Deploying Applications and Components to .NET

PowerBuilder® 12.5.2
Contents

Windows Forms Targets ..........................................................1
Choosing a Windows Forms Application Target ............1
   How .NET Deployment Works ........................................2
Security Settings ................................................................3
Strong-Named Assemblies .............................................5
ASP.NET Configuration for a .NET Project ...............6
Checklist for Deployment ...............................................10
PowerBuilder Windows Forms Applications ..............16
Deploying to a production environment ..................17
System Requirements for .NET Windows Forms
   Targets ........................................................................18
Creating a .NET Windows Forms Project ..............20
Deployment of a Windows Forms Application ..........25
Project Execution .........................................................25
Intelligent Deployment and Update .............................26
   Publishing an application for the first time ..........26
Application Installation on the User’s Computer
   ...............................................................................31
Publication of Application Updates .........................32
Application Bootstrapping .......................................35
Rolling Back .................................................................37
MobiLink Synchronization ........................................37
Unsupported Features in Windows Forms Projects ....37
   Unsupported Nonvisual Objects and Structures
      in Windows Forms ...............................................38
   Unsupported System Functions in Windows
      Forms ..................................................................44
PowerBuilder Visual Controls in Windows Forms
   Applications ..........................................................44
Unsupported Functions for Controls in Windows
   Forms .....................................................................49
Contents

PrintingPermission ......................................................126
ReflectionPermission ................................................127
RegistryPermission ...................................................128
SecurityPermission ..................................................128
SMTPPermission ......................................................129
SocketPermission .....................................................130
SQLClientPermission ...............................................130
UIPermission ..........................................................130
WebPermission ........................................................131
Custom Permission Types ..........................................131

Index ..............................................................................133
Windows Forms Targets

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

Choosing a Windows Forms Application Target

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 Windows Forms applications with and without the smart client feature.

<table>
<thead>
<tr>
<th>Application type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| 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 26.

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 Windows Forms applications with relative ease.
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 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 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.


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:
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 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 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 follows:

```xml
<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.


**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:

```powerbuilder
#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 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.

For more information about handling .NET exceptions, see *Handling Exceptions in the .NET Environment* on page 90.

**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 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 113.

### 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 .NET projects generate. The Assembly group box contains the following fields:

<table>
<thead>
<tr>
<th>Assembly group box field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign the assembly</td>
<td>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.</td>
</tr>
<tr>
<td>Choose a strong name key file</td>
<td>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.</td>
</tr>
</tbody>
</table>
Assembly group box field | Description
---|---
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 \texttt{Sn.exe} (in the .NET Framework) with the \texttt{-Vr} option to skip verification during development.
Mark the assembly with \texttt{AllowPartiallyTrustedCallerAttribute} (.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 28.

**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 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 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 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 8.
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 9."

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 NVO Web service deployment:
     a. In the IIS Manager, open the Application Pools node underneath the machine node.
     b. Right-click the PBDotNet4AppPool filter and choose Advanced Settings.
     c. Set the .NET Framework Version to 4.0.

Setting Up a SQL Anywhere Database Connection

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

Before a PowerBuilder .NET Web Service 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 tab 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.

**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 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:

<table>
<thead>
<tr>
<th>Settings for</th>
<th>Required changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows firewall</td>
<td>Enable exceptions for WMI and file and printer sharing</td>
</tr>
</tbody>
</table>
### 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. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DCOM</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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, .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 11.
Checklist for .NET Web Service targets
For .NET Web Service targets, production servers must have:

- IIS 5 or later (See IIS Installation on page 7)
- ASP.NET (See Selecting the Default ASP.NET Version on page 7)
- 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 8.

For information on different methods for deploying .NET Web Service components, see Deployment to a production server on page 72.

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:
## Windows Forms Targets

<table>
<thead>
<tr>
<th>File name</th>
<th>Required for</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbshr125.dll</td>
<td>All .NET targets</td>
</tr>
<tr>
<td>Sybase.PowerBuilder.ADO.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Common.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Core.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Interop.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Web.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Win.dll</td>
<td></td>
</tr>
<tr>
<td>pbrth125.dll</td>
<td>ADO.NET</td>
</tr>
<tr>
<td>pbdwm125.dll</td>
<td>DataWindows and DataStores</td>
</tr>
<tr>
<td>Sybase.PowerBuilder.DataWindow.Win.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Datawindow.Interop.dll</td>
<td></td>
</tr>
<tr>
<td>pbdpl125.dll</td>
<td>Data pipelines (Windows Forms only)</td>
</tr>
<tr>
<td>Sybase.PowerBuilder.EditMask.Win.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Graph.Web.dll</td>
<td>Graphs</td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Graph.Win.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Graph.Core.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Graph.Interop.dll</td>
<td></td>
</tr>
<tr>
<td>File name</td>
<td>Required for</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>pbrtc125.dll</td>
<td>Rich text</td>
</tr>
<tr>
<td>Sybase.PowerBuilder.RTC.Win.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.RTC.Interop.dll</td>
<td></td>
</tr>
<tr>
<td>tp13.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_bmp.flt</td>
<td></td>
</tr>
<tr>
<td>tp13_css.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_doc.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_gif.flt</td>
<td></td>
</tr>
<tr>
<td>tp13_hnm.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_ic.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_ic.ini</td>
<td></td>
</tr>
<tr>
<td>tp13_jpg.flt</td>
<td></td>
</tr>
<tr>
<td>tp13_obj.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_pdf.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_png.flt</td>
<td></td>
</tr>
<tr>
<td>tp13_rtf.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_tif.flt</td>
<td></td>
</tr>
<tr>
<td>tp13_tls.dll</td>
<td></td>
</tr>
<tr>
<td>tp13_wmf.flt</td>
<td></td>
</tr>
<tr>
<td>tp13_wnd.dll</td>
<td></td>
</tr>
<tr>
<td>tp4ole13.ocx</td>
<td></td>
</tr>
<tr>
<td>PBXerces125.dll</td>
<td>XML export and import</td>
</tr>
<tr>
<td>xerces-c_2_6.dll</td>
<td></td>
</tr>
<tr>
<td>xerces-depdom_2_6.dll</td>
<td></td>
</tr>
</tbody>
</table>
### Windows Forms Targets

