorType	NextPrev	Values are NextPrev, Numeric, QuickGo, NextPrevWithQuickGo, or NumericWithQuickGo. See *Take Advantage of Global Configuration Properties* on page 207.
PBDataWindowQuickGoPageNavigatorType	DropDownList	Sets the type of control to use for the Quick Go navigation bar. Values are DropDownList or Edit.
PBDataWindowRowsPerPage	20	Sets the number of DataWindow rows to show 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.
PBDataWindowScriptCallbackDDDW	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
PBDataWindowStatusInfoFormat	Page {C} of {T}	Sets the text 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, which you can 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.

Property	Default value	Description
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 appear with Windows Classic themes for Windows 2000 operating systems, and Windows XP themes for all other operating systems. Select "XP" to show controls with Windows XP themes regardless of the client operating system. Select "Classic" to show controls in all client browsers with a Windows Classic appearance.
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 semicolon-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.

Property	Default value	Description
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	vm.out	Name of the file that logs information about the JVM for applications using a JDBC connection. By default, this file is saved to the `applicationName_root\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.

Property	Default value	Description
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
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.

Property	Default value	Description
PBTraceFileName	PBTrace.log	Name of the file that logs exceptions thrown by the Web Forms application. By default, this file is saved to the `applicationName_root\Log` 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. See *File process mode* on page 58.
PBWindowDefaultHeight	600 (pixels)	Specifies the default height of the client area of the Web browser when MDI, MDIHelp, and main type windows are opened as maximized for the first time.
PBWindowDefaultWidth	1003 (pixels)	Specifies the default width of the client area of the Web browser when MDI, MDIHelp, and main type windows 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:
		`do while flag`
		`   yield()`
		`   loop`

The default global properties and their values appear only when the “System defined configuration settings” option is selected on the Configuration tab. (This is the default selection):

Additional global properties are described in the following table, and are included in the `Web.config` file that is generated in the main application directory under the IIS virtual root directory. Although you cannot set these global properties on the Configuration tab, you can change them in the `Web.Config` file after deployment. However, the selection that you make for Web Application Name on the General tab affects default values generated for all of these global properties except PBPostbackType:

Property	Default value	Description
FileFolder	*WebAppDir*..\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 *sessionID* directory is created under the `File\Session` directory that mirrors the `File\Common` directory structure and file contents.
MailFolder	*WebAppDir*..\appName_root\mail	Base directory for the mail manager.
PrintFolder	*WebAppDir*..\appName_root\print	Base directory for files that your application prints in PDF format.

Property	Default value	Description
LogFolder	`*WebAppDir*..\*appName*_root\log`	Folder that contains the `PBTrace.log` file.

For information on modifying global properties in the `Web.config` file (after you deploy a Web Forms application), see *Viewing and Modifying Global Properties in the IIS Manager* on page 9.

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

Creating Custom Global Properties

Create custom global properties for a Web Forms project from the Configuration tab page of the Project painter.

1. On the Configuration tab, select the **Custom defined configuration settings** option to enable the Add button.
2. Click **Add**.
3. Use the Add User Defined Configuration Setting 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.

Next

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. Click **Edit** to change the value of a custom global property. Click **Delete** to remove a custom global property and its value.

AutoPostBack

Reduce postbacks and improve performance by setting the AutoPostBack property for certain controls to false.

Applies to

CheckBox and RadioButton controls

Usage

In scripts, surround the AutoPostBack property in a conditional compilation code block for Web Forms applications:

```
#IF DEFINED PBWEBFORM THEN
   cbx_1.AutoPostBack = false
#END IF
```

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.

Embedded

Set the Embedded property to true to use the IFRAME element for a Web page defined in the URL property of a StaticHyperLink control. This causes the Web page to appear inline on the Web Forms page (window) containing the StaticHyperLink control.

Applies to

StaticHyperLink controls

Usage

In scripts, surround the Embedded property in a conditional compilation code block for Web Forms applications:

```
#IF DEFINED PBWEBFORM THEN
   shl_1.Embedded = true
#END IF
```

If you place the above code in the Open event for a window containing the StaticHyperLink control, or in the control’s Constructor event, the hyperlink text does not appear, but the page referenced in its URL property opens in the area defined by the control in the Web Forms page.

When you enable the Embedded property, you must consider enlarging the size of the StaticHyperLink control to permit adequate viewing of the embedded Web page, although at runtime, the IFRAME element that replaces the control includes horizontal and vertical scroll bars if the page size exceeds the size of the original control.

Some Web sites use JavaScript code to make sure their pages display as top level HTML windows. This can cause JavaScript errors and erratic behavior when Embedded is set to true.

HasFileManager

Set this property to false to hide the File Manager on a particular page in a Web Forms application.

Applies to

Window controls

Usage

In scripts, surround the HasFileManager property in a conditional compilation code block for Web Forms applications:

```
#IF DEFINED PBWEBFORM THEN
```

```
      w_mywindow.HasFileManager = false
   #END IF
```

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 appear 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 show the File Manager icon unless you set the HasFileManager for a particular window to false.

Note: In MDI applications, manager icons appear 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.

HasMailManager

Set this property to false to hide the Mail Profile Manager on a particular page in a Web Forms application.

Applies to

Window controls

Usage

In scripts, surround the HasFileManager property in a conditional compilation code block for Web Forms applications:

```
   #IF DEFINED PBWEBFORM THEN
      w_mywindow.HasMailManager = false
   #END IF
```

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 appear 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 show the Mail Profile Manager icon unless you set the HasMailManager for a particular window to false.

Note: In MDI applications, manager icons appear 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.

HasPrintManager

Set this property to false to hide the Print Manager on a particular page in a Web Forms application.

Applies to
Window controls

Usage
In scripts, surround the HasPrintManager property in a conditional compilation code block for Web Forms applications:

```
#IF DEFINED PBWEBFORM THEN
   w_mywindow.HasPrintManager = false
#END IF
```

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.

See *Web Forms Print Manager* on page 52.

By default, the HasPrintManager property is set to true. When you activate the Print Manager, all window forms show the Print Manager icon unless you set the HasPrintManager for a particular window to false. The Print Manager icon gives users access to a print manager for files on the Web Forms server.

Note: In MDI applications, manager icons appear 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.

HasThemeManager

Set this property to false to hide the Theme Manager on a particular page in a Web Forms application.

Applies to
Window controls

Usage
In scripts, surround the HasThemeManager property in a conditional compilation code block for Web Forms applications:

```
#IF DEFINED PBWEBFORM THEN
   w_mywindow.HasThemeManager = false
#END IF
```

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 appear on all window forms in your Web Forms applications. The Theme Manager icon gives users to change the appearance 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 show the Theme Manager icon unless you set the HasThemeManager for a particular window to false.

Note: In MDI applications, manager icons appear 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.

System Functions for .NET Web Forms

System functions specific for .NET Web Forms applications allow you to open the various managers for Web Forms applications, obtain global configuration settings, download files for viewing or printing by the application user, and 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.

See *Managers in Web Forms Applications* on page 51.

DownloadFile

Use to download a file from the Web server to a client computer.

Syntax

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

Parameters

- **serverFile –** A string containing the name of the file on the application's virtual file path on the Web server.
- **open –** A boolean that determines whether to access the file in open mode or download mode. Values are:

    true — show the file directly in a browser window (open mode).

false — show a dialog box that lets the user open the file, save the file, or cancel the download operation (download mode).

Returns

None.

Examples

- **–** This 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 appear, giving the user the choice of opening the file, saving the file, or cancelling the operation. The File Download dialog box shows the file name, the file type, the number of bytes in the file, and the name of the server that hosts the file.
- **–** This 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_filenum

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
```

Usage

Some types of files cannot be viewed directly in a browser window. For these types of files, the *open* argument is disregarded. Instead, the File Download dialog box appears, 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** PowerScript function to make sure the file exists in the server directory before you call **DownloadFile**.

GetConfigSetting

Use to return the value of a global configuration property.

Syntax

```
string GetConfigSetting (string key)
```

Parameters

- **key** – A string for the name of a global property in the <appSettings> section of the Web.Config file.

Returns

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

Examples

- **–** This 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"
#end if
```

GetDownloadFileURL

Use to return the URL for a file on the Web server.

Syntax

```
string GetDownloadFileURL (string serverFile, boolean open)
```

Parameters

- **serverFile –** A string containing the name of the file on the application's virtual file path on the Web server.
- **open –** A boolean that determines whether to access the file in open mode or download mode. Values are:

  true — show the file directly in a browser window (open mode).

  false — show a dialog box that lets the user open the file, save the file, or cancel the download operation (download mode).

Returns

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

Examples

- **–** This 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
```

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.

MapVirtualPath

Use to return the actual path of a file on the Web Forms server.

Syntax

`string` **MapVirtualPath** `(string` *`virtualPath`*`)`

Parameters

- **virtualPath** – `A string for a virtual path on the Web Forms server`

Returns

String. Returns the actual path of a file in the virtual file system on a Web Forms server.

Examples

- **–** This code returns the actual path on a new line in a MultiLineEdit control:

```
#if defined PBWEBFORM then
    mle_1.text +=''~r~nActual Path='' &
       + MapVirtualPath(''c:\a.txt'')
#end if
```

Usage

Use the **MapVirtualPath** function to get the actual path of files for file operations required by .NET DLLs.

OpenFileManager

Use to open the Web Forms File Manager.

Syntax

```
void OpenFileManager ()
```

Returns

None.

Examples

- **–** Use this code to open the Web Forms File Manager:

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

Usage

See *Web Forms File Manager* on page 56.

OpenMailManager

Use to open the Web Forms Mail Profile Manager.

Syntax

```
void OpenMailManager ()
```

Returns

None.

Examples

- **–** Use this code to open the Web Forms File Manager:

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

Usage

See *Web Forms Mail Profile Manager* on page 61.

OpenPrintManager

Use to open the Web Forms Print Manager.

Syntax

```
void OpenPrintManager ()
```

Returns

None.

Examples

- **–** Use this code to open the Web Forms Print Manager:

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

Usage

See *Web Forms Print Manager* on page 52.

OpenThemeManager

Use to open the Web Forms Theme Manager.

Syntax

```
void OpenThemeManager ()
```

Returns

None.

Examples

- – Use this code to open the Web Forms Theme Manager:

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

Usage

See *Web Forms Theme Manager* on page 64.

UploadFiles

Use to open the Upload Files dialog box that enables an application user to upload files from the local computer to the Web server.

Syntax

```
void UploadFiles (string serverFolder, long bgColor, int fileNum,
boolean showServerFolder, string description, string allowExts{,
string callbackFunctionName}{, PowerObject po})
```

Parameters

- **serverFolder –** The folder on the server to which you want to copy one or more files from the client computer. PowerBuilder creates this folder under the server virtual root in the *applicationName*_root\session\sessionID directory, or for someone logged in as a permanent user, in the *applicationName*_root\users\userName directory.
- **bgColor –** A long for the background color of the Upload Files dialog box.
- **fileNum –** An integer for the number of text boxes to show in the Upload Files dialog boxes. Application users can upload as many files as there are text boxes in a single upload operation.
- **showServerFolder –** A boolean specifying whether to show the server folder name in the Upload Files dialog box. Values are:

  true — the server folder name.

  false — do not show the server folder name.
- **description –** Text that you want to appear near the top of the Upload Files dialog box. You can use an empty string if you do not want to show additional text in this dialog box.
- **allowExts –** 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.
- **callbackFunctionName –** (Optional) The callback function that lets you know whether the file is correctly uploaded to the Web server.

- **po –** (Optional) The name of a PowerBuilder object that has the callback function set in the *callbackFunctionName* argument.

Returns

None.

Examples

- **–** This 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 show the server directory name in the Upload Files dialog box, but does show 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, shows 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
```

Usage

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.

Unsupported Features in Web Forms Projects

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, ProfileString, and StripRTF expression functions or the aggregate expression functions. User-defined expression functions are also not supported in Web Forms applications.

  Note: 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 — controls you can add to a DataWindow are not all supported in Web Forms applications. The Oval, RoundRectangle, 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 *Controls in DataWindow objects* on page 100.
- 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.

  See *Take Advantage of Global Configuration Properties* on page 207 and *DataWindow objects and controls* on page 103.
- 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.)
- DataWindow gradient and tooltip properties — the DataWindow gradient and tooltip properties introduced in PowerBuilder 11.5 are not supported in Web Forms applications.
- RichText column events and functions — the DataWindow RichText column events and functions introduced in PowerBuilder 11.5 are not supported in Web Forms applications.

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. These values of the MailReturnCode enumeration are not supported in Web Forms applications:

MailReturnAccessDenied
MailReturnDiskFull
MailReturnInsufficientMemory
MailReturnInvalidMessage
MailReturnMessageInUse
MailReturnNoMessages
MailReturnTextTooLarge
MailReturnTooManyFiles
MailReturnTooManyRecipients
MailReturnTooManySessions

PBNI feature

You can use the built-in Web services client extension (`pbwsclient125.pbx`) in applications that you plan to deploy to .NET. You cannot use any other 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 *Datatype Mappings* on page 179 for the list of datatype mappings from .NET to PowerBuilder.

Unsupported Objects

Some PowerScript objects cannot be used in applications deployed to ASP.NET.

Category	Objects
Data reproduction	Pipeline
EAServer integration	RemoteObject
Menu	MenuCascade
OLE	All OLE objects
PBNI extensions	PBDOM
Profiling and tracing	ProfileCall, ProfileClass, ProfileLine, ProfileRoutine, Profiling, TraceFile, TraceTree, TraceTreeNode and descendants, and TraceActivityNode and descendants
Timing	Timing
Tablet PC	InkEdit and InkPicture

Note: 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

Some PowerScript system functions cannot be used in applications deployed to ASP.NET.

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

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

Partially supported system functions

- **IsNull** — 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** — 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** — 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. See *Registry Functions for Web Forms Applications* on page 30.

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.

Unsupported functions, events, and properties

This table lists functions, events, and properties that are not supported on any control:

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, and 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 Accessible-Name AccessibleRole DragAuto DragIcon IM-EMode

Additional unsupported functions, events, and properties by control

This table lists the functions, events, and properties that are not supported on some individual objects or controls. It does not include the items listed in the table under *Unsupported functions, events, and properties* on page 89 .

The entry "No additional" in this table indicates that all items except those listed in the previous table are supported for that 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, Trans-parent
ClassDefinition	No additional	No additional	LibraryName, Varia-bleList (supported, but the sequence of varia-bles might differ in .NET applications)
DataStore	Same as DataWindow control	Destructor, Error, ItemChanged, Prin-tEnd, PrintPage, Print-Start	No additional

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
DataWindow control	AcceptText CopyRTF, PasteRTF Find, FindNext GenerateHTMLForm GenerateResultSet GetStateStatus GetRichTextmethod GetText ImportClipboard InsertDocument LineCount OLEActivate Position PrintCancel ReplaceText ResetInk SaveInk functions Scroll Selected functions SelectText functions SetActionCode SetCultureFormat SetDetailHeight SetFocus SetRedraw SetRichTextmethod SetText ShowHeadFoot TextLine	EditChanged GetFocus LoseFocus PrintEnd PrintMarginChange PrintPage PrintStart RichTextCurrentStyleChanged RichTextLimitError RichTextLoseFocus ScrollHorizontal ScrollVertical (The Clicked event is not triggered on editable text columns that already have focus. See *Partially supported control events* on page 112.	ControlMenu HSplitScroll Icon LiveScroll MaxBox MinBox Resizable RichTextToolbarActivation RightToLeft Title TitleBar

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
DataWindow object (See *DataWindow support* on page 86 for a list of controls that are not supported in a DataWindow object. See *Controls in DataWindow objects* on page 100 for unsupported properties of controls that you can place in a DataWindow object.)	Has no functions	Has no events	*Bandname*.Gradient.*Property Bandname*.Height.Autosize *Bandname*.Pointer *Bandname*.Text Brushmode Grid.ColumnMove Header.#.Suppress Help.Property HorizontalScrollProperty HideGrayLine Label.Ellipse_Property Label.Shape (support rectangle shape only) OLE.Client Picture.Property Pointer Print.Preview.Property Retrieve.AsNeeded RichText.Property Row.Resize Sparse Storage.Property Transparency Tree.Property VerticalScrollProperty Zoom
DataWindowChild	Same as DataWindow control	Has no events	No additional
DatePicker	GetCalendar	Clicked CloseUp DoubleClicked DropDown UserString ValueChanged	AllowEdit DropDownRight RightToLeft ShowUpDown TodaySection WeekNumbers
DropDownListBox, DropDownPictureListBox	DirList (supported, but can only include files from the virtual file system) Position ReplaceText SelectedLength SelectedStart SelectedText SelectText	DoubleClicked	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 GetDataLabelling GetDataTransparency GetSeriesLabelling GetSeriesTransparency ImportClipboard ImportFile SaveAs SetDataLabelling SetDataTransparency SetFocus SetSeriesLabelling SetSeriesTransparency	No additional	FocusRectangle Render3D
HProgressBar	No additional	DoubleClicked	SmoothScroll
HScrollBar	No additional	RButtonDown	No additional
HTrackBar	SelectionRange	No additional	SliderSize TickFrequency TickMarks
ListBox	DirList (supported, but can only include files from the virtual file system) SetTop Top	DoubleClicked	DisableNoScroll ExtendedSelect RightToLeft TabStop

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
ListView	Arrange EditLabel FindItem (partial support: see *Control functions with partial support* on page 110) 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 DateRecieved 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 87.)	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. See *Partially supported control events* on page 112.)	BitmapBackColor BitmapGradient MenuAnimation MenuBitmaps MenuImage MenuTitles MenuTitleText MergeOption MicroHelp TitleBackColor TitleGradient ToolbarAnimation ToolbarItemDown ToolbarItemDownName ToolbarItemSpace
MonthCalendar	GetDisplayRange	Clicked DoubleClicked	AutoSize MaxSelectCount RightToLeft 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	DirList (supported, but can only include files from the virtual file system) 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 99) 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
SimpleTypeDefinition	No additional	No additional	LibraryName
SingleLineEdit	No additional	RButtonDown	AutoHScroll HideSelection
StaticHyperLink	No additional	No additional	FillPattern FocusRectangle RightToLeft

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
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 inTabsOnLeft style) TabPosition (Enum values supported for TabsOnTop and TabsOnLeft only)

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
TreeView	AddStatePicture DeleteStatePicture DeleteStatePictures EditLabel GetItemAtPointer SetDropHighlight SetFirstVisible SetLevelPictures SetOverlayPicture	BeginDrag BeginLabelEdit BeginRightDrag EndLabelEdit Key Notify RightDoubleClicked Sort	DeleteItems DisableDragDrop EditLabels FullRowSelect HasButtons (False value unsupported for RadControl TreeView; supported in IE Web Control TreeView) HideSelection Indent (Unsupported for RadControl TreeView; supported in IE Web Control TreeView) LayoutRTL LinesAtRoot PictureHeight PictureWidth RightToLeft SingleExpand StatePictureHeight StatePictureWidth TrackSelect
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

Supported object or control	Unsupported functions	Unsupported events	Unsupported properties
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
VTrackBar	SelectionRange	No additional	SliderSize TickFrequency TickMarks
Window	DDE functions GetToolbar GetToolbarPos InputField functions SetMicroHelp SetToolbar SetToolbarPos	DDE events Deactivate DoubleClicked Hide Key Mouse events SystemKey ToolbarMoved	Border, ClientEdge ColumnsPerPage ContextHelp HScrollbar KeyboardIcon LinesPerPage PaletteWindow Resizable RightToLeft TitleBar Toolbar properties (except ToolbarVisible) Transparency 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

This table lists the properties that are not supported in Web Forms applications for controls that you can place in a DataWindow object:

Control in DataWindow	Unsupported properties
Button	AccessibleProperty, Background.Brushmode, Background.Gradient.Property, Background.Transparency, Font.Escapement, Font.Width, HideSnaked, Moveable, Pointer, Resizeable, SlideLeft, SlideUp, Tooltip.Property, Transparency, VTextAlign
Column	AccessibleProperty, Background.Brushmode, Background.Gradient.Property, Background.Transparency, 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, Resizeable, SlideLeft, SlideUp, Tooltip.Property, Transparency, UseEllipsis
Computed field	AccessibleProperty, Background.Brushmode, Background.Gradient.Property, Background.Transparency, Font.Escapement, Font.Width, Height.Autosize, HideSnaked, Moveable, Pointer, Resizeable, SlideLeft, SlideUp, Tooltip.Property, Transparency
Graph	AccessibleProperty, HideSnaked, Moveable, Pointer, Render3D, Resizeable, SlideLeft, SlideUp

Control in DataWindow	Unsupported properties
Group box	AccessibleProperty, Background.Brushmode, Background.Gradient.Property, Background.Transparency, Font.Escapement, Font.Width, Moveable, Pointer, Resizeable, SlideLeft, SlideUp, Tooltip.Property, Transparency
Line (diagonal line is unsupported)	Background.Brushmode, Background.Gradient.Property, Background.Transparency, Moveable, Pen.Style, Pen.Width, Pointer, Resizeable, SlideLeft, SlideUp, Tooltip.Property
Picture	AccessibleProperty, HideSnaked, Invert, Moveable, Pointer, Resizeable, SlideLeft, SlideUp
Rectangle	Background.Brushmode, Background.Gradient.Property, Background.Transparency, Brush.Hatch, Moveable, Pen.Style, Pen.Width, Pointer, Resizeable, SlideLeft, SlideUp, Tooltip.Property
Report	Border, Height.Autosize, Height, HideSnaked, Moveable, NewPage, Pointer, Resizeable, SlideLeft, SlideUp, Trail_Footer
Text	AccessibleProperty, Background.Brushmode, Background.Gradient.Property, Background.Transparency, Font.Escapement, Font.Width, HideSnaked, Moveable, Pointer, Resizeable, SlideLeft, SlideUp, Tooltip.Property, Transparency

Modified Appearance and Behavior of Visual Controls

All PowerBuilder visual controls are supported in .NET Web Forms application, but they may behave or be rendered in a slightly different manner.

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.

See *Web Forms Theme Manager* on page 64.

Visual properties and controls

This table 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 appearance of DataWindow controls in Web Forms applications, see *DataWindow objects and controls* on page 103.

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.
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.

Visual component or control	Behavior in Web Forms applications
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. If a RadControl TreeView (PBWebControlSource=RAD) has many nodes and the PBPostbackType is set to Synchronous, the client Web browser can take a long time to redraw the TreeView. You should use asynchronous postbacks with RadControl controls.
Window	MDI sheet windows display as tab pages instead of cascading sheets.

DataWindow objects and controls

- Freeform DataWindow — in Web Forms applications, DataWindow objects with the Freeform presentation style can show 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 fit in the control, the DataWindow shows one and a half rows. In standard PowerBuilder applications, partial rows are not visible in the control.

- TreeView DataWindow — Web Forms application users cannot use the Tab key to tab between items of a TreeView DataWindow control. The Tab key moves the focus to other controls on the current form.
- **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 appear 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 65.

Unsupported Functions for Controls in Web Forms

Some PowerBuilder control functions cannot be used in applications deployed to ASP.NET.

This table 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:

Function	Controls not supporting function
AcceptText	DataWindow

Function	Controls not supporting function
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 Power-Builder applications)
Clear	Most controls (supported in EditMask controls)
ClearAll	RichTextEdit
Clipboard	Graph
CloseChannel	Window
Copy	All controls
CopyRTF	DataStore, DataWindow, RichTextEdit
Cut	All controls
DataSource	RichTextEdit
DeleteItem	UserObject
DeleteStatePicture	TreeView
DeleteStatePictures	TreeView
DirList	ListBox, DropDownListBox, PictureListBox, DropDownPictureListBox (supported, but can only include files from the virtual file system)
Drag	Most controls (supported in list box controls)
EditLabel	ListView, TreeView
EventParmDouble	UserObject
EventParmString	UserObject
ExecRemote	Window
Find	DataWindow, RichTextEdit
FindNext	DataWindow, RichTextEdit

Function	Controls not supporting function
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
GetDataLabelling	Graph
GetDataTransparency	Graph
GetDisplayRange	MonthCalendar
GetNextSheet	Window (returns sheet instead of frame)
GetItemAtPointer	ListView, TreeView
GetOrigin	ListView
GetParagraphSetting	RichTextEdit
GetRemote	Window
GetRichTextAlign	DataWindow
GetRichTextColor	DataWindow
GetRichTextFaceName	DataWindow
GetRichTextSize	DataWindow
GetRichTextStyle	DataWindow
GetSeriesLabelling	Graph
GetSeriesTransparency	Graph
GetSpacing	RichTextEdit
GetText	DataWindow
GetTextColor	RichTextEdit
GetTextStyle	RichTextEdit

Function	Controls not supporting function
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, RichTextEdit
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
SetCultureFormat	DataWindow
SetDataDDE	Window
SetDataLabelling	Graph
SetDataTransparency	Graph
SetDetailHeight	DataWindow
SetDropHighLight	TreeView
SetFirstVisible	TreeView
SetFocus	DataWindow, Graph
SetLevelPictures	TreeView
SetMicroHelp	Window

Function	Controls not supporting function
SetOverlayPicture	ListView, TreeView
SetParagraphSetting	RichTextEdit
SetPosition	RichTextEdit
SetRedraw	All controls
SetRemote	Window
SetRichTextAlign	DataWindow
SetRichTextColor	DataWindow
SetRichTextFaceName	DataWindow
SetRichTextSize	DataWindow
SetRichTextStyle	DataWindow
SetSeriesLabelling	Graph
SetSeriesTransparency	Graph
SetSpacing	RichTextEdit
SetText	DataWindow
SetTextColor	RichTextEdit
SetTextStyle	RichTextEdit
SetToolbar	Window
SetToolbarPos	Window
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

Some PowerBuilder control events cannot be used in applications deployed to ASP.NET.

This table 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:

Event	Controls for which the event is not supported
BeginDrag	All controls
BeginLabelEdit	ListView, TreeView
BeginRightDrag	All controls
Clicked	DatePicker, MonthCalendar (supported for DataWindow, but not triggered on editable controls that already have focus)
CloseUp	DatePicker
Deactivate	Window
DeleteAllItems	ListView
DoubleClicked	DatePicker, DropDownListBox, DropDownPictureListBox, HProgressBar, ListBox, 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

Event	Controls for which the event is not supported
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
RButtonUp	RichTextEdit
RemoteExec	Window
RemoteHotLinkStart	Window
RemoteHotLinkStop	Window
RemoteRequest	Window
RemoteSend	Window
RichTextCurrentStyle-Changed	DataWindow
RichTextLimitError	DataWindow

Event	Controls for which the event is not supported
RichTextLoseFocus	DataWindow
RightClicked	ListView
RightDoubleClicked	All controls
ScrollHorizontal	DataWindow
ScrollVertical	DataWindow
Selected	Menu
Sort	ListView, TreeView
SystemKey	Window
ToolbarMoved	Window
ValueChanged	DatePicker

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.

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

Some PowerBuilder control properties cannot be used in applications deployed to ASP.NET.

This table 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:

Property	Controls for which the property is unsupported
Accelerator	All controls
AccessibleDescription	Most controls

Property	Controls for which the property is unsupported
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

Property	Controls for which the property is unsupported
DropDownRight	DatePicker, EditMask
EditLabels	TreeView
ExtendedSelect	ListBox, ListView, PictureListBox
FillPattern	StaticHyperLink, StaticText
FixedLocations	ListView
FocusOnButtonDown	Tab
FocusRectangle	Graph, Picture, PictureHyperlink, StaticText, StaticHyperlink
FullRowSelect	TreeView
GridLines	ListView
HasButtons	TreeView (false value not supported in RadControl TreeView, but supported in IE Web Control TreeView)
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
Indent	TreeView (unsupported in RadControl TreeView, supported in IE Web Control TreeView)
InputField properties	RichTextEdit
KeyboardIcon	Window
LabelWrap	ListView
LayoutRTL	ListView, TreeView
LeftMargin	RichTextEdit

Property	Controls for which the property is unsupported
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
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	TreeView
PictureWidth	TreeView
PopMenu	RichTextEdit
Render3D	Graph
Resizable	DataWindow, RichTextEdit, Window
RightMargin	RichTextEdit

Property	Controls for which the property is unsupported
RightToLeft	DataWindow, DatePicker, DropDownListBox, DropDownPicture-ListBox, ListBox, ListView, MonthCalendar, 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	TreeView
SliderSize	HTrackBar, VTrackBar
SmoothScroll	HProgressBar, VProgressBar (smooth scrolling is supported, but not step increments)
Spin	EditMask
StatePictureHeight	TreeView
StatePictureWidth	TreeView
StatusBar	RichTextEdit
Style	UserObject
TabBackColor	UserObject
TabBar	RichTextEdit
TabStop	EditMask, ListBox, MultiLineEdit, PictureListBox
TickFrequency	HTrackBar, VTrackBar
TickMarks	HTrackBar, VTrackBar
Title	DataWindow
TitleBackColor	Menu
TitleBar	DataWindow, Window
TitleGradient	Menu

Property	Controls for which the property is unsupported
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
Transparency	DataWindow object, Window
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.

PowerBuilder Windows Forms Applications

PowerBuilder applications with a rich user interface that rely on resources of 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 Windows Forms applications.

For a comparison of design considerations between Web Forms and Windows Forms applications, see *Choosing a .NET Application Target* on page 1.

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 *Best Practices for .NET Projects* on page 201.

For tables of unsupported and partially supported objects, controls, functions, events, and properties, see *Unsupported Features in Windows Forms Projects* on page 140.

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. See *Properties for a .NET Windows Forms Project* on page 123.

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. See *Intelligent Deployment and Update* on page 129.

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. See *Deployment of a Windows Forms Application* on page 128.

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 and ignored by the PowerScript compiler:

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

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

For more information about using preprocessor symbols, see *Conditional Compilation* on page 173.

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.

1. Install .NET Framework 2.0, 3.0, or 3.5 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 *Application Techniques > Deploying Applications and Components*.
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.

For information about requirements for deployed applications, see *Checklist for Deployment* on page 14.

System Requirements for .NET Windows Forms Targets

You must install version 2.0, 3.0, or 3.5 of the Microsoft .NET Framework on the same computer as PowerBuilder. For intelligent update applications, you must also install the .NET Framework 2.0, 3.0, or 3.5 SDK (x86).

Make sure that the system PATH environment variable includes:

- The location of the .NET Framework. The location of the 2.0 version is typically `C:\Windows\Microsoft.NET\Framework\v2.0.50727`. The location of the 3.5 version is typically `C:\Windows\Microsoft.NET\Framework\v3.5`.
- For intelligent update applications, the location of the .NET Framework SDK `Bin` directory. For .NET Framework 2.0, this is typically `C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin` or `C:\Program Files\Microsoft.NET\SDK\v2.0\Bin`. For version 3.5, this is typically `C:\Program Files\Microsoft Visual Studio 9\SDK\v3.5\Bin` or `C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin`.

The SDK for .NET Framework 2.0 is available from the *Microsoft .NET Framework Developer Center*. The Windows SDK for Windows Server 2008 and .NET Framework 3.5 is available on the *Microsoft .NET Framework Developer Center*.

If you installed the 1.x version of the .NET Framework or SDK, you must make sure the PATH variable lists a supported version of the .NET Framework or SDK 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 8.

Adding .NET assemblies

If you want to call methods in .NET assemblies in your Windows Forms application, you can import the assemblies into the target. For more information, see *Adding .NET Assemblies to the Target* on page 178.

.NET Windows Forms Target Wizard

Use the .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 *Intelligent Deployment and Update* on page 129.

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.

Building a Windows Forms Application and Target from Scratch

Use the .NET Windows Forms Application wizard to create 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* on page 123.

Building a Windows Forms Application from an Existing Application and Library

Use the .NET Windows Forms Application wizard to create a .NET Windows Forms application and target 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* on page 123.

Building a Windows Forms Application from an Existing Target

Use the .NET Windows Forms Application wizard to create a .NET Windows Forms application and target 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* on page 123.

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.

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 **Add** 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 shows additional pages on which you set publish and update options. See *Intelligent Deployment and Update* on page 129.

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 shows all the values you entered in the wizard and allows you to modify them. It also shows additional properties that you can set only in the painter.

Table 1. Properties in the Project painter

Tab page	Properties
General	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. 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. The rebuild scope determines whether the project build is incremental (default) or full. See *Rebuild Scope* on page 214. 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 preprocessor statements, see *Conditional Compilation* on page 173.
Resource Files	PowerBuilder .NET Windows Forms do not support PBR files, and they are unable to locate images embedded in PBD files. You can, however, search a PBR file for images required by the application. 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. 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. For smart client applications, 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. See *Resource Files and Publish Type* on page 135.

Tab page	Properties
Library Files	Use the Library Files tab page to make sure all the PowerBuilder library files (PBLs or PBDs) that contain DataWindow, Query, and Pipeline objects used by the application are deployed with the application. If you select the check box next to the name of a PBL that contains these types of objects, PowerBuilder compiles the selected PBL into a PBD file before deploying it. **Note:** You can reference only DataWindow, Query, or Pipeline objects in a PBD file. The PBD files that are generated when you compile a Windows Forms project do not contain other PowerBuilder objects, such as functions or user objects. If you include a PBD file in your target that contains these other types of objects, you cannot reference them from the Windows Forms application. They can be referenced only from a target PBL that is converted to a .NET assembly. 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 `pbshr125.dll` and `pbdwe125.dll` (if the project uses DataWindows), on this page, or you can add them on the Prerequisites page.
Version	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. See *Publication Process and Results* on page 132.
Post-build	Use the Post-build tab 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.)
Security	Use the Security tab page to generate a manifest file (either external or embedded) and to set the execution level of the application.To meet the certification requirements of the Windows Logo program the application executable must have an embedded manifest that defines the execution level and specifies whether access to the user interface of another window is required. You can also use the Security tab to configure CAS security zones for your applications, minimizing the amount of trust required before application code is run by an end user. For information about manifest file requirements, see *Security Requirements* on page 127. For information about customized permission settings, see *Security Settings* on page 3 and *Custom Permission Settings* on page 231.
Run	Use the Run tab 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.

Tab page	Properties
Sign	The Assembly group box on the Sign tab page allows you to attach strong name key files to the assemblies that your project generates. You must also use the Sign tab page to attach digital certificates to manifest files that you publish for smart client applications. See *Strong-Named Assemblies* on page 6 and *Digital Certificates* on page 131.

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. See *Intelligent Deployment and Update* on page 129.

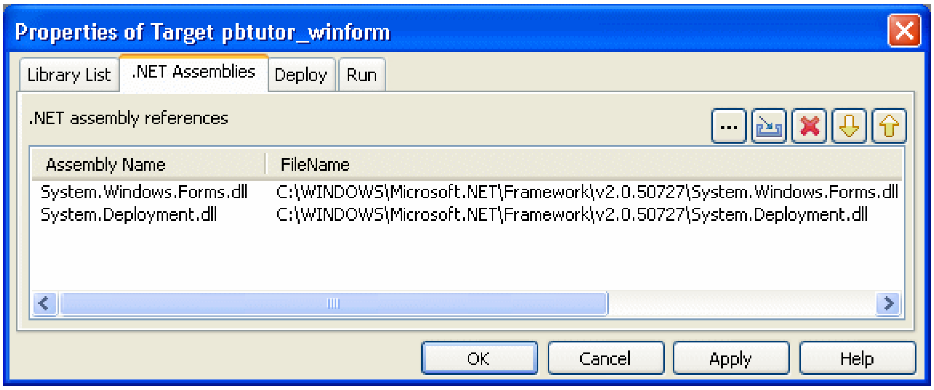
Resources and Other Required Files

All resource files must be relative to the path of the .NET Windows Forms target.