<table>
<thead>
<tr>
<th>File name</th>
<th>Required for</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExPat125.dll</td>
<td>Web service clients</td>
</tr>
<tr>
<td>libeay32.dll</td>
<td></td>
</tr>
<tr>
<td>ssleay32.dll</td>
<td></td>
</tr>
<tr>
<td>xerces-c_2_6.dll</td>
<td></td>
</tr>
<tr>
<td>xerces-depdom_2_6.dll</td>
<td></td>
</tr>
<tr>
<td>EasySoap125.dll</td>
<td></td>
</tr>
<tr>
<td>pbnetwsruntime125.dll</td>
<td></td>
</tr>
<tr>
<td>pbsoapclient125.pbxml</td>
<td></td>
</tr>
<tr>
<td>pbwsclient125.pbxml</td>
<td></td>
</tr>
<tr>
<td>pblab125.ini</td>
<td>Label DataWindow presentation style</td>
</tr>
<tr>
<td>pbtra125.dll</td>
<td>Database connection tracing</td>
</tr>
<tr>
<td>pbtrs125.dll</td>
<td></td>
</tr>
</tbody>
</table>

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 15.

You must also install the database interfaces your application uses:

<table>
<thead>
<tr>
<th>File name</th>
<th>Required for</th>
</tr>
</thead>
<tbody>
<tr>
<td>pbin9125.dll</td>
<td>Informix I-Net 9 native interface</td>
</tr>
<tr>
<td>pbo90125.dll</td>
<td>Oracle9i native interface</td>
</tr>
<tr>
<td>pbo10125.dll</td>
<td>Oracle 10g native interface</td>
</tr>
<tr>
<td>pbsnc125.dll</td>
<td>SQL Native Client for Microsoft SQL Server native interface</td>
</tr>
<tr>
<td>pbdir125.dll</td>
<td>Sybase DirectConnect™ native interface</td>
</tr>
<tr>
<td>File name</td>
<td>Required for</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------</td>
</tr>
<tr>
<td>pbase125.dll</td>
<td>Sybase Adaptive Server® Enterprise native interface (Version 15 and later)</td>
</tr>
<tr>
<td>pbsyc125.dll</td>
<td>Sybase Adaptive Server Enterprise native interface</td>
</tr>
<tr>
<td>pbado125.dll</td>
<td>ADO.NET standard interface</td>
</tr>
<tr>
<td>pbrth125.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.Db.dll</td>
<td></td>
</tr>
<tr>
<td>Sybase.PowerBuilder.DbExt.dll</td>
<td></td>
</tr>
<tr>
<td>pbjvm125.dll</td>
<td>JDBC standard interface</td>
</tr>
<tr>
<td>pbjdb125.dll</td>
<td></td>
</tr>
<tr>
<td>pbjdbc12125.jar</td>
<td></td>
</tr>
<tr>
<td>pbodb125.dll</td>
<td>ODBC standard interface</td>
</tr>
<tr>
<td>pbodb125.ini</td>
<td></td>
</tr>
<tr>
<td>pbole125.dll</td>
<td>OLE DB standard interface</td>
</tr>
<tr>
<td>pbodb125.ini</td>
<td></td>
</tr>
</tbody>
</table>

### 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.

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
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.

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, see Best Practices for .NET Projects on page 103.

For tables of unsupported and partially supported objects, controls, functions, events, and properties, see Unsupported Features in Windows Forms Projects on page 37.

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 20.

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 26.

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 25.

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\texttt{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*/
#endif
```

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 75.

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 10.
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`.


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 7.

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 80.

.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 26.

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 20.

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 20.

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 20.
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 26.

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

<table>
<thead>
<tr>
<th>Tab page</th>
<th>Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>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 110. 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 75.</td>
</tr>
<tr>
<td>Resource Files</td>
<td>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 32.</td>
</tr>
<tr>
<td>Tab page</td>
<td>Properties</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 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 29. |
| 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 24. For information about customized permission settings, see *Security Settings* on page 3 and *Custom Permission Settings* on page 121. |
| 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. |
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 5 and Digital Certificates on page 28.

**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 26.

**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.

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 118.

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 37.

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, see *Best Practices for .NET Projects* on page 103.

**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 111. For information about the system option, see *System Option* on page 111.

For information on debugging .NET Windows Forms targets, see *Debugging a .NET Application* on page 113.

### 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 26
- Publish the application on page 29

**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 32.

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.

<table>
<thead>
<tr>
<th>Intelligent Updater field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sign the manifests</td>
<td>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.</td>
</tr>
<tr>
<td>Select from Store</td>
<td>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.</td>
</tr>
</tbody>
</table>
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\pbui\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 24.

### 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.
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 redeployes 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 118.
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 35.

- **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.
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 35.

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:

```plaintext
string is_datafilename
long li_datafileid

is_datafilename="datafile.txt"

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

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

For information about using preprocessor symbols such as PBWINFORM, see Conditional Compilation on page 75.

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 37.

**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.

- **Version: 1.0.0.1**
- **Check for Update**
- **Retrieve Update**
- **Restart with New Version**
- **Poll for Updates**
- **Options**

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 29.

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 15.

If you do not require all the files included in the package, you can create your own package. See Prerequisites Page Customizations on page 36.

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 32, 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.
Windows Forms Targets

- 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 81 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 18.

- 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**

<table>
<thead>
<tr>
<th>Class name</th>
<th>Supported</th>
<th>Partially supported</th>
<th>Unsupported</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdoResultSet</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ArrayBounds</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClassDefinition *</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ClassDefinitionObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Connection</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConnectionInfo</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ConnectObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContextInformation</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContextKeyword</td>
<td></td>
<td>XX</td>
<td></td>
</tr>
<tr>
<td>CorbaCurrent</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CorbaObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CorbaSystemException (and its descendents)</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CorbaUnion</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CorbaUserException</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataStore</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataWindowChild</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DivideByZeroError</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DWObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DWRuntimeError</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynamicDescriptionArea</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DynamicStagingArea</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnumerationDefinition</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>EnumerationItemDefinition</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ErrorLogging</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Windows Forms Targets