Click **Add Files** on the Resource Files page of the project painter to select image files that your application requires.

PowerBuilder .NET Windows Forms applications do not support PBR files, and they are unable to locate images embedded in PBD files. 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.

If your application uses .NET assemblies, specify them on the .NET Assemblies tab page in the target's Properties dialog box. Before you deploy a PowerBuilder .NET smart client application that uses data files, make sure the `System.Windows.Forms.dll` and `System.Deployment.dll` assemblies are listed on this page.



Assembly Name	FileName
System.Windows.Forms.dll	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Windows.Forms.dll
System.Deployment.dll	C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Deployment.dll

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.

Security Requirements

Use the Security tab page of the project painter to specify whether an application has a manifest file to set its requested execution level, and whether the manifest file is external or embedded in the application.

This manifest file is not the same as the manifest files generated when you publish a Windows Forms application as a smart client (ClickOnce) application. The concept of execution level is part of the User Account Control (UAC) protocol.

If you want to deploy an application that meets the certification requirements of the Windows Logo program, you must follow UAC guidelines. The executable file must have an embedded manifest that defines the execution level and specifies whether access to the user interface of another window is required. The Application Information Service (AIS) checks the manifest file to determine the privileges with which to launch the process.

Generate options

Select Embedded manifest if your application needs to be certified for Vista or later. A manifest file with the execution level you select is embedded in the application's executable file.

You can also select External manifest to generate a standalone manifest file in XML format that you ship with your application's executable file, or No manifest if you do not need to distribute a manifest file.

Note: If you select Embedded manifest for a Windows Forms target, you must have a supported version of the .NET Framework SDK installed on your system, because the process that embeds the manifest in the executable file uses the **mt.exe** tool that is distributed with the SDK.

Execution level

Select As Invoker if the application does not need elevated or administrative privileges. Selecting a different execution level will probably require that you modify your application to isolate administrative features in a separate process to receive Vista or later certification.

Select Require Administrator if the application process must be created by a member of the Administrators group. If the application user does not start the process as an administrator, a message box displays so that the user can enter the appropriate credentials.

Select Highest Available to have the AIS retrieve the highest available access privileges for the user who starts the process.

UI access

If the application needs to drive input to higher privilege windows on the desktop, such as an on-screen keyboard, select the "Allow access to protected system UI" check box. For most applications you should not select this check box. Microsoft provides this setting for user interface Assistive Technology (Section 508) applications.

Note: If you check the Allow access to protected system UI check box, the application must be Authenticode signed and must reside in a protected location, such as `Program Files` or `Windows\system32`.

Deployment of a Windows Forms Application

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 context menu on the Deploy tab page in the target's Properties dialog box.

The Output window displays the progress of the deployment. PowerBuilder compiles PBLs into PBD files when they contain DataWindow, Query, or Pipeline objects that are referenced in the application. 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 *Connecting to Your Database > Using the ADO.NET Interface*. The sample shows how to configure tracing in the `appname.exe.config` file, as shown in *Runtime Errors* on page 228.

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 *Unsupported Features in Windows Forms Projects* on page 140.

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 *Best Practices for .NET Projects* on page 201.

Project Execution

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 context 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 context 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.

When you debug or run the project from PowerBuilder, a system option setting can cause a message box to appear if the application has been modified since it was last deployed. The message box prompts you to redeploy the application, although you can select No to debug or run the older application, and you can set the system option to prevent the message box from appearing.

For information about the message box, see *Triggering Build and Deploy Operations* on page 215. For information about the system option, see *System Option* on page 215.

For information on debugging .NET Windows Forms targets, see *Debugging a .NET Application* on page 217.

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 Windows Forms applications can use this "intelligent update" feature.

As the developer of a 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 a user’s computer.

When the user clicks on a link, typically on a Web page or in an e-mail, 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.

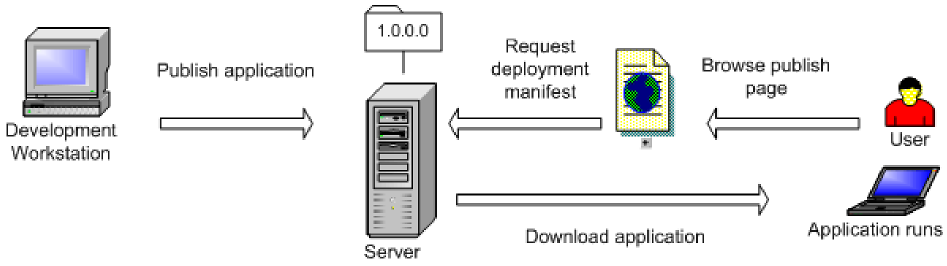
Publishing an application for the first time

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.

You need to:

- *Create a project and set publishing properties* on page 129
- *Publish the application* on page 132

Figure 1: Deploying an intelligent update application



Set Publishing Properties

If you did not create a .NET Windows Forms project when creating an application that you want to publish with intelligent update capabilities, you can use a wizard or icon on the Project page of the New dialog box to create the project.

1. On the Project page of the New dialog box, select the .NET Windows Forms Application wizard or project icon.
2. On the Specify Support for Smart Client page in the wizard, select the check box to specify that the application uses intelligent update.
   Selecting this check box enables additional pages in the wizard.
3. On the Specify Application Running Mode page, specify whether the application can be used both online and offline (default), or online only.
4. 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.
5. On the Specify Application Update Mode page, specify whether the application checks for updates before starting, after starting, or neither. See *Publication of Application Updates* on page 135.

   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.

Digital Certificates

A digital certificate is a file that contains a cryptographic public/private key pair, along with metadata describing the publisher to whom the certificate was issued and the agency that issued the certificate.

Digital certificates are a core component of the Microsoft Authenticode authentication and security system. Authenticode is a standard part of the Windows operating system. To be compatible with the .NET Framework security model, all PowerBuilder .NET applications must be signed with a digital certificate, regardless of whether they participate in Trusted Application Deployment. For more information about Trusted Application Deployment, see the *Microsoft Web site*.

Signing manifests with digital certificates

You can select a digital certificate from a certificate store or from a file browser. to sign your smart client application manifests. You make the selection on the Sign page of the Project painter by selecting the Sign the manifests check box in the Certificate group box.

This table describes the fields in the Intelligent Updater group box on the Sign page of the Windows Forms Project painter. These fields are grayed out when the Publish as Smart Client Application check box on the General tab of the Project painter has not been selected.

Intelligent Updater field	Description
Sign the manifests	Select this check box to enable the Select from Store and Select from File buttons. Use the buttons to select a certificate from a certificate store or from your file system. If you select a valid certificate, its details display in the multiline edit box under the check box. If you do not specify a certificate, PowerBuilder attaches a test certificate automatically. Use test certificates for development only.
Select from Store	Click this button to view the certificates available in the local certificate store. Select a certificate from the Select a Certificate dialog box, then click View Certificate if you want to view its details, and click OK to select it.

Intelligent Updater field	Description
Select from File	Click this button to view the certificates available in the local file system. Select a certificate with the `.snk` extension from the Select File dialog box and click Open.

Use the Select from Store or Select from File buttons to select a certificate from a certificate store or from your file system. If the certificate requires a password, a dialog box displays so that you can enter it. When you select a valid certificate, detailed information displays in the Project painter.

If you do not specify a certificate, PowerBuilder signs the published manifest file with the default test certificate, `mycert.fx`. This test certificate is installed by the PowerBuilder setup program in the PowerBuilder `DotNet\pbiu\commands` directory. However, when you are ready to publish a production application, you should not sign it with the test certificate.

For information about application manifests required on the Vista and later operating systems, see *Security Requirements* on page 127.

Setting Full Trust Permissions

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.

1. From the Windows Control Panel, select **Administrative Tools > Microsoft .NET Framework 2.0 Configuration**.
2. In the .NET Framework Configuration tool, expand My Computer and select **Runtime Security Policy > Machine > Code Groups > All_Code > LocalIntranet_Zone**.
3. From the context menu, select Properties.
4. In the Permission set drop-down list on the Permission Set tab page, select FullTrust.

Publication Process and Results

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 wizard defaults, the application is deployed to a subdirectory of the IIS root directory on your local computer, usually `C:\Inetpub\wwwroot`.

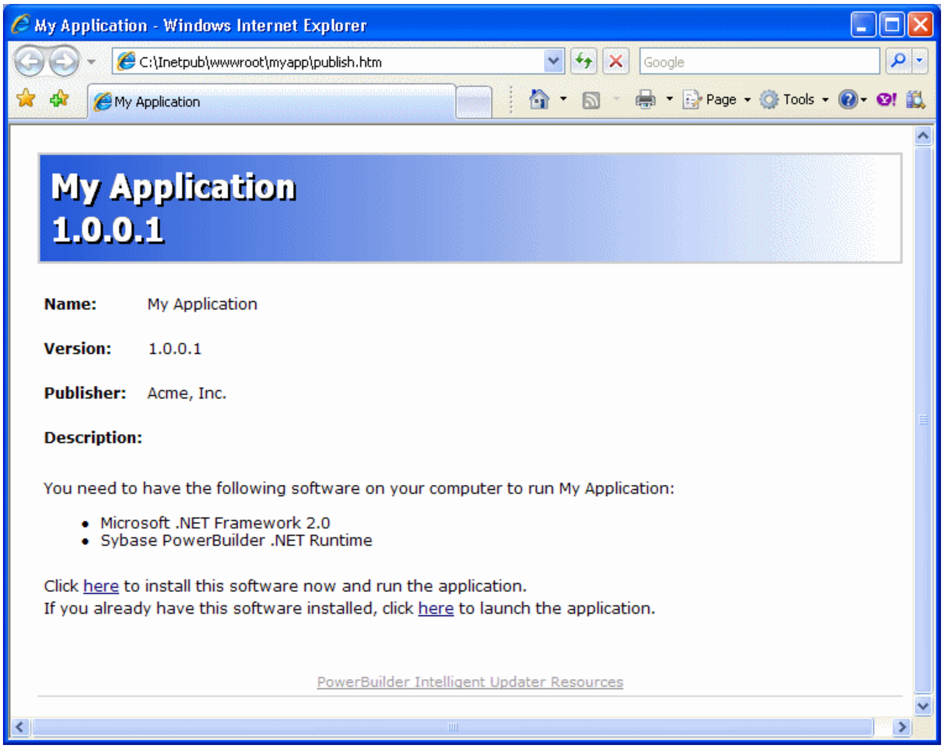
If you encounter problems when publishing the application, see *Troubleshooting Tips for Windows Forms Applications* on page 228.

These 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 *Application Bootstrapping* on page 138.
- 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 Version tab page in the Project painter. If you supply a Publish description on the Publish tab page in the Project painter, it displays on the `publish.htm` page.

Figure 2: Publish page with prerequisites


My Application - Windows Internet Explorer
C:\Inetpub\wwwroot\myapp\publish.htm
Google
My Application
Page
Tools

My Application
1.0.0.1

Name: My Application

Version: 1.0.0.1

Publisher: Acme, Inc.

Description:

You need to have the following software on your computer to run My Application:

- Microsoft .NET Framework 2.0
- Sybase PowerBuilder .NET Runtime

Click here to install this software now and run the application.
If you already have this software installed, click here to launch the application.

PowerBuilder Intelligent Updater Resources

Application Installation 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. See *Application Bootstrapping* on page 138.

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 (Programs and Features page on Vista and later) 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.