<table>
<thead>
<tr>
<th>Class name</th>
<th>Supported</th>
<th>Partially supported</th>
<th>Unsupported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graxis</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrDispAttr</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inet</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InternetResult</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JaguarOrb</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailFileDescription</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailMessage</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailRecipient</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MailSession</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NonVisualObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NullObjectError</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OleObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OleRuntimeError</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OleStorage</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OleStream</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OleTxnObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OmObject</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OmStorage</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OmStream</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orb</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PBDOM</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PbRuntimeError</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipeline</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProfileCall</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProfileClass</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProfileLine</td>
<td>XX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class name</td>
<td>Supported</td>
<td>Partially supported</td>
<td>Unsupported</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-----------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ProfileRoutine</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>Profiling</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>RemoteObject</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>ResultSet</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResultSets</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RuntimeError</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ScriptDefinition</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Service</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SimpleTypeDefinition</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SSLCallback</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>SSLServiceProvider</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Throwable</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Timing</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TraceActivityNode</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceBeginEnd</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceError</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceESQL</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceFile</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceGarbageCollect</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceTreeLine</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceTreeNode</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceTreeObject</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceTreeRoutine</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceTreeUser</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>TraceUser</td>
<td></td>
<td></td>
<td>XX</td>
</tr>
<tr>
<td>Transaction</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TransactionServer</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>TypeDefinition</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Class name</td>
<td>Supported</td>
<td>Partially supported</td>
<td>Unsupported</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------</td>
<td>---------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>VariableCardinalityDefinition</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VariableDefinition</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>WSConnection</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* 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**

<table>
<thead>
<tr>
<th>Class name</th>
<th>Unsupported functions</th>
<th>Unsupported events</th>
<th>Unsupported properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>• SetLibraryList</td>
<td>• None</td>
<td>• ToolbarUserControl</td>
</tr>
<tr>
<td></td>
<td>• SetTransPool</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CorbaSystemException (and its descendants)</td>
<td></td>
<td></td>
<td>• Class</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Line</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Number</td>
</tr>
<tr>
<td>DataStore</td>
<td>• CopyRTF</td>
<td>• Destructor</td>
<td>• None</td>
</tr>
<tr>
<td></td>
<td>• GenerateHTMLForm</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GenerateResultSet</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GetStateStatus</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• InsertDocument</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PasteRTF</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Print (supported but not for data with rich text formatting)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataWindowChild</td>
<td>• DBErrorCode</td>
<td>• None</td>
<td>• None</td>
</tr>
<tr>
<td></td>
<td>• DBErrorMessage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SetRedraw</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SetRowFocusIndicator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Class name</td>
<td>Unsupported functions</td>
<td>Unsupported events</td>
<td>Unsupported properties</td>
</tr>
<tr>
<td>----------------------------</td>
<td>---------------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>------------------------</td>
</tr>
</tbody>
</table>
| OmObject                   | • GetAutomationNativePointer  
• SetAutomationLocale  
• SetAutomationTimeOut | • None             | • None                 |
| RuntimeError (and its descendants) |                                                                                     |                    | • Class  
• Line  
• Number |
| ScriptDefinition            |                                                                                      | • AliasName  
• ExternalUserFunction (supported for external functions only)  
• LocalVariableList  
• Source  
• SystemFunction | |
| SimpleTypeDefinitio n      |                                                                                      | • LibraryName     |                        |
| TypeDefinition             |                                                                                      | • LibraryName     |                        |
| VariableDefinition         |                                                                                      | • InitialValue (supported for instance variables and primitive types)  
• IsConstant (supported for instance variables)  
• OverridesAncestorValue  
• ReadAccess (supported for instance variables)  
• WriteAccess (supported for instance variables) |
Unsupported System Functions in Windows Forms

Most PowerBuilder system functions are supported in Windows Forms applications.

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDE functions</td>
<td>CloseChannel, ExecRemote, GetCommandDDE, GetCommandDDEOrigin, GetDataDDE, GetDataDDEOrigin, GetRemote, OpenChannel, RespondRemote, SetDataDDE, SetRemote, StartHotLink, StartServerDDE, StopHotLink, StopServerDDE</td>
</tr>
<tr>
<td>Garbage collection functions</td>
<td>GarbageCollectGetTimeLimit, GarbageCollectSetTimeLimit</td>
</tr>
<tr>
<td>Miscellaneous functions</td>
<td>PBGetMenuString</td>
</tr>
<tr>
<td>Input method functions</td>
<td>IMEGetCompositionText, IMEGetMode, IMESetMode</td>
</tr>
<tr>
<td>Profiling and tracing functions</td>
<td>TraceBegin, TraceClose, TraceDisableActivity, TraceDump, TraceEnableActivity, TraceEnd, TraceError, TraceOpen, TraceUser</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Class name</th>
<th>Supported</th>
<th>Partially supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Checkbox</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>CommandButton</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>DataWindow</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DatePicker</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DropDownListBox</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DropDownPictureListBox</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>EditMask</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Graph</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GroupBox</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>HProgressBar</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>HScrollBar</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>HTrackBar</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>InkEdit</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>InkPicture</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Line</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ListBox</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ListView</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ListViewItem</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Menu</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>MenuCascade</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>MonthCalendar</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>MultiLineEdit</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OleControl</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>OleCustomControl</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OmCustomControl</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OmEmbeddedControl</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Oval</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Class name</td>
<td>Supported</td>
<td>Partially supported</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Picture</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PictureButton</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>PictureHyperLink</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>PictureListBox</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RadioButton</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Rectangle</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RichTextEdit</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RoundRectangle</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SingleLineEdit</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>StaticHyperLink</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>StaticText</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Tab</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>TreeView</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>TreeViewItem</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>UserObject</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>VProgressBar</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>VScrollBar</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>VTrackBar</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Window</td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