Resource Files and Publish Type

In a smart client application, image files that you add on the Resource Files page in the project painter 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!)
```

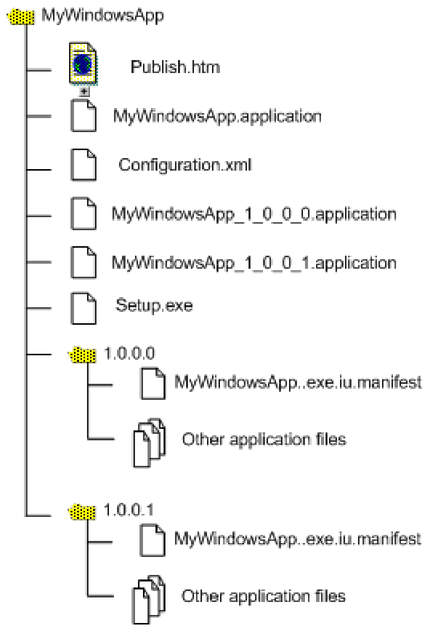
For information about using preprocessor symbols such as PBWINFORM, see *Conditional Compilation* on page 173.

Publication of Application Updates

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.

This figure 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. See *Rolling Back* on page 140.

Online-only applications

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

Online and offline applications

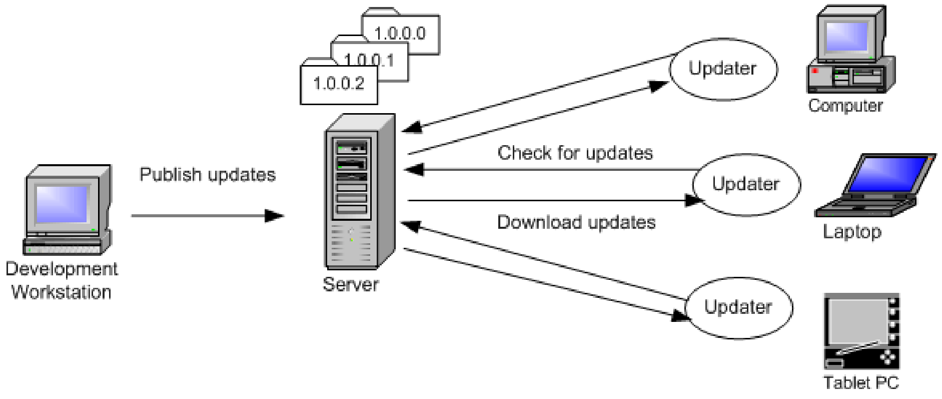
If the application is available offline as well as online, the user is notified of new updates according to the update strategy you specified in the wizard or Project painter. Whether the application was originally installed from the Web, a file server, or a CD or DVD, the intelligent updater component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

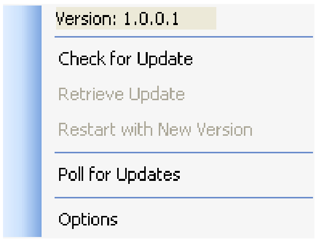
If an update is available, a dialog box displays to inform the user, who can choose to download the update immediately or skip the current update and check again later. The user cannot skip the update if you have specified that it is mandatory. You set all these properties on the Install/ Update page.

Figure 3: 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 context menu that appears when a user right-clicks the notifier icon shows 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.

Application Bootstrapping

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 the figure in *Publication Process and Results* on page 132.

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 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.

Prerequisites Page Customizations

The selections available on the Prerequisites page can be customized by adding a new subdirectory to the `PowerBuilder` *`version`*`\DotNET\pbiu\BootStrapper\Packages` directory. To this subdirectory, 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 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 dotNetInstaller open source tool to set up your own customizations. It can be downloaded from the *CodePlex Web site*.

A comparison of Windows Installer tools is available on the *InstallSite organization's Web site*.

Packages on the Prerequisites page

There are two packages available on the Prerequisites page: the .NET Framework runtime files and the Sybase PowerBuilder .NET 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 package, you need to copy the .NET Framework 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 Sybase PowerBuilder .NET Runtime 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. See *Installing assemblies in the global assembly cache* on page 18.

If you do not require all the files included in the package, you can create your own package. See *Prerequisites Page Customizations* on page 139.

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

For information about editing `configuration.xml` files, see the tutorial for the dotNetInstaller available on the *Code Project Web site*.

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 the figure in *Publication of Application Updates* on page 135, 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.

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 *Users Guide > Using the ASA MobiLink synchronization wizard* and *Application Techniques > Using MobiLink Synchronization*.

Unsupported Features in Windows Forms Projects

PowerBuilder Windows Forms applications do not currently support some standard PowerBuilder features. Some of these are not implemented in the current release of PowerBuilder, and others 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 the list of datatype mappings from .NET to PowerBuilder in the *Datatype Mappings* on page 179 topic.
- You can use the built-in Web services client extension (`pbwsclient125.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 SDK, which can be freely downloaded from the Microsoft Web site. See *System Requirements for .NET Windows Forms Targets* on page 121.
- These features are not currently supported in .NET targets: tracing and profiling, DDE functions, and SSLCallback.
- The .NET Framework replaces fonts that are not TrueType or OpenType fonts, such as Courier or MS Sans Serif. To avoid issues with replacement fonts, always use a TrueType or OpenType font.

Unsupported Nonvisual Objects and Structures in Windows Forms

Windows Forms applications support most PowerBuilder objects, controls, functions, events, and properties.

This table lists all PowerBuilder nonvisual objects and structures and indicates whether they are currently supported in Windows Forms applications. When there is an X in the Partially Supported column of this table, see the second table for detailed information about what is not supported. The XX symbol in the Unsupported column of the first table indicates that there are no current plans to support the corresponding object in future versions of PowerBuilder:

Table 2. 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		
EnumerationDefinition		X	
EnumerationItemDefinition	X		
Environment	X		
ErrorLogging	X		

Class name	Supported	Partially supported	Unsupported
Exception	X		
Graxis	X		
GrDispAttr	X		
Inet	X		
InternetResult	X		
JaguarOrb	X		
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
PBDOM			XX
PbxRuntimeError		X	
Pipeline	X		
ProfileCall			XX
ProfileClass			XX
ProfileLine			XX

Class name	Supported	Partially supported	Unsupported
ProfileRoutine			XX
Profiling			XX
RemoteObject			XX
ResultSet	X		
ResultSets	X		
RuntimeError		X	
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	

Class name	Supported	Partially supported	Unsupported
VariableCardinalityDefinition	X		
VariableDefinition		X	
WSConnection	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.

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

Table 3. 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
DataStore	• CopyRTF • GenerateHTMLForm • GenerateResultSet • GetStateStatus • InsertDocument • PasteRTF • Print (supported but not for data with rich text formatting)	• Destructor	• None
DataWindowChild	• DBErrorCode • DBErrorMessage • SetRedraw • SetRowFocusIndicator	• None	• None

Class name	Unsupported functions	Unsuppor-ted events	Unsupported properties
OmObject	• GetAutomationNative-Pointer • SetAutomationLocale • SetAutomationTimeOut	• None	• None
RuntimeError (and its descendants)			• Class • Line • Number
ScriptDefinition			• AliasName • ExternalUserFunc-tion (supported for external functions only) • LocalVariableList • Source • SystemFunction
SimpleTypeDefini-tion			• LibraryName
TypeDefinition			• LibraryName
VariableDefinition			• InitialValue (sup-ported for instance variables and prim-itive types) • IsConstant (suppor-ted for instance var-iables) • OverridesAncestor-Value • ReadAccess (sup-ported for instance variables) • WriteAccess (sup-ported for instance variables)

Unsupported System Functions in Windows Forms

Most PowerBuilder system functions are supported in Windows Forms applications.

This table lists categories of system functions that are not supported.

Table 4. 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**
Miscellaneous functions	**PBGetMenuString**
Input method functions	**IMEGetCompositionText**, **IMEGetMode**, **IMESetMode**
Profiling and tracing functions	**TraceBegin**, **TraceClose**, **TraceDisableActivity**, **TraceDump**, **TraceEnableActivity**, **TraceEnd**, **TraceError**, **TraceOpen**, **TraceUser**

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 when 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.

PowerBuilder Visual Controls in Windows Forms Applications

For most PowerBuilder visual controls, the only unsupported event in Windows Forms applications is the Other event, and the only unsupported property is IMEMode.

This table lists PowerBuilder visual controls and indicates whether they are fully or partially supported in Windows Forms applications. If a control has no unsupported events, properties, or functions besides the Other event and the IMEMode property, it is listed in the Supported column. When there is an X in the Partially Supported column, see the second table for detailed information about which functions, events, and properties are not supported:

Table 5. 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
OmCustomControl		X
OmEmbeddedControl		X
Oval	X	

Class name	Supported	Partially supported
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 6. Unsupported functions, events, and properties by control

Supported control	Unsupported functions	Unsupported events	Unsupported properties
DataWindow	• DBErrorCode • DBErrorMessage • GenerateHTML-Form • GetStateStatus	• Other	• RightToLeft

Supported control	Unsupported functions	Unsupported events	Unsupported properties
DatePicker	• GetCalendar • Resize (does not support changing height, otherwise supported)	• DoubleClicked • Other UserString	• AllowEdit • Border • BorderStyle
DropDownListBox	• None	• Other	• HScrollBar • IMEMode
Graph	• None • AddData, GetDataValue, InsertData, ModifyData do not support string values	• Other	• BorderStyle
ListBox	• None	• Other	• TabStop
ListView	• AddColumn, InsertColumn, SetColumn limitation: the alignment of the first column cannot be set to center or right	• Other	• IMEMode
Menu	• None	• Help	• MenuItemType • MergeOption • ToolbarAnimation • ToolbarHighlightColor • ToolbarItemSpace
MenuCascade	• None	• Help	• Columns • CurrentItem • DropDown • MenuItemType • MergeOption • ToolbarAnimation • ToolbarHighlightColor • ToolbarItemSpace

Supported control	Unsupported functions	Unsupported events	Unsupported properties
MonthCalendar	• None	• DoubleClicked • Other	
MultiLineEdit	• None	• Other	• IMEMode • TabStop
OmCustomControl	• None	• None	• Alignment • Cancel • Default
OmEmbedded-Control	• _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 156)

Supported control	Unsupported functions	Unsupported events	Unsupported properties
TreeView	• None	• Other	• IMEMode • StatePictureHeight • StatePictureWidth
UserObject	• AddItem • DeleteItem • EventParmDouble • EventParmString • InsertItem	• Other	• BackColor, TabTextColor (for tab pages —see *Tab properties* on page 156) • Style
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

If your application uses unsupported functions for Windows Forms targets, you must rework the application before you deploy it.

This table is an alphabetical listing of unsupported functions. It also lists the controls on which they are not supported, and any notes that apply to specific controls:

Table 7. Unsupported functions for Windows Forms deployment

Function	Controls
AddItem	UserObject
CloseChannel	Window

Function	Controls
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
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

Function	Controls
SetDataDDE	Window
_Set_ObjectData	OmEmbeddedControl
SetCultureFormat	DataWindow
SetRemote	Window
SetWSObject	DataWindow
StartHotLink	Window
StartServerDDE	Window
StopHotLink	Window
StopServerDDE	Window
UpdateLinksDialog	OmEmbeddedControl

Unsupported Events for Controls in Windows Forms

If your application uses unsupported events for Windows Forms targets, you must rework the application before you deploy it.

This table is an alphabetical listing of unsupported events, and indicates the controls on which they are not supported:

Table 8. 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

If your application uses unsupported properties for Windows Forms targets, you must rework the application before you deploy it.

This table is an alphabetical listing of unsupported properties. It also indicates the controls on which they are not supported, and any notes that apply to specific controls.

Table 9. Unsupported properties for Windows Forms deployment

Property	Controls
Alignment	OmCustomControl
AllowEdit	DatePicker
Activation	OmEmbeddedControl
BackColor	Tab, UserObject (see *Tab properties* on page 156)
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
Help	Menu, MenuCascade
HScrollbar	DropDownListBox
IMEMode	All controls
LinkUpdateOptions	OmEmbeddedControl
Map3DColors	PictureButton
MenuItemType	Menu
MergeOption	Menu
ObjectData	OmEmbeddedControl
ParentStorage	OmEmbeddedControl
RaggedRight	Tab (see *Tab properties* on page 156)

Property	Controls
RightToLeft	DataWindow, ListBox, ListView, TreeView
SizeMode	OmEmbeddedControl
StatePictureHeight	TreeView
StatePictureWidth	TreeView
Style	UserObject
TabStop	ListBox, MultiLineEdit
TabTextColor	UserObject (see *Tab properties* on page 156)
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 **GdipCreateFont**) 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 that the width of the Tab control, and the MultiLine property is set to **true**. However, when the PerpendicularText property is true, RaggedRight is not supported.

While the TabPosition property value is tabsonleft! or tabsonright!, and there is not enough room for all the tabs in a single row, the tabs appear in more than one row, regardless of the Multiline property setting. If you then dynamically set Multiline to true, the tabs display on top of the Tab control, regardless of the TabPosition setting.

Dual position display is not supported by the .NET Tab control (System.Windows.Forms.TabControl), so the TabPosition value tabsontopandbottom! displays tabs on top only. The tabsonrightandleft! value displays tabs only on the right, and the tabsonleftandright! value displays tabs only on the left.

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

.NET Component Targets

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

.NET Assembly Targets

PowerBuilder includes a target type for creating .NET assemblies from nonvisual custom class objects.

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.

Note: The .NET Assembly target type is available in both PowerBuilder Classic and PowerBuilder .NET. To take advantage of Common Language Specification (CLS) compliant features, use the .NET Assembly target in PowerBuilder .NET.

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 (NVO). 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.

This table describes the information you must provide for .NET Assembly targets that you create 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.

Wizard field	Description
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	List of resource files, or directories containing resource files, that you want to deploy with the project. 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. 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 161.
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.

Note: All objects from an existing target are visible 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.

This table describes these additional fields:

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.
Only include functions with supported datatypes	Select this check box if you do not want to list functions that are not supported in the .NET environment. The functions will be listed in the Select Objects dialog box that you can open for the project from the Project painter.

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 the table for creating a project from scratch, except for the “PowerBuilder object name” and “Description” fields. They also include all fields in the table for creating a project from an existing target, except for 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 these tables are available for modification in the Project painter.

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, and select and rename functions of the nonvisual objects that 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 *.NET Assembly Targets* on page 157.

The General tab also has fields that are not available in the target or project wizards. This table describes the additional fields:

Project painter field	Description
Debug or Release	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	Option 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 option 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.

This table describes the fields available on the Objects tab:

Project painter 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 Name-space	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.

Project painter field	Description
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, but you cannot use dashes ("-") in the modified names. 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 Unse-lect 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 the first table in *.NET Assembly Targets* on page 157.

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. This table describes these fields:

Project painter field	Description
Version tab: Product name, Compa-ny, Description, and Copyright	Use these fields to specify identification, description, and copyright information that you want to associate with the as-sembly you generate for the project.
Version tab: Product version, File version, and Assembly	Enter major, minor, build, and revision version numbers for the product, file, and assembly.

Project painter field	Description
Post-build tab: Post-build command line list for build type	Select the build type (Debug or Release) and click **Add** 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.
Run tab: Application	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.
Run tab: Argument	You use this text box to enter any parameters for an application that invokes the classes and methods of the deployed project.
Run tab: Start In	You use this text box to enter the starting directory for an application that invokes the classes and methods of the deployed project.

Sign tab

The fields that you can edit on the Sign tab of the Project painter are the same as the fields available for other .NET projects, although one of the fields that permits calls to strong-named assemblies from partially trusted code is available only for .NET Assembly and .NET Web Service projects. For descriptions of the fields on the Sign tab, see *Strong-Named Assemblies* on page 6.

Supported Datatypes

The PowerBuilder to .NET compiler converts PowerScript datatypes to .NET datatypes.

This table shows the datatype mapping between PowerScript and C#:

PowerScript datatype	C# datatype
boolean	bool
blob	byte [ ]
byte	byte
int, uint	short, ushort

PowerScript datatype	C# datatype
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

Note: Arrays are also supported for all standard datatypes.

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 context 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 and any resource files you listed in the wizard or Project painter.

Note: You can use the Runtime Packager to copy required PowerBuilder runtime files to deployment computers.

For information on required runtime files, see *Checklist for Deployment* on page 14. For information about the Runtime Packager, see *Application Techniques > Deploying Applications and Components*.

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.

.NET Web Service Targets

PowerBuilder includes a target type for creating .NET Web services from nonvisual custom class objects.

The .NET Web Service target wizard gives you the option of creating a target from scratch or from an existing PowerBuilder target.

Creating a target from scratch

The .NET Web Service target wizard shares the following fields in common with the .NET Assembly target: Project Name, Target, Library, Library Search Path, PowerBuilder Object Name, Description, Resource Files, and Win32 Dynamic DLLs. However, it has four additional fields (Web service virtual directory name, Web service URL preview, Generate setup file, and Directly deploy to IIS), and the Namespace and Assembly File Name fields are specific to the .NET Assembly wizard.

This table describes the fields in the .NET Web Service wizard when you create a target 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.
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.

Wizard field	Description
Resource file and directory list	List of resource files, or directories containing resource files, that you want to deploy with the project. 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. 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 195.
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.
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.
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 Web service on those computers.
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 for a target you are creating 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, you can use the .NET Web Service target wizard to create 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 wizard for a target you are creating from an existing target, 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. This table describes these additional fields:

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.
Only include functions with supported datatypes	Select this check box if you do not want to list functions that are not supported in the .NET environment. After you create the .NET project, the functions are listed on the Objects tab for the project when you open it in the Project painter.
Only include functions with supported datatypes	Select this check box if you do not want to list functions that are not supported in the .NET environment. The functions will be listed in the Select Objects dialog box that you can open for the project from the Project painter.

Modifying a .NET Web Service Project

You can modify a .NET Web Service project from the Project painter.

The Project painter shows 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, and 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 these tab pages:

- General tab — includes debug fields that are not available in the target or project wizards.

Project painter field	Description
Debug or Release	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	Option 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 option is selected by default.

- Deploy tab — 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 the first table in *.NET Assembly Targets* on page 157.
- Objects tab — allows you to select the methods to make available for each nonvisual object you deploy as a Web service. You can rename the methods as Web service messages. This table describes the Objects tab fields for a .NET Web Service project:

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: http://tempurl.org. 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.
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.

Objects tab field	Description
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.

- Resource Files tab — the fields on this tab 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 — includes fields for the Win 32 dynamic libraries you want to deploy with your project. These fields are described in *.NET Web Service Targets* on page 164. The Library Files tab also includes a list of PBL files for the target. You can select a check box next to each PBL files containing DataWindow or Query objects to make sure they are compiled and deployed as PBD files.
- Version tab — the fields on this tab cannot be set in the target or project wizards:

Version tab field	Description
Product name, Company, Description, and Copyright	Use these fields to 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	Enter major, minor, build, and revision version numbers for the product, file, and assembly.

- Post-build tab — the items on this tab cannot be set in the target or project wizards. Select a build type and click **Add** 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.
- Security tab — on this tab, configure CAS security zones for Web Service components, minimizing the amount of trust required before component code is run from a user application. A radio button group field on the Security tab allows you to select full trust (default) or a customized trust option. The list box below the radio button group is disabled when full trust is selected, but it allows you to select or display the permissions you want to include or exclude when the custom option is selected.

  For information on custom permission requirements, see *Security Settings* on page 3 and *Custom Permission Settings* on page 231.
- Run tab — the fields on this tab cannot be set in the target or project wizards:

Run tab field	Description
Application	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	Use this text box to enter any parameters for an application that invokes the classes and methods of the deployed project.
Start In	Use this text box to enter the starting directory for an application that invokes the classes and methods of the deployed project.

- Sign tab — the settings on this tab are the same as those available for other .NET projects, although the field that permits calls to strong-named assemblies from partially trusted code is available only for .NET Assembly and .NET Web Service projects. For descriptions of the fields on the Sign tab, see *Strong-Named Assemblies* on page 6.

Configuring ASP.NET for a .NET Web Service Project

Configure .NET Web Service projects in the same way you configure .NET Web Forms projects.

IIS and ASP.NET

ASP.NET configuration includes making sure the Web server has a compatible version of IIS and that 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 *ASP.NET Configuration for a .NET Project* on page 7. For information on the directory structure for deployed projects, see *Directory Structure on the Server* on page 9.

SQL Anywhere database connections

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. 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
PBDeleteTempFileInterval

See *Viewing and Modifying Global Properties in the IIS Manager* on page 9 and *Global Web Configuration Properties* on page 65.

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 context 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. In fact, PowerBuilder deploys a Web Service target as a Web Forms target, but it creates an additional assembly containing the Web service wrapper class. The file name for this assembly is generated by appending the characters "_ws" to the file name of the main application assembly. It is generated with the main assembly in the application's bin directory.

For more information, see *Directory Structure on the Server* on page 9.

Note: In some versions of IIS for the Windows XP platform, ASPNET Web services use the Temp system directory during method processing. If the ASPNET user (IIS 5), the IIS_WPG user group (IIS 6), or the IIS_IUSRS user group (IIS 7 and 7.5) 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. 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.

Note: 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, see *Checklist for Deployment* on page 14. For information about the Runtime Packager, see *Application Techniques > Deploying Applications and Components*.

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 computer.

.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 describes how to connect to an EAServer component from a .NET client. A chapter on best practices provides suggestions for enhancing the .NET applications and components you build in PowerBuilder.

Conditional Compilation

Use the number sign (#) at the start of a line or block of code in PowerBuilder 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 standard PowerBuilder 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 or passed to the compiler depending on which preprocessor symbol is used.

This table shows 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 10. 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.
PBWINFORM	.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.
DEBUG	Standard PowerBuilder 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.

Note: The PBWPF preprocesser can be used for WPF Window Application targets in the PowerBuilder .NET IDE. PowerBuilder Classic ignores the scripts inside these code blocks, except when the NOT operator is used with this preprocesser. The PBDOTNET and DEBUG code blocks are valid for both PowerBuilder Classic and PowerBuilder .NET.

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 structures, or in menu objects. 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.

This table shows the types of error messages displayed for incorrect conditional compilation code:

Table 11. 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 (`//`)

Surrounding Code in 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.")

#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.

You can paste preprocessor statements into the Script view. Select **Edit** > **Paste Special** > **Preprocessor** and select the statement you need.

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
   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 include the @ identifier when you reference the .NET *Type* class in PowerScript code (`@System.@Type` or `System.@Type`). Also, if you use a PowerBuilder keyword for a .NET namespace name other than System, you must prefix the namespace name with the @ identifier.

Although PowerBuilder can support .NET Framework classes and namespaces, it does not support .NET keywords. For example, you cannot use the .NET keyword *typeof*, even if you prepend it with the @ identifier.

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 PowerScript. For information about using enumeration constants in the .NET environment, see *User-Defined Enumerations* on page 183.

Adding .NET Assemblies to the Target

To call methods in .NET assemblies in your .NET application, you need to import the assemblies into the target.

1. Right-click the target in the System Tree and select .NET Assemblies.
2. To import a private .NET Assembly:
   a) Click **Browse**
   b) Browse to select a private assembly with the `.dll`, `.tlb`, `.olb`, `.ocx`, or `.exe` extension and click Open.

   To import multiple assemblies, you must select and import them one at a time.
3. To import a shared .NET Assembly:
   a) Click **Add** to open the Import .NET Assembly dialog box.
   b) Select a shared assembly from the list and click **OK**.

To import multiple assemblies, you must select and import them one at a time. You can use the Import .NET Assembly dialog box to import recently used assemblies.

For more information about shared and private assemblies, see *Installing assemblies in the global assembly cache* on page 18.

Datatype Mappings

When you call methods from managed assemblies in PowerScript, you must use PowerBuilder datatypes in any method arguments or return values.

This table shows the mappings between .NET, C#, and PowerBuilder datatypes:

Table 12. Datatype mappings in managed assembly methods

.NET datatype	C# datatype	PowerBuilder 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
```

Calling PowerScript methods from .NET assemblies

If you generate a .NET assembly or Web service from a PowerBuilder target, the generated methods can be called by a different .NET assembly or application, but these calls must be made using .NET syntax and datatypes. In the table for *Datatype mappings in managed assembly methods* on page 179, the datatype mapping is bidirectional, so you can call methods on the .NET assemblies you generate from PowerBuilder using the .NET equivalents for PowerScript datatypes shown in the table.