Table 6. Unsupported functions, events, and properties by control

<table>
<thead>
<tr>
<th>Supported control</th>
<th>Unsupported functions</th>
<th>Unsupported events</th>
<th>Unsupported properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataWindow</td>
<td>• DBErrorCode</td>
<td>• Other</td>
<td>• RightToLeft</td>
</tr>
<tr>
<td></td>
<td>• DBErrorMessage</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GenerateHTML-Form</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• GetStateStatus</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Supported control</td>
<td>Unsupported functions</td>
<td>Unsupported events</td>
<td>Unsupported properties</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------</td>
<td>-------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>DatePicker</td>
<td>• GetCalendar</td>
<td>• DoubleClicked</td>
<td>• AllowEdit</td>
</tr>
<tr>
<td></td>
<td>• Resize (does not support changing height, otherwise supported)</td>
<td>• Other</td>
<td>• Border</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• BorderStyle</td>
</tr>
<tr>
<td>DropDownListBox</td>
<td>• None</td>
<td>• Other</td>
<td>• HScrollBar</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• IMEMode</td>
</tr>
<tr>
<td>Graph</td>
<td>• None</td>
<td>• Other</td>
<td>• BorderStyle</td>
</tr>
<tr>
<td></td>
<td>• AddData, GetDataValue, InsertData, ModifyData do not support string values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ListBox</td>
<td>• None</td>
<td>• Other</td>
<td>• TabStop</td>
</tr>
<tr>
<td>ListView</td>
<td>• AddColumn, InsertColumn, SetColumn limitation: the alignment of the first column cannot be set to center or right</td>
<td>• Other</td>
<td>• IMEMode</td>
</tr>
<tr>
<td>Menu</td>
<td>• None</td>
<td>• Help</td>
<td>• MenuItemType</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• MergeOption</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ToolBarAnimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ToolBarHighlightColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ToolBarItemSpace</td>
</tr>
<tr>
<td>MenuCascade</td>
<td>• None</td>
<td>• Help</td>
<td>• Columns</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CurrentItem</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DropDown</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• MenuItemType</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• MergeOption</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ToolBarAnimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ToolBarHighlightColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ToolBarItemSpace</td>
</tr>
<tr>
<td>Supported control</td>
<td>Unsupported functions</td>
<td>Unsupported events</td>
<td>Unsupported properties</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------</td>
<td>----------------------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>MonthCalendar</td>
<td>• None</td>
<td>• DoubleClicked</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Other</td>
<td></td>
</tr>
<tr>
<td>MultiLineEdit</td>
<td>• None</td>
<td>• Other</td>
<td>• IMEMode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• TabStop</td>
</tr>
<tr>
<td>OmCustomControl</td>
<td>• None</td>
<td>• None</td>
<td>• Alignment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Cancel</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Default</td>
</tr>
<tr>
<td>OmEmbedded-Control</td>
<td>• _Get_DocFileName</td>
<td>• None</td>
<td>• Activation</td>
</tr>
<tr>
<td></td>
<td>• _Get_ObjectData</td>
<td></td>
<td>• ContentsAllowed</td>
</tr>
<tr>
<td></td>
<td>• _Set_ObjectData</td>
<td></td>
<td>• DisplayType</td>
</tr>
<tr>
<td></td>
<td>• Drag</td>
<td></td>
<td>• DocFileName</td>
</tr>
<tr>
<td></td>
<td>• InsertClass</td>
<td></td>
<td>• LinkUpdateOptions</td>
</tr>
<tr>
<td></td>
<td>• InsertFile</td>
<td></td>
<td>• ObjectData</td>
</tr>
<tr>
<td></td>
<td>• InsertObject</td>
<td></td>
<td>• ParentStorage</td>
</tr>
<tr>
<td></td>
<td>• LinkTo</td>
<td></td>
<td>• Resizable</td>
</tr>
<tr>
<td></td>
<td>• Open</td>
<td></td>
<td>• SizeMode</td>
</tr>
<tr>
<td></td>
<td>• PasteLink</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• PasteSpecial</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SaveAs</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• SelectObject</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• UpdateLinksDialog</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PictureButton</td>
<td>• None</td>
<td>• Other</td>
<td>• Map3DColors</td>
</tr>
<tr>
<td>StaticHyperLink</td>
<td>• None</td>
<td>• Other</td>
<td>• BorderColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• FillPattern</td>
</tr>
<tr>
<td>StaticText</td>
<td>• None</td>
<td>• Other</td>
<td>• BorderColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• FillPattern</td>
</tr>
<tr>
<td>Tab</td>
<td>• None</td>
<td>• Other</td>
<td>• BackColor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• RaggedRight (see Tab properties on page 53)</td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>Function</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddItem</td>
<td>UserObject</td>
</tr>
<tr>
<td>CloseChannel</td>
<td>Window</td>
</tr>
</tbody>
</table>

*Note: See Tab properties on page 53 for tab page properties.*
<table>
<thead>
<tr>
<th>Function</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBErrorCode</td>
<td>DataWindow</td>
</tr>
<tr>
<td>DBErrorMessage</td>
<td>DataWindow</td>
</tr>
<tr>
<td>DeleteItem</td>
<td>UserObject</td>
</tr>
<tr>
<td>Drag</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>EventParmDouble</td>
<td>UserObject</td>
</tr>
<tr>
<td>EventParmString</td>
<td>UserObject</td>
</tr>
<tr>
<td>ExecRemote</td>
<td>Window</td>
</tr>
<tr>
<td>GenerateHTMLForm</td>
<td>DataWindow</td>
</tr>
<tr>
<td>_Get_DocFileName</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>_Get_ObjectData</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>GetStateStatus</td>
<td>DataWindow</td>
</tr>
<tr>
<td>GetCommandDDE</td>
<td>Window</td>
</tr>
<tr>
<td>GetCommandDDEOrigin</td>
<td>Window</td>
</tr>
<tr>
<td>GetDataDDE</td>
<td>Window</td>
</tr>
<tr>
<td>GetDataDDEOrigin</td>
<td>Window</td>
</tr>
<tr>
<td>GetRemote</td>
<td>Window</td>
</tr>
<tr>
<td>InsertClass</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>InsertFile</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>InsertItem</td>
<td>UserObject</td>
</tr>
<tr>
<td>LinkTo</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>Open</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>OpenChannel</td>
<td>Window</td>
</tr>
<tr>
<td>PasteLink</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>PasteSpecial</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>Resize</td>
<td>DatePicker (only changing height is unsupported)</td>
</tr>
<tr>
<td>RespondRemote</td>
<td>Window</td>
</tr>
<tr>
<td>SaveAs</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>SelectObject</td>
<td>OmEmbeddedControl</td>
</tr>
</tbody>
</table>
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**