Some PowerScript datatypes do not have a one-to-one correspondence with datatypes in .NET. When you generate a .NET assembly or Web service from PowerBuilder, PowerBuilder converts these datatypes as shown in the following table. If you call methods using these datatypes from a .NET application, you must substitute the .NET datatype equivalents shown in this table:

Table 13. Mappings for PowerScript datatypes unsupported in .NET

PowerBuilder datatype	C# datatype	.NET datatype
Blob	Byte [ ]	System.Byte [ ]
Date	System.Datetime	System.Datetime
Time	System.Datetime	System.Datetime

Support for .NET language features

You can write conditional code for the .NET environment, taking advantage of features that are not available directly in the PowerBuilder Classic 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* on page 182.
- 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.

  You can use instance references to access static members of .NET classes, as in the following example for the static property "Now" of the System.DateTime class:

  ```
  #if defined PBDOTNET then
      System.DateTime dt_instance
      System.DateTime current_datetime
      dt_instance = create System.DateTime
      current_datetime = dt_instance.Now
  #end if
  ```

  Alternatively, you can access static .NET properties without using instance references, as the following code illustrates:

  ```
  #if defined PBDOTNET then
      System.DateTime current_datetime
      current_datetime = System.DateTime.Now
  #end if
  ```
- Namespaces, interfaces, and user-defined enumerations — you can reference namespaces and .NET interfaces and enumerations in conditional code blocks for the .NET environment. In standard PowerScript code, namespaces are not available and you cannot declare an interface or enumeration.

  See *User-Defined Enumerations* on page 183.
- Function calls on .NET primitive types and enumerations — the **pb2cs** compiler merges functionality of .NET primitive types with the functionality of corresponding PowerBuilder primitive types. Function calls are also supported on .NET enumerated types that you import to a PowerBuilder .NET target.

  See *Function Calls on .NET Primitive and Enumerated Types* on page 184.
- .NET index access — you can access the indexes of .NET classes in the same way you access PowerBuilder array elements.

See *Accessing Indexes for .NET Classes* on page 185.

- Function arguments defined as out parameters — in .NET, functions can pass parameters using the “out” passing mode. Although there is no equivalent concept in PowerScript, you can access “out” parameters—as well as parameters passed by reference—using the “ref” keyword.

  .NET also allows you to overload a function that has a parameter passed by value with a prototype that differs only in the passing mode of the parameter. In these cases, if you want to call the function prototype with the parameter that uses the reference or out passing mode, you must use the ref keyword in your PowerScript call:

  ```
  my_obj.TestMethod(ref l_string)
  ```

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.

This table describes the result of using the bitwise operators:

Table 14. 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.

Operator	Description
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

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.

Declaring .NET enumerations in PowerScript

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 PowerScript, 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.

Function Calls on .NET Primitive and Enumerated Types

You can make function calls on .NET primitive and enumerated types from a PowerBuilder application. The function calls must be made inside a conditional compilation block for a .NET target.

.NET primitive types

To support function calls on .NET primitive types, the PowerBuilder .NET compiler (**pb2cs**) merges the functionality of these primitive types with the functionality of corresponding PowerBuilder primitive types. This allows you to use .NET primitive types and their corresponding PowerBuilder primitive types in a similar fashion. The following example makes the same **ToString** function call on both the .NET *System.Int32* datatype and the PowerScript *long* datatype:

```
System.Int32 i1
long i2
```

```
i1.ToString()
i2.ToString()
```

For a table of equivalencies between .NET and PowerScript primitive datatypes, see *Datatype Mappings* on page 179.

Note: The *System.IntPtr* and *SystemUIntPtr* primitive types do not have precise corresponding types in PowerBuilder—they are always treated as *long* datatypes. Calling functions or modifying properties on these .NET primitive types leads to a compilation error in PowerBuilder.

.NET enumerated types

Function calls are also supported on .NET enumerated types that you import to a PowerBuilder .NET target. For example, suppose you define a .NET enumerated type in a .NET assembly as follows:

```
Public enum TimeOfDay
{
    Morning = 0,
        AfterNoon,
        Evening
}
```

PowerBuilder allows you to call the **ToString** method on the .NET TimeOfDay enumerated type after you import it to your target:

```
#if defined PBDOTNET then
    ns1.ns2.TimeOfDay daytime
    daytime = ns1.ns2.TimeOfDay.Morning!
    daytime.ToString()
#end if
```

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.

This 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
```

Using Multithreading

When you deploy a PowerBuilder application that contains shared objects or an NVO assembly to .NET, the application can be run in a multithreaded environment. The PowerBuilder .NET runtime library also supports .NET synchronization, enabling your application to avoid possible data corruption.

.NET Threading in PowerScript

This PowerScript code fragment uses .NET threading:

```
#if defined PBDOTNET then
//Declare a .NET Class
System.Threading.Thread ithread

//Declare a delegate for .NET Thread
System.Threading.ThreadStart threadproc

//Assign a user defined PowerScript
//function to the delegate
threadproc = f_compute

FOR Count = 1 TO a_count
 ithread = create System.Threading.Thread(threadproc)
    ithread.IsBackground = true
 ithread.Start()
 ithread.sleep(500)
NEXT

#else
    /*action*/
#end if
```

Using .NET Synchronization Functions

To use .NET synchronization functions directly in PowerScript:

1. Declare a global variable.
2. Initialize the global variable.
3. Use the global variable in your .NET synchronization functions. Define your types and functions within #IF DEFINED and #END IF preprocessor statements.

Windows Form example:

```
/* declare and initialize global variable */
System.Object obj
obj = create System.Object

#if defined PBWINFORM then
  System.Threading.Monitor.Enter(obj);
#else
    /*action*/
#end if

b = 1000/globala
globala = 0
```

```
a = 1000

globala = 10
b = a / globala

#if defined PBWINFORM then
  System.Threading.Monitor.Exit(obj);
#else
    /*action*/

#end if

return 1
```

Limitations

There are some important limitations on the code you can enclose in conditional compilation blocks.

- Case sensitivity — PowerScript is case insensitive, but C# is case sensitive. If a resource has the same name as another resource with differences only in the case of one or more characters, PowerBuilder cannot process the resource names correctly.
- Calls to PowerScript from .NET functions — you cannot call a .NET method inside a conditional code block if that method calls back into PowerScript functions.
- Delegates are not supported — a delegate is a type that safely encapsulates a method, similar to a function pointer in C and C++. You cannot use delegates in conditional code blocks.
- .NET classes and interfaces — you cannot use .NET classes and interfaces as parameters to functions and events.
- Inheriting from .NET classes — you cannot create user objects, windows, or window controls that inherit from .NET classes.
- Implementing .NET interfaces — you cannot create user objects that implement .NET interfaces.
- Consuming .NET generics — you cannot consume .NET generic classes or generic methods in conditional code blocks. The .NET Framework 2.0 introduced generics to act as templates that allow classes, structures, interfaces, methods, and delegates to be declared and defined with unspecified or generic type parameters instead of specific types. Several namespaces, such as System Namespace and System.Collections.Generic, provide generic classes and methods.

  The System.Nullable type is a standard representation of optional values and as such it is also classified as generic and therefore cannot be consumed in PowerBuilder .NET applications.

  In .NET Assembly and Web service targets, you can select a 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.

- AutoScript does not support .NET classes — AutoScript works as expected for PowerBuilder objects, properties, and methods inside conditional code blocks, but it does not display for .NET classes.

- DYNAMIC and POST do not support .NET methods — you cannot use the DYNAMIC or POST keywords when you call a .NET method.

- .NET arrays of arrays — .NET arrays of arrays are supported in conditional code blocks for .NET targets only.

Handling Exceptions in the .NET Environment

The PowerBuilder to .NET compiler changes the exception hierarchy used by the native PowerScript compiler.

Modified exception hierarchy

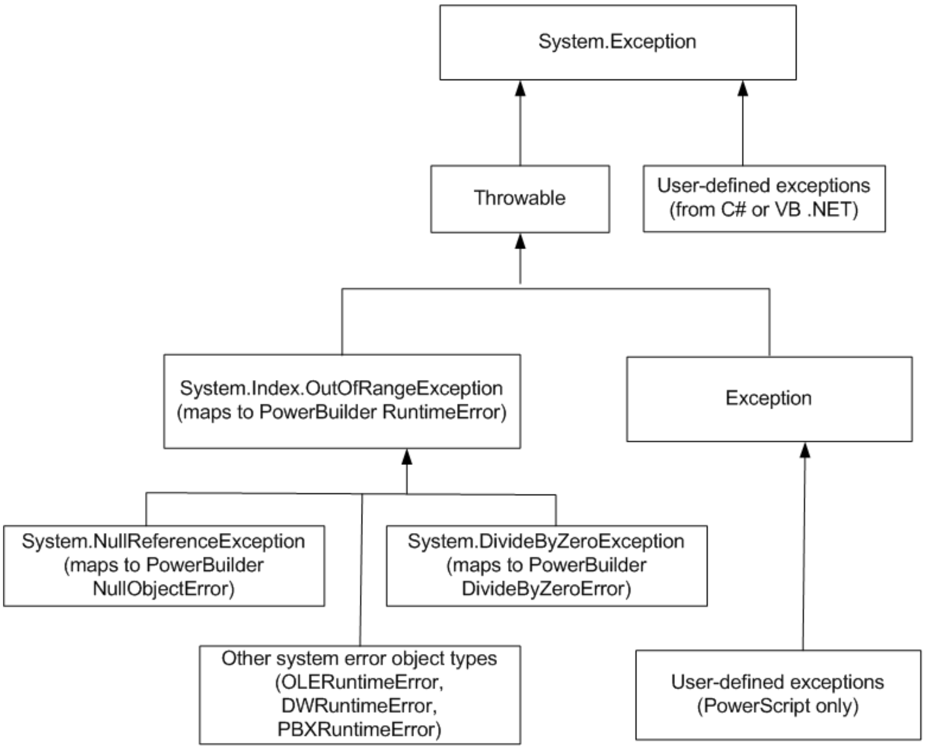
In the native PowerBuilder environment, Throwable is the root datatype for all user-defined exception and system error types. Two other system object types, RuntimeError and Exception, inherit directly from Throwable.

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

This figure shows the exception hierarchy for PowerBuilder applications in the .NET environment:

Figure 4: Exception hierarchy for PowerBuilder in the .NET environment


System.Exception
Throwable
User-defined exceptions (from C# or VB .NET)
System.Index.OutOfRangeException (maps to PowerBuilder RuntimeError)
Exception
System.NullReferenceException (maps to PowerBuilder NullObjectError)
System.DivideByZeroException (maps to PowerBuilder DivideByZeroError)
Other system error object types (OLERuntimeError, DWRuntimeError, PBXRuntimeError)
User-defined exceptions (PowerScript only)

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:

```
namespace ExceptionSample

{

  // Custom exception class used in method second_test(int a) below

  public class MyCustomException : Exception

  {

    public string GetMessage()

    {

       public string GetMessage()

       {

          return "Custom Error Thrown";
       }
    }

    public class Test
    {
      public int division_test (int a)
      {
         int zero = 0;
         // this will throw a System.DivideByZero exception
         return a/zero;
      }
      public int second_test(int a)
      {
         a = a / 2;
         throw new MyCustomException();
      }
  }
}
```

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
     ExceptionSample.Test t
       t = create ExceptionSample.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 ( )
   MessageBox("Exception Error", ls_error)
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
     ExceptionSample.Test t
       t = create ExceptionSample.Test
     i = t.second_test(i)
   catch (ExceptionSample.MyUserException e)
   //this will also work: catch (System.Exception e)
     ls_error = e.getMessage()
       MessageBox("Custom Exception", ls_error)
   end try
#END IF
```

Connections to EAServer Components

You can build a .NET client application or component that invokes methods of Enterprise JavaBeans (EJB) components or PowerBuilder EAServer components running in EAServer 6.1 or later.

This capability is based on the .NET client ORB library introduced in EAServer 6.1.

Note: When you install EAServer, you must install the .NET support option.

You can use either the Connection object or the JaguarORB object to connect to the component in EAServer, and you can connect from .NET Windows Forms and Web Forms applications and from .NET assemblies and Web services.

Using the Connection Object

Build a .NET client application for an EAServer component using the Connection object.

1. Use the Template Application target wizard to create a client application, then use a .NET application wizard to create a .NET target using the library list and application object of the target you just created.

   Alternatively, use a .NET target wizard to build a client application from scratch.
2. Use the EAServer Connection Object Wizard to create a standard class user object inherited from the Connection object. You can then use this object in a script to establish a connection. First set connection options, then call the **ConnectToServer** function.

   If you use the Template Application wizard to create the client application, you can create the Connection object in that wizard.
3. Use the EAServer Proxy Wizard to create a project for building a proxy object for each EAServer component that the .NET client will use, then generate the proxy objects. The EAServer Proxy icons on the Project page of the New dialog box are enabled for all .NET target types.
4. Write the code required to create the EAServer component instance using the **CreateInstance** function.
5. Call one or more component methods from the client.

The steps are the same for .NET clients and standard PowerBuilder clients. For detailed steps, see *Application Techniques > Building an EAServer Client*.

.NET Client Differences

There are some differences you should be aware of when you use a Connection object with a .NET client.

This table lists some properties that have different behavior in .NET client applications. Properties and functions that are obsolete or for internal use only in standard PowerBuilder applications are also unsupported in .NET applications. All other properties, functions, and events are supported.

Property	Description
Driver	Only Jaguar and AppServer are supported values. Any other value results in a runtime error.
ErrorCode	These error codes are supported: • 0 — success • 50 — distributed service error • 57 — not connected • 92 — required property missing or invalid • 100 — unknown error The same error codes are returned by the **ConnectToServer** function.
Options	These options support SSL connections from .NET clients. They are case sensitive and are not available for standard PowerBuilder (Win 32) clients: • ORBclientCertificateFile • ORBclientCertificatePassword See *SSL Connection Support* on page 196.

Connections Using the JaguarORB Object

To create a CORBA-compatible client, you can use the JaguarORB object instead of the Connection object to establish the connection to the server.

The JaguarORB object allows you to access EAServer from PowerBuilder clients in the same way as C++ clients.

Two techniques

The JaguarORB object supports two techniques for accessing component interfaces, using its **String_To_Object** and **Resolve_Initial_References** functions.

Using the **String_To_Object** function works in the same way that the **ConnectToServer** and **CreateInstance** functions on the Connection object do internally. The **String_To_Object** function allows you to instantiate a proxy instance by passing a string argument that describes how to connect to the server that hosts the component. The disadvantage of this approach is that you lose the benefits of server address abstraction that are provided by using the naming service API explicitly.

To use the EAServer naming service API, you can call the **Resolve_Initial_References** function to obtain the initial naming context. However, this technique is not recommended because it requires use of a deprecated SessionManager::Factory **create** method.

Most PowerBuilder clients do not need to use the CORBA naming service explicitly. Instead, they can rely on the name resolution that is performed automatically when they create EAServer component instances using the **CreateInstance** and **Lookup** functions of the Connection object.

See *Application Techniques > Building an EAServer Client*.

.NET client differences

There are some differences you should be aware of when you use a JaguarORB object with a .NET client. The **Init** function has slightly different behavior in .NET client applications:

- You do not need to call the **Init** function to use the JaguarORB object from a .NET client. If you do not call **Init**, the EAServer ORB driver uses the default property values.
- .NET clients support these standard options only:
  - ORBHttp
  - ORBWebProxyHost
  - ORBWebProxyPort
  - ORBHttpExtraHeader
- The following options support mutual authentication in SSL connections from a .NET client. They are case sensitive and are not available for standard PowerBuilder (Win 32) clients:
  - ORBclientCertificateFile
  - ORBclientCertificatePassword

  See *SSL Connection Support* on page 196.

All other properties, functions, and events are supported and work in the same way as in standard PowerBuilder client applications.

Support for CORBAObject and CORBACurrent Objects

The CORBAObject object gives PowerBuilder clients access to several standard CORBA methods. All proxy objects generated for EAServer components using the EAServer proxy generator are descendants of CORBAObject.

The CORBACurrent service object provides information about the EAServer transaction associated with a calling thread and enables the caller to control the transaction. The CORBACurrent object supports most of the methods defined by the EAServer CORBACurrent interface.

All CORBAObject and CORBACurrent properties, functions, and events are supported with .NET clients.

Supported Datatypes

Simple and complex datatypes are convertible between .NET clients and EAServer components.

This table describes the basic CORBA IDL types supported and their corresponding PowerScript type:

CORBA IDL type	Mode	PowerScript type
boolean	in, return	Boolean by value
	out, inout	Boolean by reference
char	in, return	Char by value
	out, inout	Char by reference
octet	in, return	Byte by value
	out, inout	Byte by reference
short	in, return	Integer by value
	out, inout	Integer by reference
long	in, return	Long by value
	out, inout	Long by reference
long long	in, return	Longlong by value
	out, inout	Longlong by reference
float	in, return	Real by value
	out, inout	Real by reference
double	in, return	Double by value
	out, inout	Double by reference
string	in, return	String by value
	out, inout	String by reference
BCD::Binary	in, return	Blob by value
	out, inout	Blob by reference
BCD::Decimal	in, return	Decimal by value
	out, inout	Decimal by reference
BCD::Money	in, return	Decimal by value
	out, inout	Decimal by reference

CORBA IDL type	Mode	PowerScript type
MJD::Date	in, return	Date by value
	out, inout	Date by reference
MJD::Time	in, return	Time by value
	out, inout	Time by reference
MJD::Timestamp	in, return	DateTime by value
	out, inout	DateTime by reference
TabularResults::ResultSet	in, return	ResultSet by value
	out, inout	ResultSet by reference
TabularResults::ResultSets	in, return	ResultSets by value
	out, inout	ResultSets by reference
Void	return	(None)

Arrays and sequences of structures and basic types are also supported. This table lists the complex datatypes that are supported:

CORBA IDL type	Mode	PowerScript type
Array	in	Bounded array by value
	inout	Bounded array by reference
Sequence	in	Unbounded array by value
	inout	Unbounded array by reference
Structure	in, return	Structure by value
	out, inout	Structure by reference

SSL Connection Support

To enable .NET client applications developed in PowerBuilder to connect with EAServer using the Secure Sockets Layer (SSL), the computer where the .NET application runs must be configured to work correctly with the SSL authentication mode.