<table>
<thead>
<tr>
<th>Event</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>DoubleClicked</td>
<td>DatePicker, MonthCalendar</td>
</tr>
<tr>
<td>Help</td>
<td>Menu, MenuCascade</td>
</tr>
<tr>
<td>Notify</td>
<td>TreeView</td>
</tr>
<tr>
<td>Other</td>
<td>All controls</td>
</tr>
<tr>
<td>Resize</td>
<td>DatePicker</td>
</tr>
<tr>
<td>UserString</td>
<td>DatePicker</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Property</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment</td>
<td>OmCustomControl</td>
</tr>
<tr>
<td>AllowEdit</td>
<td>DatePicker</td>
</tr>
<tr>
<td>Activation</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>BackColor</td>
<td>Tab, UserObject (see <em>Tab properties</em> on page 53)</td>
</tr>
<tr>
<td>Border</td>
<td>DatePicker</td>
</tr>
<tr>
<td>BorderColor</td>
<td>StaticHyperLink, StaticText</td>
</tr>
<tr>
<td>BorderStyle</td>
<td>DatePicker, Graph</td>
</tr>
<tr>
<td>Cancel</td>
<td>OmCustomControl</td>
</tr>
<tr>
<td>Columns</td>
<td>MenuCascade</td>
</tr>
<tr>
<td>ColumnsPerPage</td>
<td>UserObject</td>
</tr>
<tr>
<td>ContentsAllowed</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>CurrentItem</td>
<td>MenuCascade</td>
</tr>
<tr>
<td>Default</td>
<td>OmCustomControl</td>
</tr>
<tr>
<td>DisplayType</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>DocFileName</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>DropDownList</td>
<td>MenuCascade</td>
</tr>
<tr>
<td>FillPattern</td>
<td>StaticHyperLink, StaticText</td>
</tr>
<tr>
<td>Height</td>
<td>DatePicker</td>
</tr>
<tr>
<td>Help</td>
<td>Menu, MenuCascade</td>
</tr>
<tr>
<td>HScrollbar</td>
<td>DropDownListBox</td>
</tr>
<tr>
<td>IMEMode</td>
<td>All controls</td>
</tr>
<tr>
<td>LinkUpdateOptions</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>Map3DColors</td>
<td>PictureButton</td>
</tr>
<tr>
<td>MenuItemType</td>
<td>Menu</td>
</tr>
<tr>
<td>MergeOption</td>
<td>Menu</td>
</tr>
<tr>
<td>ObjectData</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>ParentStorage</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>RaggedRight</td>
<td>Tab (see <em>Tab properties</em> on page 53)</td>
</tr>
</tbody>
</table>
### Property Controls

<table>
<thead>
<tr>
<th>Property</th>
<th>Controls</th>
</tr>
</thead>
<tbody>
<tr>
<td>RightToLeft</td>
<td>DataWindow, ListBox, ListView, TreeView</td>
</tr>
<tr>
<td>SizeMode</td>
<td>OmEmbeddedControl</td>
</tr>
<tr>
<td>StatePictureHeight</td>
<td>TreeView</td>
</tr>
<tr>
<td>StatePictureWidth</td>
<td>TreeView</td>
</tr>
<tr>
<td>Style</td>
<td>UserObject</td>
</tr>
<tr>
<td>TabStop</td>
<td>ListBox, MultiLineEdit</td>
</tr>
<tr>
<td>TabTextColor</td>
<td>UserObject (see Tab properties on page 53)</td>
</tr>
<tr>
<td>ToolbarAnimation</td>
<td>Menu</td>
</tr>
<tr>
<td>ToolbarHighlightColor</td>
<td>Menu</td>
</tr>
<tr>
<td>ToolbarItemSpace</td>
<td>Menu</td>
</tr>
</tbody>
</table>

**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.
Windows Forms Targets
.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:

<table>
<thead>
<tr>
<th>Wizard field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name</td>
<td>Name of the project object the wizard creates.</td>
</tr>
<tr>
<td>Library</td>
<td>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.</td>
</tr>
<tr>
<td>Target</td>
<td>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.</td>
</tr>
<tr>
<td>Library search path</td>
<td>Lets you add PBLs and PBDs to the search path for the new target.</td>
</tr>
<tr>
<td>PowerBuilder object name</td>
<td>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.</td>
</tr>
<tr>
<td>Description</td>
<td>Lets you add a description for the project object the wizard creates.</td>
</tr>
</tbody>
</table>
### 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 59.
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:

<table>
<thead>
<tr>
<th>Wizard field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose a target</td>
<td>Select a target from the list of targets in the current workspace.</td>
</tr>
<tr>
<td>Specify a project name</td>
<td>Select a name for the project you want to create. You must create a project object to deploy nonvisual objects as .NET components.</td>
</tr>
<tr>
<td>Choose a project library</td>
<td>Specify a library from the list of target libraries where you want to store the new project object.</td>
</tr>
<tr>
<td>Choose NVO objects to be deployed</td>
<td>Expand the library node or nodes in the list box and select check boxes next to the nonvisual objects that you want to deploy.</td>
</tr>
<tr>
<td>Use .NET nullable types</td>
<td>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.</td>
</tr>
<tr>
<td>Only include functions with supported datatypes</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 55.

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

<table>
<thead>
<tr>
<th>Project painter field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug or Release</td>
<td>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.</td>
</tr>
<tr>
<td>Enable DEBUG symbol</td>
<td>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.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Project painter field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom class</td>
<td>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.</td>
</tr>
<tr>
<td>Object name, Class name, and Namespace</td>
<td>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.</td>
</tr>
<tr>
<td>Project painter field</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Method names and Function prototypes</td>
<td>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.</td>
</tr>
<tr>
<td>Change method name and description</td>
<td>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.</td>
</tr>
<tr>
<td>Select All and Unselect All</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 55.

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:

<table>
<thead>
<tr>
<th>Project painter field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version tab: Product name, Company, Description, and Copyright</td>
<td>Use these fields to specify identification, description, and copyright information that you want to associate with the assembly you generate for the project.</td>
</tr>
<tr>
<td>Version tab: Product version, File version, and Assembly</td>
<td>Enter major, minor, build, and revision version numbers for the product, file, and assembly.</td>
</tr>
</tbody>
</table>
### Project painter field

<table>
<thead>
<tr>
<th>Post-build tab: Post-build command line list for build type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select the build type (Debug or Release) and click <strong>Add</strong> 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.</td>
<td></td>
</tr>
</tbody>
</table>

| 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 5.