You can connect using Server authentication or Mutual authentication.

Server Authentication

If only server authentication is required, the EAServer client must provide authentication to the server to prove that the client can be trusted before it can connect to the server.

By default, EAServer 6.x uses 2001 as the port for this type of SSL connection.

Connection Code

In the PowerScript connection code, change the EAServer host's address to a URL that begins with "iiops" and ends with the correct SSL port.

All other code is the same as if the client was connecting to a server without using SSL.

The following sample code connects with EAServer using an SSL connection:

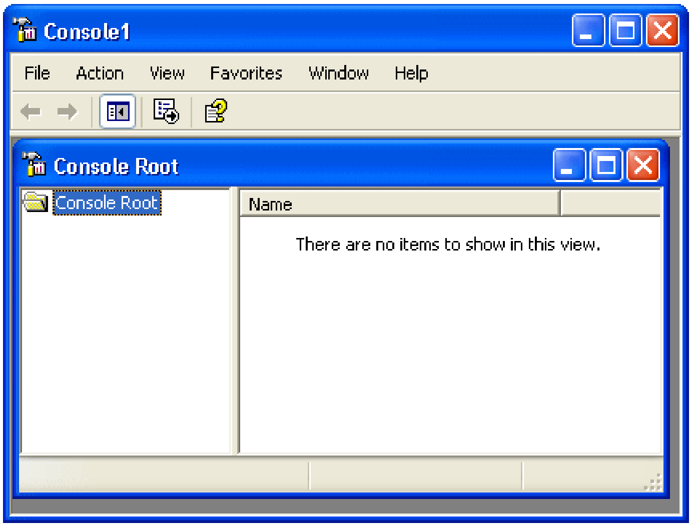
```
Connection myconnect
int rc

myconnect = create Connection
myconnect.Application = "pbtest"
myconnect.Driver = "jaguar"
myconnect.UserID = "admin@system"
myconnect.Password = "abc"
myconnect.Location = "iiops://mydesktop:2001"

rc = myconnect.connecttoserver( )
```

Importing an EAServer Certificate into the Client Certificate Store

The EAServer host's certificate file must be imported into the Microsoft certificate store on the client's computer.

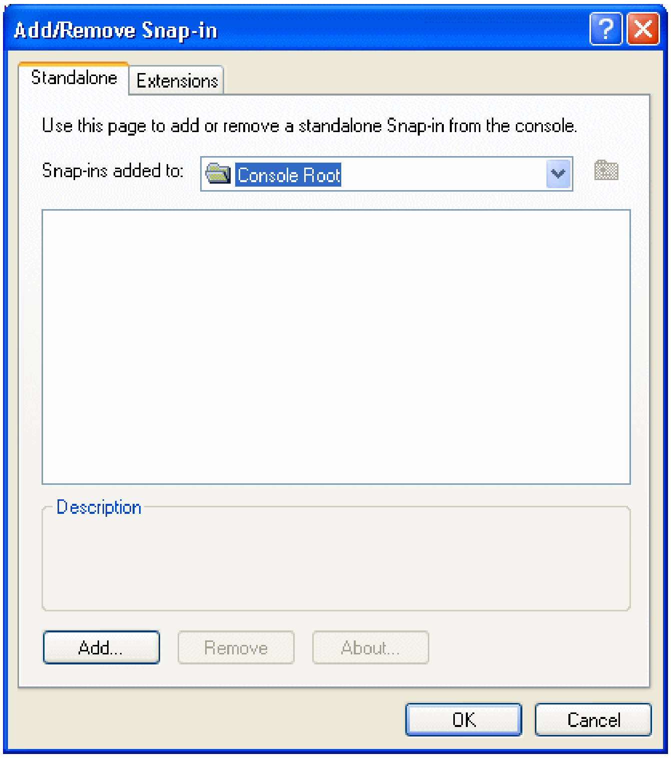
You can do this using the Certificate snap-in in the Microsoft Management Console (MMC).

1. Select **Run** from the Windows Start menu, type `mmc` in the Run dialog box, and click **OK** to open the Microsoft Management Console.



There are no items to show in this view.

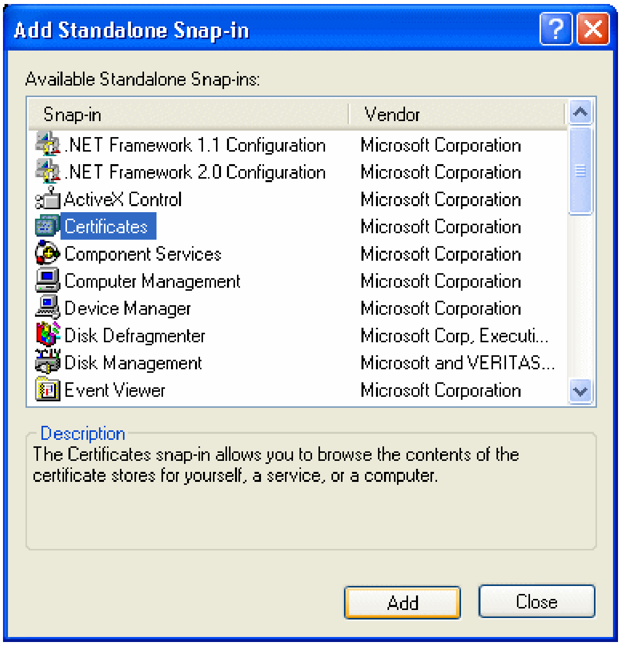
2. Select **File** > **Add/Remove Snap-in** to open the Add/Remove Snap-in dialog box.



Use this page to add or remove a standalone Snap-in from the console.

Snap-ins added to: Console Root

3. Click **Add** to open the Add Standalone Snap-in dialog box.
4. Select **Certificates** from the Snap-in list and click **Add** to open the Certificates Snap-in dialog box.



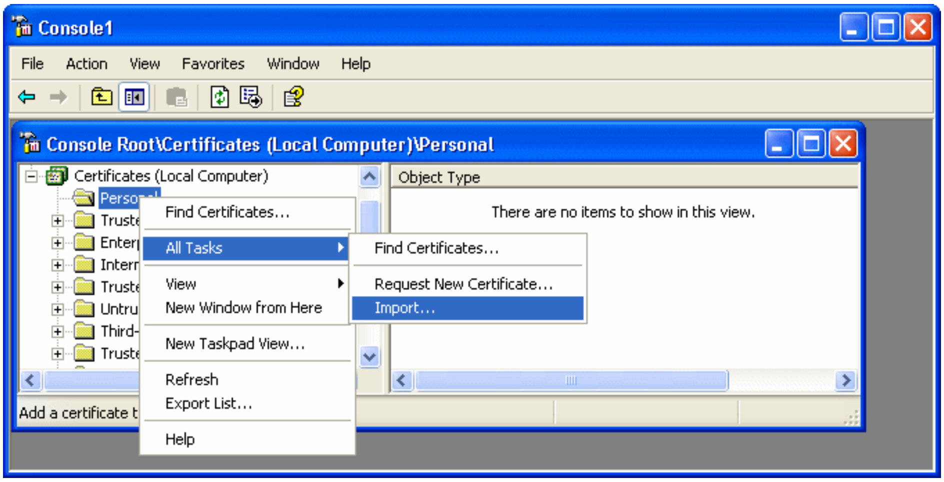
Snap-in	Vendor
.NET Framework 1.1 Configuration	Microsoft Corporation
.NET Framework 2.0 Configuration	Microsoft Corporation
ActiveX Control	Microsoft Corporation
Certificates	Microsoft Corporation
Component Services	Microsoft Corporation
Computer Management	Microsoft Corporation
Device Manager	Microsoft Corporation
Disk Defragmenter	Microsoft Corp, Executi...
Disk Management	Microsoft and VERITAS...
Event Viewer	Microsoft Corporation

Description
The Certificates snap-in allows you to browse the contents of the certificate stores for yourself, a service, or a computer.

5. Select the Computer account radio button, click **Next**, click **Finished**, and close the Add Standalone Snap-in and Add/Remove Snap-in dialog boxes.

   A Certificates node displays in the MMC.

6. Expand the Certificates node in the MMC, right-click **Personal**, select **All Tasks**, and then select **Import**.



There are no items to show in this view.

   The Certificate Import Wizard opens.

7. Follow the instructions in the Certificate Import Wizard to import the certificate.

The wizard prompts you to provide a certificate file. For server authentication, this is the certificate file that is configured as the certificate for EAServer on port 2001 or any other port that is specified for use in server-only authentication SSL mode. You may already have such a file from configuring EAServer for SSL connections, or, if you have access rights to the built-in Java keystore on the EAServer host, you can export the required certificate from the keystore.

For more information about exporting a certificate, see the *EAServer documentation*.

Note: The server's certificate file need not include its private key.

Mutual Authentication

If mutual authentication is required, the server and client must authenticate each other to ensure that both can be trusted.

By default, EAServer 6.x uses 2002 as the port for this type of SSL connection.

Both the server's certificate and the client's certificate must be imported into the Microsoft certificate store on the client computer as described in *Importing an EAServer Certificate into the Client Certificate Store* on page 197.

Note: The client's certificate file must include the private key for the client's certificate. The server's certificate file need not include its private key.

The server certificate used for mutual authentication cannot be the same as the certificate used for server-only authentication. Make sure you obtain the correct certificate file.

For mutual authentication, the client's certificate file must be imported into the certificate store on the client computer *and* it must be available in the file system on the client computer, because it is referenced in the PowerScript code required to connect to EAServer.

Two new key/value pairs in the Options property of the Connection object are used for mutual authentication:

- ORBclientCertificateFile is used to specify the file name of the client certificate file.
- ORBclientCertificatePassword is used to specify the password for the certificate if any. There is no need to use this key if the certificate is not protected by password.

Connection code

In the PowerScript connection code, change the EAServer host's address to a URL that begins with "iiops" and ends with the correct SSL port. The following sample code connects to an EAServer host that requires mutual authentication:

```
Connection myconnect
int rc

myconnect = create Connection

myconnect.Application = "pbtest"
myconnect.Driver = "jaguar"
myconnect.UserID = "admin@system"
myconnect.Password = "sybase"
myconnect.Location = "iiops://mydesktop:2002"
myconnect.Options = "ORBclientCertificateFile=
'd:\work\sample1.p12',ORBclientCertificatePassword =abc"

rc = myconnect.connecttoserver( )
```

Configuration step required for Web Forms and Web services

For mutual authentication, PowerBuilder .NET Web Forms applications and .NET Web services that are clients for EAServer require that the ASPNET account on the IIS server have access to the private key of the client certificate. Access to the private key of the server certificate is not required.

Use the Windows HTTP Services Certificate Configuration Tool (`WinHttpCertCfg.exe`) to configure client certificates. You can download this tool from the *Microsoft Download Center*.

To grant access rights to the private key of the client certificate for the ASPNET account on the IIS server, type the following commands at a command prompt:

```
cd C:\Program Files\Windows Resource Kits\Tools
WinHttpCertCfg -g -c LOCAL_MACHINE\MY -s "ABC" -a "ASPNET"
```

These commands assume that the tool is installed in the default location at `C:\Program Files\Windows Resource Kits\Tools` and that the client certificate's subject name is “ABC”. The `-s` argument is equivalent to the Issued To field in the MMC. The ASPNET account is valid for XP computers. You should use the “NetworkService” account for other Windows platforms. For the `-c` argument, always use “LOCAL_MACHINE\MY” rather than the actual name of the local computer.

For more information about the configuration tool’s options, type `WinHttpCertCfg -help` at the command prompt. For more information about installing client certificates for Web applications and services, see the *Microsoft Help and Support site*.

Best Practices for .NET Projects

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.

Syntax issues

These language-level items apply when you plan to transform a PowerBuilder application to a Windows Forms or Web Forms application.

- 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.
- Do not call 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 this 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."
- Do not cast to object without inheritance relationship — the PowerBuilder compiler allows you to cast an object to classes that are not ancestors of the object you are casting, such as sibling object classes. However, this is not considered good coding practice, and is not allowed for .NET targets.

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_filedatetime 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_filedatetime 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.

  If the code looks like this:

```
char* WINAPI fnReturnEnStrA()
{
  return "ANSI String";
}
```

  it is recommended that you alter it like this:

```
#include <objbase.h>
... ...
char* WINAPI fnReturnEnStrA()
{
  char* s = (char*)CoTaskMemAlloc(12);
  memcpy(s, "ANSI string\0", 12);
  return s;
}
```

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 .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_IUSRS user group (IIS 7 and 7.5) has write permission. Calling an external function to get the current directory from the virtual file system fails.

Use the DESTROY statement

The .NET garbage collection service does not trigger the Destructor event for PowerBuilder objects. If you need to trigger the Destructor event for a nonvisual object, you must explicitly call the PowerScript DESTROY statement for that object.

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 computer. 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.

Use multiple text patterns for string matching

If you want to test whether a string's value contains any of a multiple set of matching text patterns, you can use the pipe character (|) in your .NET applications or components. The pipe character is a metacharacter in the .NET environment that functions as an OR operator, although it is not a metacharacter in the standard PowerBuilder client-server environment.

Therefore, when you call the **Match** function in the .NET environment, you can use pipe characters to determine if either of two (or one of many) text patterns match a string you are evaluating. In standard client-server applications, you can use the **Match** function to evaluate only one text pattern at a time.

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.
- Use global 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 Classic 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 65. 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 the IIS Manager* on page 9.

Global properties also allow you to share data across application sessions. See *Sharing Data Across Sessions* on page 29.

DataWindow Pagination

If the HTMLGen.PageSize property 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.

Note: You set the HTMLGen.PageSize property in the DataWindow painter by selecting the Rows Per Page option on the Web Generation tab.

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.

Note: 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.

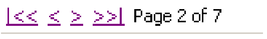
To disable pagination of Web Forms DataWindow objects, set the PBDataWindowRowsPerPage property to -1. To disable pagination for a specific DataWindow object, set its HTMLGen.PageSize property to -1.

DataWindow Page Navigation

There are several global properties related to DataWindow page navigation. Set the navigation bar at the top or the bottom of a DataWindow page by modifying the PBDataWindowNavigationBarPosition property.

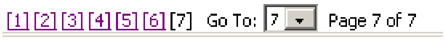
The PBDataWindowPageNavigatorType property lets you select the type of navigation bar you want to use: NextPrev, Numeric, QuickGo, or combined types.

- QuickGo — 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.
- NextPrev — this figure shows the default NextPrev navigation bar:

  It shows page status information with default text for the current and total page count. You can use the PBDataWindowStatusInfoFormat property to modify the text.

  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 show 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.
- NumericWithQuickGo — this figure shows the NumericWithQuickGo navigation bar:



The numeric portion of the navigation bar lists each page by its page number. You can set the PBDataWindowPageNavigatorType 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 PBDataWindowPageNavigatorType property to `NextPrevWithQuickGo`.

Although the QuickGo navigation control appears by default as a drop-down list, you can change this to a text box with an associated command button by setting the PBDataWindowQuickGoPageNavigatorType property to `Button`. You can edit the button label by setting the PBDataWindowGoToButtonText property. You set the label for the text box or the drop-down list by modifying the PBDataWindowGoToDescription 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.

See *Conditional Compilation* on page 173.

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 PowerScript, 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 *Default Event Handlers* on page 33.

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

These DataWindow events can be handled on the client side in JavaScript code:

- Clicked
- ButtonClicking
- ButtonClicked
- DoubleClicked
- ItemChanged
- ItemError
- ItemFocusChanged
- RButtonDown
- RowFocusChanged
- RowFocusChanging

See *Alphabetical Liist of Web DataWindow Client-Side Events* on page 37.

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.

This example references a script called MyDwClickedEventHandler for the client-side DataWindow Clicked event:

```
#IF Defined PBDOTNET THEN
   dw_1.JavaScriptFile = “D:\Scripts\MyScriptFile.js”
   dw_1.OnClientClicked = “MyDWClickedEventHandler”
#END IF
```

The script for the MyDwClickedEventHandler event handler must use the syntax for the client-side Clicked event described in *Clicked* on page 39.

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 *Web Forms Properties* on page 65.

Compiling, Debugging, and Troubleshooting

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

Incremental Builds

Incremental builds allow you to save time while deploying applications for testing or production purposes. For incremental builds, only object classes that are affected by one or more changes are recompiled during the build process.

Target level

The incremental rebuild process for .NET targets is conducted as the first step of a project's deployment to a .NET platform. Although deployment remains at the project level, incremental rebuilds are done at the target level. This means that multiple projects within a single target are able to benefit from this time saving feature by sharing the same incremental build assemblies or .NET modules.

Note: Incremental builds are not available for .NET component targets. The PowerBuilder .NET compiler always does full rebuilds for these target types.

Build and Deploy Directories

When you deploy a .NET application project, PowerBuilder creates a build directory under the directory for the current target.

The name of the build directory is *TargetName*.pbt_build, where *TargetName* is the name of the current target. If the project you deploy has a debug build type, the build files are generated in a "debug" subdirectory of the *TargetName*.pbt_build directory. If the project you deploy has a release build type, the build files are generated in a subdirectory named "release."

The debug and release subdirectories store incremental build results only. PowerBuilder does a full rebuild if files are missing or damaged in one of these subdirectories. The subdirectories or their parent directory cannot be used for a project's output path or working path.

In addition to the debug and release directories, PowerBuilder creates a deploy directory when you first deploy a project from the current target. The deploy directory contains an XML file for each project in the target that you deploy.

Rebuild Scope

An option on the General tab page of .NET Windows Forms and Web Forms Project painters allows you to choose whether to do a full rebuild or an incremental build when deploying a .NET project. The default option is incremental.

If the application has not been previously deployed, a full build is triggered by the PowerBuilder IDE even when the incremental rebuild option is selected. The incremental rebuild option is also overridden if you remove the build directory that PowerBuilder generates from a previous build, or if some of the build files are missing or damaged in the build directory or its subdirectories.

.NET Modules

For a debug build, the PowerBuilder .NET compiler creates a .NET module for each PowerBuilder class or class group. A class group consists of a container object that instantiates a primary class, and the controls in that container object, that are instances of subsidiary classes.

For example, a window normally contains several controls. The window and the controls are declared as separate classes that are bound together as a class group in the .NET build process.

For a release build, the compiler creates a .NET module for each PBL rather than for each class or class group. Although basing the generated .NET modules on classes and class groups increases performance for incremental builds, this is mostly needed at development time when the application is being debugged. At production time, basing the generated .NET modules on target PBLs is more advantageous, since it minimizes the number of modules that need to be deployed.

Incremental rebuilds are supported for deployment to remote servers as well as for MSI file generation. In addition to saving time on deployment, the generation of .NET modules is especially beneficial for smart client Windows Forms applications, because the modules can reduce the size of the assembly files that need to be updated.

PBD Generation

In addition to .NET modules or assemblies, PowerBuilder can generate PBD files for application PBLs containing DataWindow, Query, or Pipeline objects.

Pipeline objects are supported in Windows Forms targets, but are not currently supported in Web Forms targets or in the .NET component targets. The PBD files are linked as external resources with the generated .NET modules and assemblies.

If you use incremental builds for your Windows Forms or Web Forms targets, the PBD files are generated only for selected PBLs in which modifications have been made to DataWindow, Query, or Pipeline objects. For these target types, the PBD files are generated in a “pbd” subdirectory of the *TargetName*.pbt_build directory. The PBD files are deployed together with the generated .NET modules or assemblies. On deployment, they are not deleted from this subdirectory since they are used to check for changes during subsequent incremental builds.

If you use full builds, PBD files are always generated for selected PBLs containing DataWindow, Query, or Pipeline objects even when there are no changes to these objects—although you can prevent generation by clearing the check box next to the PBL name on the Library Files tab page of the Project painter. Since you cannot use incremental builds with .NET component targets, PBD files are always generated by default for these target types.

Triggering Build and Deploy Operations

PowerBuilder lets you trigger build and deploy operations when you run or debug a .NET Web Forms or Windows Forms project.

By default, when you click the running man or debugging icon in the PowerBuilder toolbar, or select Run from a project menu or context menu for one of these target types, PowerBuilder determines if there is a corresponding build directory for the selected target. If there is, PowerBuilder checks whether the .NET modules in the build directory are consistent with the latest changes to each object in your current application.

If implementation or interface changes are detected or if the build directory does not exist for the current target, PowerBuilder displays a message box that tells you the project is out of date and that prompts you to redeploy the project. The message box has three buttons (Yes, No, and Cancel) and a check box that lets you prevent the display of the message box the next time you click or select run or debug.

If you click Yes in the message box, PowerBuilder builds the project using an incremental or full rebuild—depending on the current rebuild scope—and then redeploys it, using the current project's deployment specifications. If you click No in the message box with the redeployment prompt, PowerBuilder attempts to run or debug the currently deployed target even though it is out of date. Clicking Cancel terminates the run or debug request.

If you select the Do not ask me again check box and then click Yes or No, PowerBuilder modifies a drop-down list selection on the General tab of the System Options dialog box.

System Option

Select an option to determine whether a message box appears if you run or debug a project when it is out of date.

The On click Run, if .NET application projects are out of date drop-down list on the General tab of the System Options dialog box controls the appearance of a message box when a project is out of date.

This table describes the selections available in the drop-down list:

Table 15. Drop-down list selections for incremental builds

Selection	Effect when you click or select Run or Debug
Ask me	(Default selection.) Causes a message box to appear if the current project has been modified since the last time it was deployed, or if it has never been deployed before.
Always redeploy	Always redeploys a project before running or debugging it. It first rebuilds the project using the rebuild scope set in the Project painter.
Never redeploy	Never redeploys a project before trying to run it, although it does deploy a project that has not been previously deployed, and then attempts to run or debug that project. (Do not use this option to debug a project that you have previously deployed.)

The message box that prompts you to redeploy an out-of-date project appears only when the drop-down list selection is “Ask me.” This selection changes automatically to “Always redeploy” if you click Yes in the message box when the “Do not ask me again” option is selected. It changes to “Never redeploy” if you click No. You can always reset the option from the System Options dialog box.

Incremental Build Processing

When you save recently edited code, the PowerBuilder IDE invokes the PowerScript compiler to get information for updating the System Tree and the property sheet.

There are basically three kinds of changes that the compiler handles:

- Implementation changes, such as modifications to a function body or to the properties of a class.
- Interface changes, such as the removal of a function or the modification of a function prototype.
- Data changes, including edits made to a DataWindow, Query, or Pipeline object.

The IDE collects the information that has changed, performs a full or incremental PowerScript rebuild, and passes the necessary information to the **pb2cs** .NET translator. If the PowerScript compiler reports any errors the IDE does not invoke the .NET translator.

An interface change that is successfully compiled by the PowerScript compiler and then passed to **pb2cs** can also affect code in classes that are compiled in a different .NET module of the same target. In this case, if you rebuild the project using the incremental rebuild process, the .NET runtime throws an exception when you try to run the application.

PowerBuilder catches and translates .NET runtime exceptions to error messages describing the exception source. Before redeploying the application, you can correct this type of error by changing the PowerScript code based on the contents of the error message or by performing a full rebuild. If there are many places in other .NET modules affected by the interface change, it is best to do a full rebuild.

If you only make data changes to DataWindow objects before an incremental rebuild, the .NET rebuild process is skipped entirely and only application PBD files are redeployed.

Debugging a .NET Application

After you have deployed a PowerBuilder Web Forms or Windows Forms application, you can debug it.

1. To open the debugger, you can:
   - Right-click the target or project in the System Tree and select **Debug** from its context menu.
   - Open the project to debug, and select **Design > Debug Project** from the Project painter menu bar.
   - Make sure the application you want to debug is current and select **Debug *applicationName*** in the PainterBar.
2. To start the debugging process:
   - From the Debugger toolbar, select **Start *applicationName*** .
   - From the Debugger menu, select **Debug > Start *applicationName***.

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:

1. In the Project painter, select **Run > Attach to .NET Process**.
2. In the dialog box that opens, select the process you want to attach to.
   After you attach to the process, it starts running in the debugger and you can set breakpoints as you normally do.

Next

Select **Run > Detach** to detach from the process. This gives you more flexibility than simply using just-in-time (JIT) debugging.

.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 appear in the Local Variables view in .NET targets.

Additional debugging restrictions include the following:

- Debugger icons appear 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 shows the next line of code. However, if you step into a different event script, the debugger continues execution to the next breakpoint. 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.
- Setting breakpoints in modified code — if you modify your code after successfully debugging a .NET application, you must redeploy the application before you debug it again. Although you can still set breakpoints in modified lines of code before you redeploy an application, the debugger debugs only the last deployed version of your application.
- 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.
- 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 *Users Guide > Debugging an application*.

Release and Debug Builds

If you choose to compile an application or component as a debug build, an extra file with the extension .PDB is generated in the output directory, and additional information is included in the Output window.

Select a build type for your application or component on the General page in the Project painter. 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

Enable the DEBUG preprocessor symbol if you want to add code to your application to help you debug while testing the application.

This is a selection on the General tab of the Project painter. 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.

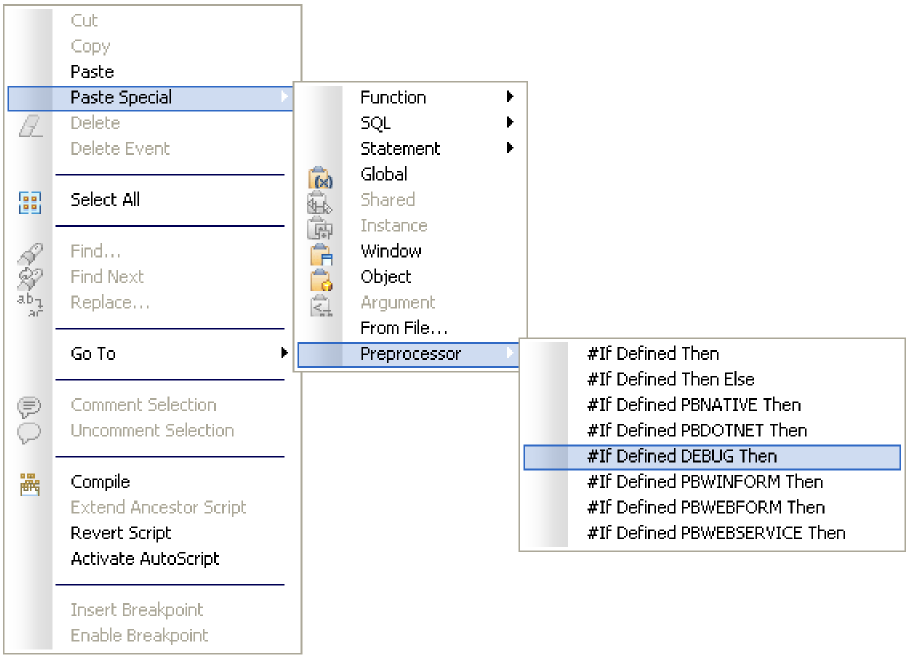
When the DEBUG symbol is enabled, code that is enclosed in a code block with the following format is parsed by the **pb2cs** code emitter:

```
#if defined DEBUG then
   /*debugging code*/
#else
   /* other action*/
#end if
```

Note: 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.

This figure shows the context menu item that you can use to paste the **#If Defined DEBUG Then** template statement in the Script view:



For information about using preprocessor symbols such as DEBUG, see *Conditional Compilation* on page 173.

Breaking into the Debugger When an Exception is Thrown

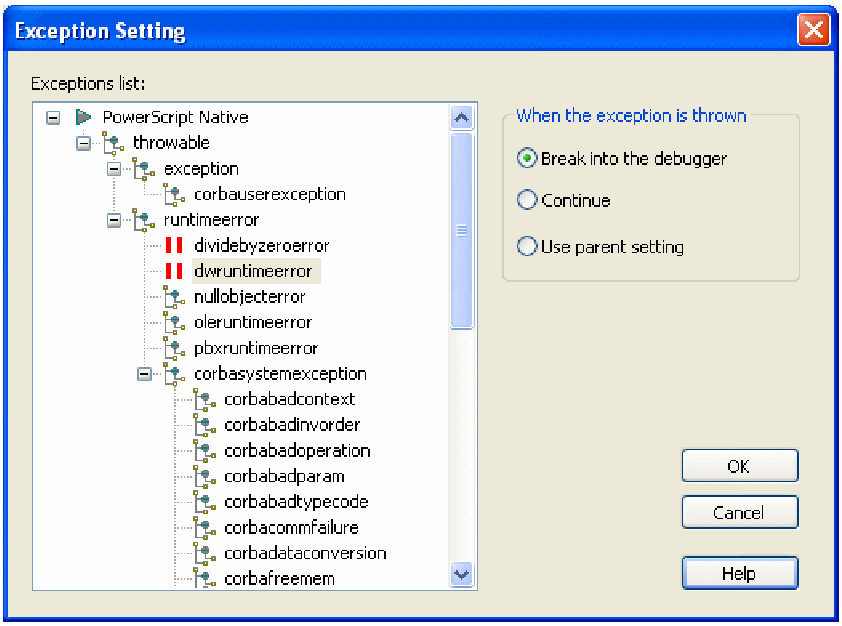
When an application throws an exception while it is being debugged, the debugger sees the exception before the program has a chance to handle it.

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.

This figure shows the DWRuntimeError exception has been set to Break into the debugger:


Exception Setting

Exceptions list:

- PowerScript Native
  - throwable
    - exception
      - corbauserexception
    - runtimeerror
      - dividebyzeroerror
      - dwruntimeerror
      - nullobjecterror
      - oleruntimeerror
      - pbxruntimeerror
      - corbasystemexception
        - corbabadcontext
        - corbabadinorder
        - corbabadoperation
        - corbabadparam
        - corbabadtypecode
        - corbacommfailure
        - corbadataconversion
        - corbafreemem

When the exception is thrown

[✓ Break into the debugger] Continue Use parent setting

When this exception is thrown, a dialog box lets you choose whether to open the debugger or pass the exception to the program.

Debugging a .NET Component

You can debug .NET components as well as .NET applications that you build in PowerBuilder.

.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. See *Version, Post-build, and Run tab* on page 161 for a description of 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.

If you use a PowerBuilder .NET Windows Forms or .NET Web Forms application to debug the .NET component project, you must copy the generated PDB file containing the DEBUG symbols for the component to the deployment directory of the .NET Windows Forms or .NET Web Forms application. Otherwise it is likely that the debugger will not stop at breakpoints in the assembly that you generate from the .NET component project.

Troubleshooting .NET Targets

Troubleshooting tips for PowerBuilder .NET applications and components can help you diagnose and correct deployment and runtime errors.

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 `pbc2cs.exe` file is present in the `PowerBuilder 12.5\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 *Datatype Mappings* on page 179 for a list of datatype mappings from .NET to PowerBuilder.

If your application uses conditional compilation blocks, see *Limitations* on page 187 to make sure that you have not used any .NET classes, interfaces, or methods in ways that are not supported. See also *Best Practices for .NET Projects* on page 201 and *Design-Level Considerations* on page 204.

Errors that display in the Output window with a C0 prefix, such as error C0312, are generated by the PowerBuilder compiler. There is a link from these errors back to the source code in PowerBuilder painters. Explanations for PowerBuilder compiler errors can be found in the online help.

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*.

Troubleshooting Runtime Errors

If a Web Forms application shows a blank page, or if any .NET application or component produces unexpected errors, make sure that the PowerBuilder runtime files on the target computer or server have the same version and build number as the PowerBuilder files on the development computer.

Troubleshooting Tips for Web Forms Applications

Review the suggestions in this section if you experience difficulty deploying, running, or updating a Web Forms application.

Also review the known issues listed in the PowerBuilder *Release Bulletin*.

Web Forms Deployment Errors

Inadequate write permissions for a deployment directory, or a deployment computer with a hyphen in its name, may prevent you from deploying Web Forms applications.

Using a 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 computer)

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.

Host name issue

Web Forms deployment fails with an "IIS Server not found" error when the host name of the computer running IIS contains a hyphen. You must remove the hyphen from the computer name before deployment can be successful.

Vista or later platform requirement

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

Browser Error Messages

Runtime errors for Web Forms applications are often posted in the client browser.

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 4.0 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 4.0 is installed on the server computer, register ASP.NET 4.0 with IIS by running `aspnet_regiis.exe -I` from the .NET Framework 4.0 directory, and make sure ASP.NET 4.0 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 .NET assemblies in the server environment. To resolve this issue, remove extra copies of PowerBuilder .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 computer. 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 `pbdwm`*`version`*`.dll` or `pbshr`*`version`*`.dll`, or the Microsoft Visual C++

runtime libraries `msvcr71.dll`, `msvcp71.dll`, `msvcp100.dll`, `msvcr100.dll`, and the Microsoft .NET Active Template Library (ATL) module, `atl71.dll`. To resolve this issue, make sure the DLLs are available on the server and that the directory where the DLLs are located is included in the system path on the server.

Unexpected error was thrown

When a runtime JavaScript error occurs, the application terminates and the following error message displays: “Unexpected error was thrown, the browser will be closed!” Using a column or field that has a JavaScript keyword for its name can cause this type of error.

Failure to Connect to a Database

Limited access rights for ASP.NET users is a common cause for the inability to connect to a database from a Web Forms application.

DSN

Due to limited access rights of ASP.NET user and user group accounts, data sources created as User DSNs may not 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_IUSRS user group (IIS 7 and 7.5) 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_IUSRS 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_IUSRS 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, IIS 7, and IIS 7.5). Alternatively, you can restart the IIS server to make sure that the changes take effect.

DataWindows Are Not Visible

Make sure the PBL files that contain the DataWindows used by a Web Forms application are copied to the directory specified by 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.

On Vista and later, TreeView DataWindows are not visible if you have not given the IIS_IUSRS group full permissions (including write and delete) to application directories. See *Application Directory Permissions* on page 13.

Pictures Are Not Visible

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 Do Not Load

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 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.

See *Requirements for Saving Files in PDF or XSL Format* on page 54.

Log Files

A PowerBuilder application that compiles successfully with the PowerBuilder native compiler might not compile successfully with the PowerBuilder to .NET compiler. Log files help you trace compiler and runtime errors.

Log.txt

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 *Best Practices for .NET Projects* on page 201. 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 with IIS 7.5

When you deploy a PowerBuilder Web Forms application to IIS 7 or later, the application is deployed to a PowerBuilder-specific application pool named PBDotnet4AppPool.

PBDotnet4AppPool uses the default application pool identity, which is NetworkService on IIS 7, and ApplicationPoolIdentity on IIS 7.5. However several Web Forms application features, including the creation of permanent user accounts, SSL authentication, and DataWindow **Print** and **SaveAs** commands, do not work with the default IIS 7.5 application pool identity. To enable these features, you must change the PBDotnet4AppPool application pool identiity to NetworkService.

See *Changing Application Pool Identity for IIS 7.5* on page 12.

Troubleshooting Tips for Windows Forms Applications

Review the suggestions in this section if you experience difficulty deploying, running, publishing, or updating a Windows Forms application.