### Supported Datatypes
The PowerBuilder to .NET compiler converts PowerScript datatypes to .NET datatypes.

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

<table>
<thead>
<tr>
<th>PowerScript datatype</th>
<th>C# datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>bool</td>
</tr>
<tr>
<td>blob</td>
<td>byte [ ]</td>
</tr>
<tr>
<td>byte</td>
<td>byte</td>
</tr>
<tr>
<td>int, uint</td>
<td>short, ushort</td>
</tr>
</tbody>
</table>
### PowerScript Datatypes vs. C# Datatypes

<table>
<thead>
<tr>
<th>PowerScript Datatype</th>
<th>C# Datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>long, ulong</td>
<td>int, uint</td>
</tr>
<tr>
<td>longlong</td>
<td>long</td>
</tr>
<tr>
<td>decimal</td>
<td>decimal</td>
</tr>
<tr>
<td>real</td>
<td>float</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>string</td>
<td>string</td>
</tr>
<tr>
<td>user-defined structure</td>
<td>struct</td>
</tr>
<tr>
<td>user-defined nonvisual object</td>
<td>class</td>
</tr>
<tr>
<td>Date</td>
<td>DateTime</td>
</tr>
<tr>
<td>Time</td>
<td>DateTime</td>
</tr>
<tr>
<td>DateTime</td>
<td>DateTime</td>
</tr>
</tbody>
</table>

**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 10. 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:

<table>
<thead>
<tr>
<th>Wizard field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project name</td>
<td>Name of the project object the wizard creates.</td>
</tr>
<tr>
<td>Library</td>
<td>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.</td>
</tr>
<tr>
<td>Target</td>
<td>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.</td>
</tr>
<tr>
<td>Library search path</td>
<td>Lets you add PBLs and PBDs to the search path for the new target.</td>
</tr>
<tr>
<td>PowerBuilder object name</td>
<td>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.</td>
</tr>
<tr>
<td>Description</td>
<td>Lets you add a description for the project object the wizard creates.</td>
</tr>
<tr>
<td>Web service virtual directory name</td>
<td>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.</td>
</tr>
<tr>
<td>Web service URL preview</td>
<td>Address for accessing the .NET Web service from an application.</td>
</tr>
<tr>
<td>Wizard field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Resource file and directory list</td>
<td>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.</td>
</tr>
<tr>
<td>Win32 dynamic library file list</td>
<td>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.</td>
</tr>
<tr>
<td>Generate setup file</td>
<td>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.</td>
</tr>
<tr>
<td>Setup file name</td>
<td>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.</td>
</tr>
<tr>
<td>Directly deploy to IIS</td>
<td>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”.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Wizard field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choose a target</td>
<td>Select a target from the list of targets in the current workspace.</td>
</tr>
<tr>
<td>Specify a project name</td>
<td>Select a name for the project you want to create. You must create a project object to deploy nonvisual objects as .NET components.</td>
</tr>
<tr>
<td>Choose a project library</td>
<td>Specify a library from the list of target libraries where you want to store the new project object.</td>
</tr>
<tr>
<td>Choose NVO objects to be deployed</td>
<td>Expand the library node or nodes in the list box and select check boxes next to the nonvisual objects that you want to deploy.</td>
</tr>
<tr>
<td>Use .NET nullable types</td>
<td>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.</td>
</tr>
<tr>
<td>Only include functions with supported datatypes</td>
<td>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.</td>
</tr>
<tr>
<td>Only include functions with supported datatypes</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug or Release</td>
<td>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.</td>
</tr>
<tr>
<td>Enable DEBUG symbol</td>
<td>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.</td>
</tr>
</tbody>
</table>

- **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 55.

- **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:

<table>
<thead>
<tr>
<th>Objects tab field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom class</td>
<td>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.</td>
</tr>
<tr>
<td>Object name</td>
<td>You can change the object name only by selecting a different object in the Custom Class treeview.</td>
</tr>
<tr>
<td>Web service name</td>
<td>Specifies the name for the Web service. By default, this takes the name of the current custom class user object.</td>
</tr>
<tr>
<td>Target namespace</td>
<td>Specifies the target namespace. The default namespace for an IIS Web service is: <a href="http://tempurl.org">http://tempurl.org</a>. Typically you change this to a company domain name.</td>
</tr>
<tr>
<td>Web service URL</td>
<td>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.</td>
</tr>
<tr>
<td>Web service WSDL</td>
<td>Specifies the WSDL file created for the project. This is a read-only field. It appends the “&quot;?WSDL” suffix to the Web service URL.</td>
</tr>
<tr>
<td>Objects tab field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Browse Web Service</td>
<td>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.</td>
</tr>
<tr>
<td>View WSDL</td>
<td>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.</td>
</tr>
<tr>
<td>Message names and Function prototypes</td>
<td>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.</td>
</tr>
<tr>
<td>Change message name</td>
<td>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.</td>
</tr>
<tr>
<td>Select All and Unselect All</td>
<td>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.</td>
</tr>
</tbody>
</table>

- 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 62. 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:
<table>
<thead>
<tr>
<th>Version tab field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product name, Company, Description, and Copyright</td>
<td>Use these fields to specify identification, description, and copyright information that you want to associate with the assembly you generate for the project.</td>
</tr>
<tr>
<td>Product version, File version, and Assembly</td>
<td>Enter major, minor, build, and revision version numbers for the product, file, and assembly.</td>
</tr>
</tbody>
</table>

- **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 121.

- **Run tab** — the fields on this tab cannot be set in the target or project wizards:

<table>
<thead>
<tr>
<th>Run tab field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>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.</td>
</tr>
<tr>
<td>Argument</td>
<td>Use this text box to enter any parameters for an application that invokes the classes and methods of the deployed project.</td>
</tr>
<tr>
<td>Start In</td>
<td>Use this text box to enter the starting directory for an application that invokes the classes and methods of the deployed project.</td>
</tr>
</tbody>
</table>

- **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 5.
Configuring ASP.NET for a .NET Web Service Project

Configure .NET Web Service 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.

For information on installing IIS and setting the default version of ASP.NET, see ASP.NET Configuration for a .NET Project on page 6.

SQL Anywhere database connections
Set up a database connection for your Web service components in the same way as for a smart client application. See Setting Up a SQL Anywhere Database Connection on page 8.