Make sure you have installed the .NET Framework and SDK as described in *System Requirements for .NET Windows Forms Targets* on page 121, and review the known issues for Windows Forms applications listed in the PowerBuilder *Release Bulletin*.

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"/>
   <!-- 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, `pbshr125.dll` and `pbdwm125.dll`, and the Microsoft runtime files on which they depend, `at71.dll`, `msvcp100.dll`, `msvcr100.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 *Setting Full Trust Permissions* on page 132.
- If the application cannot connect to a database, make sure that the required PowerBuilder database interface, such as `pbodb125.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.

These errors may 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 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.

Note: 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.

These errors may 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, troubleshoot the problem by verifying files, locations, and network connections.

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 intended, and that the update files are in the specified location.

Appendix

The appendix describes custom permissions you can set on the Security tabs of Web Forms, Web Service, and Windows Forms projects.

Custom Permission Settings

You can set custom permissions for .NET Windows Forms and Web Forms applications, and for .NET Web Service components, in the Project painter Security tab.

Most of the permission classes that you can customize are defined in the System.Security.Permissions namespace. For more information on these permission classes, see the Microsoft Web site at *http://msdn.microsoft.com/en-us/library/system.security.permissions.aspx*.

Adding Permissions in the .NET Framework Configuration Tool

The list of permissions that display in the Security tab permissions list box is the same as the list in the "Everything" permission set of the .NET Framework 4.0 SDK Configuration tool runtime security policy.

To add permission settings that are not in the custom permissions list:

1. Close PowerBuilder if it is open, and create an XML file with the permission settings you want to add.

   For example, by default, the SMTPPermission setting is not included in the assigned permissions in the "Everything" permission set. To create this permission, save a file named `SMTPPermission.xml` with the following content:

   ```
   <IPermission class="System.Net.Mail.SmtpPermission, System,
   Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" version="1" Unrestricted="true"/
   >
   ```
2. Open the .NET Framework SDK Configuration tool from the Administrative Tools folder in your computer Control Panel.
3. In the left pane of the configuration tool, select **My Computer** > **Runtime Security Policy** > **Machine** > **Permission Sets** > **Everything**, then select the **Action** > **Change Permissions** menu item.
4. In the Create Permission Set dialog box, click **Import** to open the Import a Permission dialog box, browse to the `SMTPPermission.xml` file, and click **OK**.
5. Click **Finish**, close the configuration tool, and open a .NET project in PowerBuilder to the Security tab page.

The SMTPPermission displays in the list box of the Security tab page. You can scroll the list to see it when you select any radio button option other than Full Trust.

EnvironmentPermission

In a .NET Windows Forms application, you must have minimal "Read" EnvironmentPermission settings if your application uses the **GetContextKeywords** function.

The default setting is "Unrestricted='true'" when the EnvironmentPermission check box is selected on the Security tab of the Project painter, although you can change this to "Read" and still use the **GeContextKeywords** function. If you modify the setting to "Write" or "NoAccess", **GetContextKeywords** will fail.

Table 16. EnvironmentPermission required in Windows Forms

System function	Permission required
GetContextKeywords	Read

You can customize the EnvironmentPermission setting to allow the use of the **GetContextKeywords** function in XML, as in this sample setting:

```
<IPermission
class="System.Security.Permissions.EnvironmentPermission, mscorlib,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
version="1"
Read="Path " />
```

EventLogPermission

EventLogPermission settings are required for the PBTraceTarget global property for .NET targets.

You can set this property on the Configuration tab of the Project painter for Web Forms and Web Service targets. For Windows Forms targets, you must set this after deployment in the `*appname*.exe.config` file (where *appname* is the name of the executable file).

Table 17. EventLogPermission required in .NET targets

Global .NET property	Permission required
PBTraceTarget when the value is set to file	Windows Forms: `Pbtrace.log` FileIO permission asserted by default in runtime library Web Forms: `*appname*_root\log\pbtrace.log` FileIO permissions added by default when property set in IDE
PBTraceTarget when the value is set to EventLog	Administer

In Windows and Web Forms targets, if PBTraceTarget is set to "EventLog", the application needs Administer permission to write to the log. You can set this in Security tab as follows:

```
<IPermission class=" EventLogPermission" version="1">
    <Machine name="testmachine" access="Administer"/>
</IPermission>
```

FileDialogPermission

FileDialogPermission settings are required for the **GetFileOpenName** and **GetFileSaveName** functions in Windows Forms targets.

Table 18. FileDialogPermission required in Windows Forms

System function	Permission required
GetFileOpenName	Open or OpenSave (unrestricted FileIOPermission also required)
GetFileSaveName	Save or OpenSave (unrestricted FileIOPermission also required)

FileIOPermission

FileIOPermission settings are required for PowerScript system functions in Windows Forms targets.

Permission requirements in Web Forms

For Web Forms targets, file-releated functions require Read, Write, Append and PathDiscovery permissions on the `appname_root` directory. For print-related functions, you must set the PrintingPermission value to AllPrinting, in addition to setting Read, Write, and PathDiscover permissions on the `appname_root/print` directory.

Permission requirements for Windows Forms

Table 19. FileIOPermission required for system functions in Windows Forms

System function	Permission required
AddToLibraryList,**DirectoryExists** , **FileEncoding**, **FileExists**, **FileLength**, **FileLength64**, **FileRead**, **FileReadEx**, **FileSeek**, **FileSeek64**, **LibraryDirectory**, **LibraryDirectoryEx**, **LibraryExport**, **PrintBitmap**, **ProfileInt**, **ProfileString**, **SetProfileString**, **SetLibraryList**, **ShowHelp**, **ShowPopupHelp**, and **XMLParseFile**	Read
FileWrite, **FileWriteEx**, **RemoveDirectory**, **LibraryCreate**, **LibraryDelete**, and **LibraryImport**	Write
FileDelete	Read and Write
FileOpen	When the FileAccess argument is Read! • Read permission for file named in FileName (first) argument When the FileAccess argument is Write! • Append and Write permission when the WriteMode argument is Append! • Read and Write permission when the WriteMode argument is Replace!
FileCopy (string s, string t)	Read for the source file (first) argument; Write for the target file (second) argument
FileMove (string s, string t)	Read and Write for the source file (first) argument; Write for the target file (second) argument
GetFolder	Unrestricted
GetCurrentDirectory	PathDiscovery for the current directory
CreateDirectory (string d)	Read for the parent directory; Write for the directory name argument

This table shows the required FileIOPermission settings for object and control functions in Windows Forms targets.

Object or control	Function or property	Permission required
Animation	**AnimationName**	Read
DataWindow	**SaveAsAscii** , **SaveAsFormattedText** , **Sa-vInk** , **SaveInkPicture**	Write
	ImportFile	Read permission if the (usually second) file name argument is supplied; if file name argument is emp-ty or null, requires Open-Save FileDialogPermis-sion and Unrestricted Fil-eIOPermission
	SaveAs	Write permission if the (usually second) file name argument is supplied; if file name argument is emp-ty or null, requires Open-Save FileDialogPermis-sion and Unrestricted Fil-eIOPermission
DataWindow (RichText only)	**InsertDocument**	Read
DataStore	**ImportFile**	Read
	SaveAs, **SaveAsAscii** , **SaveAsFormat-tedText** , **SavInk** , **SaveInkPicture**	Write
DragObject	DragIcon property	Read
DropDownListBox, List-Box	**DirList** (string s, uint filetype)	Read and PathDiscovery for the file specification (first) argument
DropDownPictureList-Box, PictureListBox	**AddPicture**, and PictureName property	Read
	DirList (string s, uint filetype)	Read and PathDiscovery for the file specification (first) argument
Graph	**ImportFile**	Read

Object or control	Function or property	Permission required
	SaveAs	Unrestricted for function with no arguments; Write on the file name (first) argument for function with arguments
InkPicture	PictureFileName property	Read
	LoadInk	Read and Write for the file (first) argument
	LoadPicture	Read for the file (first) argument
	Save, **SaveInk**	Write for the file (first) argument and for the current temporary file directory
Listview	**AddLargePicture**, **AddSmallPicture**, **AddStatePicture**, and LargePictureName, SmallPictureName, StatePictureName properties	Read
Picture	PictureName property	Read
PictureButton	DisabledName, PictureName properties	Read
RichTextEdit	**InsertDocument**, **InsertPicture**	Read
	SaveDocument	Write
Treeview	**AddPicture**, **AddStatePicture**, and PictureName, StatePictureName properties	Read
UserObject	PictureName property	Read

This XML example gives Read access to two files and write access to one of those files:

```
<IPermission class="System.Security.Permissions.FileIOPermission,
mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" version="1" Read="d:\test.txt;c:
\demo.jpg" Write="c:\demo.jpg" />
```

This example for a Web Forms application grants Read, Write, Append, and PathDiscovery permissions to the main deployment and root directories for the MyWebApp application:

```
<IPermission class="FileIOPermission" version="1"
Read="MyWebApp_root;MyWebApp" Write="MyWebApp_root;MyWebApp"
```

```
Append="MyWebApp_root;MyWebApp"
PathDiscovery="MyWebApp_root;MyWebApp"/>
```

PrintingPermission

PrintingPermission settings are required for PowerScript system functions in Windows Forms targets.

Permission requirements for system functions

Table 20. Printing Permission required for system functions in Windows Forms

System function	Permission required
Print, **PrintBitmap**, **PrintCancel**, **PrintClose**, **PrintDataWindow**, **PrintDefineFont**, **PrintGetPrinter**, **PrintScreen**, **PrintSend**, **PrintSetFont**, **PrintSetSpacing**, **PrintLine**, **PrintOpen**. **PrintOval**, **PrintPage**, **PrintRect**, **PrintRoundRect**, **PrintSetupPrinter**, **PrintText**, **PrintWidth**, **PrintX**, **PrintY**	DefaultPrinting or AllPrinting
PrintGetPrinters, **PrintSetPrinter**, **PrintSetup**	AllPrinting

This table shows the required PrintingPermission settings for object and control functions in Windows Forms targets.

Table 21. PrintingPermission required for object or control functions in Windows Forms

Object or control	Function or property	Permission required
DataWindow	**Print** with no arguments	DefaultPrinting or AllPrinting
	Print (canceldialog, true)	AllPrinting
DataStore	**Print**	DefaultPrinting or AllPrinting
DragObject	**Print**	DefaultPrinting or AllPrinting
RichTextEdit	**PrintEx** (cancelDialog)	DefaultPrinting or AllPrinting
Window	**Print**	DefaultPrinting or AllPrinting

This example allows printing to the default printer and the use of a restricted printer selection dialog box:

```
<IPermission
  class="System.Drawing.Printing.PrintingPermission, System.Drawing,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
version="1" Level="DefaultPrinting"/>
```

ReflectionPermission

ReflectionPermission settings are required for PowerScript reflection functions and objects in .NET targets.

Table 22. ReflectionPermission required in .NET targets

System function or object	Permission required
FindClassDefinition, **FindTypeDefinition**	TypeInformation
ScriptDefinition object	TypeInformation

This permission setting in Windows Forms targets allows reflection for members of a type that are not visible:

```
<IPermission class=
   "System.Security.Permissions.ReflectionPermission,
   mscorlib, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" version="1"
   Flags="TypeInformation" />
```

This permission setting in Web Forms targets allows reflection for members of a type that are not visible:

```
<IPermission class="ReflectionPermission" version="1"
   Flags="TypeInformation"/>
```

RegistryPermission

RegistryPermission settings are required for system registry functions and MLSync object functions in .NET targets.

Table 23. Required RegistryPermission settings for system functions

System function	Permission required
RegistryGet, **RegistryKeys**, **RegistryValues**	Read
RegistrySet	Write; if registry key does not exist, requires Create
RegistryDelete	Read and Write

This table shows the required RegistryPermission settings for MLSync object functions in .NET targets:

Table 24. Required RegistryPermission settings for MLSync functions

MLSync function	Permission required
GetObjectRevisionFromRegistry, **GetsSyncRegistryProperties**	Read on HKEY_CURRENT_USER registry key
GetDBMLSyncPath	Read on the Software\Sybase\SQL Anywhere registry keys under HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE
SetsSyncRegistryProperties	Unrestricted on HKEY_CURRENT_USER registry key

This example for a Windows Forms application grants read permission for the HKEY_CURRENT_USER registry key, which extends to its subkeys:

```
   <IPermission
class="System.Security.Permissions.RegistryPermission,
   mscorlib, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" version="1"
   Read="HKEY_CURRENT_USER" />
```

This example for a Web Forms application grants read permission for the HKEY_CURRENT_USER registry key, which extends to its subkeys:

```
   <IPermission class="RegistryPermission" version="1
   Read="HKEY_CURRENT_USER" />
```

SecurityPermission

Execution permission is required for a SecurityPermission setting on all .NET applications and for any managed code that you want a user to run.

This table shows the required SecurityPermission settings for functions and objects in Windows Forms targets.

Table 25. SecurityPermission required in Windows Forms targets

Function, object, property, or feature	Permission required
OLEControl	Unrestricted (or the Full Trust option)
ChangeDirectory, Handle, Post, Restart, Run, Send	UnmanagedCode
URL (PictureHyperlink and StaticHyperlink property),	UnmanagedCode
HyperlinkToURL (Inet property)	UnmanagedCode

Function, object, property, or feature	Permission required
Language interoperation feature	Variable permissions required, depending on .NET function called or property accessed
Win32 API feature	UnmanagedCode

This table shows the required SecurityPermission settings for interactions with .NET or Win32 functions and properties in Web Forms targets.

Table 26. SecurityPermission required in Web Forms targets

Feature	Permission required
Language interoperation	Variable permissions required, depending on .NET function called or property accessed
Win32 API	UnmanagedCode

This example sets required security permissions for Windows Forms targets:

```
   <IPermission
class="System.Security.Permissions.SecurityPermission,
   mscorlib, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" version="1"
   Flags="Assertion, Execution, BindingRedirects,
   UnmanagedCode" />
```

This example sets required security permissions for Web Forms targets

```
   <IPermission class="SecurityPermission" version="1"
   Flags="Assertion, Execution, ControlThread,
   ControlPrincipal, RemotingConfiguration,
   UnmanagedCode"/>
```

SMTPPermission

An SMTPPermission setting is required for the MailSession object log on function in .NET targets.

Table 27. SMTPPermission required in .NET targets

MailSession object function	Permission required
MailLogon	Connect (if using default port) or ConnectToUnrestrictedPort

This permission setting allows a Windows Forms application to log onto a mail session and receive mail through a default port:

```
<IPermission class="System.Net.Mail.SmtpPermission,
System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" version="1"
Access="Connect"/>
```

This permission setting in a Web Forms target allows application users to log onto a mail provider and receive mail through any available port:

```
<IPermission class=" SmtpPermission" version="1"
Access="ConnectToUnrestrictedPort"/>
```

SocketPermission

A SocketPermission setting is required for the Connection object **ConnectToServer** function in .NET targets.

The SocketPermission class belongs to the System.Net namespace described on the Microsoft Web site at *http://msdn.microsoft.com/en-us/library/system.net.aspx*.

Table 28. SocketPermission required in .NET targets

Connection object function	Permission required
ConnectToServer	Connect

Thispermission setting allows a Windows Forms application to get or set a network access method:

```
<IPermission class="System.Net.SocketPermission,
System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" version="1">
<ConnectAccess>
   <ENDPOINT host="10.42.144.40" transport="Tcp"
   port="2000"/>
</ConnectAccess>
</IPermission>
```

This permission setting allows a Web Forms application to get or set a network access:

```
<IPermission class="SocketPermission" version="1">
<ConnectAccess>
   <ENDPOINT host="10.42.144.40" transport="Tcp"
   port="2000"/>
</ConnectAccess>
</IPermission>
```

SQLClientPermission

A SocketPermission setting is required for the database connection feature in .NET targets.

Table 29. SQLClientPermission required in .NET targets

Feature	Permission required
Database connect (including pipeline functionality for Windows Forms clients)	Unrestricted

This permission setting allows database connections for a Windows Forms application:

```
<IPermission class=
"System.Data.SqlClient.SqlClientPermission,
System.Data, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" version="1"
Unrestricted="true" />
```

This permission setting allows database connections for a Web Forms application:

```
 <IPermission class=" SqlClientPermission" version="1"
 Unrestricted="true" />
```

UIPermission

The Unrestricted UIPermission setting is required for Windows Forms applications, although you can customize the setting to use a combination of AllowDrop and AllWindow permission values.

The UIPermission setting has no effect on WebForms applications.

WebPermission

WebPermission settings are required for features and functions in .NET targets.

The WebPermission class belongs to the System.Net namespace described on the Microsoft Web site at *http://msdn.microsoft.com/en-us/library/system.net.aspx*.

Table 30. WebPermission required in .NET targets

Function or feature	Permission required
GetURL (Inet function)	Connect for *urlname* argument
PostURL (Inet function)	Connect for *urlname* and *serverport* arguments
Web Service call feature	Unrestricted="true"

This permission setting allows a Windows Forms application to connect to the Sybase Web site:

```
<IPermission class="System.Net.WebPermission, System,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" version="1">
<ConnectAccess>
  <URI uri="http://www.sybase.com"/>
</ConnectAccess>
</IPermission>
```

This permission setting allows a Web Forms application to connect to a remote host:

```
<IPermission class="WebPermission" version="1">
<ConnectAccess>
  <URI uri="$OriginHost$"/>
</ConnectAccess>
</IPermission>
```

Custom Permission Types

Permission types that you can customize on the Security tab page of the Project painter (besides the permissions described elsewhere in this appendix) have no direct impact on PowerScript functions or properties in .NET targets.

However, if you use the language interoperation feature of PowerBuilder, this may also require customized permissions for the following permission types:

- ASPNETHostingPermission
- ConfigurationPermission
- DataProtectionPermission
- DNSPermission
- IsolatedStoragePermission
- KeyContainerPermission
- OleDBPermission
- PerformanceCounterPermission
- StorePermission

Index