Global properties
The following global properties can be used by Web service projects:

- LogFolder
- FileFolder
- PrintFolder
- PBWebFileProcessMode
- PBCurrentDir
- PBTempDir
- PBLibDir
- PBDenyDownloadFolders
- PBTrace
- PBTraceTarget
- PBTraceFileName
- PBMaxSession
- PBEventLogID
- PBDeleteTempFileInterval

See and Global Web Configuration Properties on page 68.

Global Web Configuration Properties
A set of global properties is available for .NET Web Services.

Global properties are set in `web.config`, which is deployed to the `...\wwwroot\application_name` folder by the .NET Web Service project. You cannot set global properties in script.

This table lists global properties that you can set for .NET Web Service targets:
<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogFolder</td>
<td>WebAppDir.. \appName_root \log</td>
<td>Folder that contains the PBTrace.log file.</td>
</tr>
<tr>
<td>FileFolder</td>
<td>WebAppDir.. \appName_root \file</td>
<td>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.</td>
</tr>
<tr>
<td>PrintFolder</td>
<td>WebAppDir.. \appName_root \print</td>
<td>Base directory for files that your application prints in PDF format.</td>
</tr>
<tr>
<td>PBWebFileProcessMode</td>
<td>Share</td>
<td>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.</td>
</tr>
<tr>
<td>PBCurrentDir</td>
<td>c:\</td>
<td>Specifies the current directory for the Web Service.</td>
</tr>
<tr>
<td>PBTempDir</td>
<td>c:\temp</td>
<td>A temporary directory under the virtual file root on the server.</td>
</tr>
<tr>
<td>PBLibDir</td>
<td>c:~pl_</td>
<td>The directory on the server where dynamic libraries are generated.</td>
</tr>
<tr>
<td>PBDenyDownloadFolders</td>
<td>c:~pl_</td>
<td>A semicolon-delimited string of directory names.</td>
</tr>
<tr>
<td>PBTrace</td>
<td>Enabled</td>
<td>Indicates whether to log exceptions thrown by the application. Values are Enabled or Disabled.</td>
</tr>
<tr>
<td>PBTraceTarget</td>
<td>File</td>
<td>Defines where to log exceptions thrown by the application. Values are File or EventLog.</td>
</tr>
</tbody>
</table>
### Property Default value Description

<table>
<thead>
<tr>
<th>Property</th>
<th>Default value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBTraceFileName</td>
<td>PBTrace.log</td>
<td>Name of the file that logs exceptions thrown by the application. By default, this file is saved to the <code>applicationName_root \Log</code> directory under the virtual root directory on the server.</td>
</tr>
<tr>
<td>PBEventLogID</td>
<td>1100</td>
<td>The event ID if exceptions are logged to the EventLog.</td>
</tr>
<tr>
<td>PBDeleteTempFileInterval</td>
<td>600 (minutes)</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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 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.

**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 8.

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 10. 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 Web Service Deployment Considerations
This topic discusses requirements, restrictions, and options for deploying .NET Web Service projects.

When a .NET Web Service 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 Service 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 Service version of the application. Most of these warnings are benign and do not prevent users from running the application as a Web Service.

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 Service 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\webservice\applicationName`, and the extraction wizard creates the `C:\Program Files\webservice\applicationName` and `C:\Program Files\webservice\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 Service application to a production server either by:

- Extracting an MSI file that you build from a Web Service 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 6. 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 Service 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 Service 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 Service 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.

**ASP .NET user permissions**

If you copy files to a production server, or extract your Web Service 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 8. 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:

<table>
<thead>
<tr>
<th>IIS version</th>
<th>Commands for granting appropriate user permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td><code>cacls applicationName\temp /t /e /c /g ASPNET:f</code>&lt;br&gt;<code>cacls applicationName_root /t /e /c /g ASPNET:f</code></td>
</tr>
<tr>
<td>6</td>
<td><code>cacls applicationName\temp /t /e /c /g IIS_WPG:f</code>&lt;br&gt;<code>cacls applicationName_root /t /e /c /g IIS_WPG:f</code></td>
</tr>
<tr>
<td>7 and 7.5</td>
<td><code>cacls applicationName\temp /t /e /c /g IIS_IUSRS:f</code>&lt;br&gt;<code>cacls applicationName_root /t /e /c /g IIS_IUSRS:f</code></td>
</tr>
</tbody>
</table>

**Event logging on the production server**

If you log Web Service 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 Service 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 68.

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

<table>
<thead>
<tr>
<th>Preprocessing symbols</th>
<th>Project type</th>
<th>Code in this processing block</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBNATIVE</td>
<td>PowerBuilder client-server or distributed applications</td>
<td>Fully parsed by the PowerScript compiler. It is converted to blank lines for the <code>pb2cs</code> compiler.</td>
</tr>
<tr>
<td>PBWINFORM</td>
<td>.NET Windows Forms applications</td>
<td>Fully parsed by the <code>pb2cs</code> compiler for .NET Windows Forms targets only. It is converted to blank lines for the PowerScript compiler and all other types of .NET targets.</td>
</tr>
<tr>
<td>PBWEBSERVICE</td>
<td>.NET Web Service component targets</td>
<td>Fully parsed by the <code>pb2cs</code> compiler for .NET Web Service targets only. It is converted to blank lines for the PowerScript compiler and all other types of .NET targets.</td>
</tr>
<tr>
<td>PBDOTNET</td>
<td>Windows Forms applications and .NET Assembly and .NET Web Service components</td>
<td>Fully parsed by the <code>pb2cs</code> compiler for all .NET target types. It is converted to blank lines for the PowerScript compiler.</td>
</tr>
<tr>
<td>DEBUG</td>
<td>Standard PowerBuilder targets and all .NET application and component targets</td>
<td>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 <code>pb2cs</code> compiler. Code is converted to blank lines when the check box is cleared.</td>
</tr>
</tbody>
</table>

**Note:** The PBWPF preprocessor 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 preprocessor. 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:

```plaintext
#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:

```plaintext
#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>
<thead>
<tr>
<th>Error message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Invalid if statement</td>
<td><code>#if</code> statement without a defined symbol, with an incorrectly defined symbol, or without a <code>then</code> clause</td>
</tr>
<tr>
<td><code>#end if</code> directive expected</td>
<td><code>#if</code> statement without an <code>#end if</code> statement</td>
</tr>
<tr>
<td>Unexpected preprocessor directive</td>
<td>Caused by an <code>#else</code>, <code>#elseif</code>, or <code>#end if</code> statement when not preceded by an <code>#if</code> statement</td>
</tr>
<tr>
<td>Preprocessor syntax error</td>
<td>Caused by including text after an <code>#else</code> or <code>#end if</code> statement when the text is not preceded by comment characters (<code>//</code>)</td>
</tr>
</tbody>
</table>

**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
```

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: 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:

```powerscript
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:

```powerscript
#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:

```powerbuilder
#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:

```powerbuilder
#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 85.

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 15.
**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>
<thead>
<tr>
<th>.NET datatype</th>
<th>C# datatype</th>
<th>PowerBuilder datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>System.Boolean</td>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>System.Byte</td>
<td>Byte</td>
<td>Byte</td>
</tr>
<tr>
<td>System.Sbyte</td>
<td>Sbyte</td>
<td>Sbyte</td>
</tr>
<tr>
<td>System.Int16</td>
<td>short</td>
<td>Int</td>
</tr>
<tr>
<td>System.UInt16</td>
<td>ushort</td>
<td>Uint</td>
</tr>
<tr>
<td>System.Int32</td>
<td>int</td>
<td>Long</td>
</tr>
<tr>
<td>System.UInt32</td>
<td>uint</td>
<td>Ulong</td>
</tr>
<tr>
<td>System.Int64</td>
<td>long</td>
<td>Longlong</td>
</tr>
<tr>
<td>System.UInt64</td>
<td>ulong</td>
<td>Unsignedlonglong</td>
</tr>
<tr>
<td>System.Single</td>
<td>float</td>
<td>Real</td>
</tr>
<tr>
<td>System.Double</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>System.Decimal</td>
<td>Decimal</td>
<td>Decimal</td>
</tr>
<tr>
<td>System.Char</td>
<td>Char</td>
<td>Char</td>
</tr>
<tr>
<td>System.String</td>
<td>String</td>
<td>String</td>
</tr>
<tr>
<td>System.DateTime</td>
<td>System.Datetime</td>
<td>Datetime</td>
</tr>
</tbody>
</table>

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:

```csharp
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`):

```powerbuilder
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 81, 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>
<thead>
<tr>
<th>PowerBuilder datatype</th>
<th>C# datatype</th>
<th>.NET datatype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>Byte [ ]</td>
<td>System.Byte [ ]</td>
</tr>
<tr>
<td>Date</td>
<td>System.Datetime</td>
<td>System.Datetime</td>
</tr>
<tr>
<td>Time</td>
<td>System.Datetime</td>
<td>System.Datetime</td>
</tr>
</tbody>
</table>

**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 84.
• 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
#endif
```

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
#endif
```

• 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 85.

• 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 86.

• .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 87.

• 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:

```powerbuilder
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>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>XOR</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>NOT</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>
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:

```powerscript
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:

```powerscript
#if defined PBDOTNET THEN
    ns_1.ns_2.TimeOfDay a
    a=ns_1.ns_2.TimeOfDay.Morning!
#endif
```

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:

```powerscript
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 81.

**Note:** The `System.IntPtr` and `System.UIntPtr` 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:

```powershell
#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`:

```powershell
#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:

```powershell
#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*/
#endif

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*/
#endif

b = 1000/globala
globala = 0

a = 1000

globala = 10
b = a / globala

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

#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

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:

```csharp
namespace ExceptionSample
{
    // Custom exception class used in method second_test(int a) below
    public class MyCustomException : Exception
    {
        public string GetMessage()
        {
            return "Custom Error Thrown";
```
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:

```powerbuilder
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)
```
Example using a custom .NET exception class
Suppose the .NET Test class is modified to catch a custom .NET exception:

```csharp
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:

```csharp
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
#endif
```
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 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>Only Jaguar and AppServer are supported values. Any other value results in a runtime error.</td>
</tr>
<tr>
<td>ErrorCode</td>
<td>These error codes are supported:</td>
</tr>
<tr>
<td></td>
<td>• 0 — success</td>
</tr>
<tr>
<td></td>
<td>• 50 — distributed service error</td>
</tr>
<tr>
<td></td>
<td>• 57 — not connected</td>
</tr>
<tr>
<td></td>
<td>• 92 — required property missing or invalid</td>
</tr>
<tr>
<td></td>
<td>• 100 — unknown error</td>
</tr>
<tr>
<td></td>
<td>The same error codes are returned by the ConnectToServer function.</td>
</tr>
<tr>
<td>Options</td>
<td>These options support SSL connections from .NET clients. They are case sensitive and are not available for standard PowerBuilder (Win 32) clients:</td>
</tr>
<tr>
<td></td>
<td>• ORBclientCertificateFile</td>
</tr>
<tr>
<td></td>
<td>• ORBclientCertificatePassword</td>
</tr>
<tr>
<td></td>
<td>See SSL Connection Support on page 98.</td>
</tr>
</tbody>
</table>

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 98.

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:

<table>
<thead>
<tr>
<th>CORBA IDL type</th>
<th>Mode</th>
<th>PowerScript type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>in, return</td>
<td>Boolean by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Boolean by reference</td>
</tr>
<tr>
<td>char</td>
<td>in, return</td>
<td>Char by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Char by reference</td>
</tr>
<tr>
<td>octet</td>
<td>in, return</td>
<td>Byte by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Byte by reference</td>
</tr>
<tr>
<td>short</td>
<td>in, return</td>
<td>Integer by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Integer by reference</td>
</tr>
<tr>
<td>long</td>
<td>in, return</td>
<td>Long by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Long by reference</td>
</tr>
<tr>
<td>long long</td>
<td>in, return</td>
<td>Longlong by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Longlong by reference</td>
</tr>
<tr>
<td>float</td>
<td>in, return</td>
<td>Real by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Real by reference</td>
</tr>
<tr>
<td>double</td>
<td>in, return</td>
<td>Double by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Double by reference</td>
</tr>
<tr>
<td>string</td>
<td>in, return</td>
<td>String by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>String by reference</td>
</tr>
<tr>
<td>BCD::Binary</td>
<td>in, return</td>
<td>Blob by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Blob by reference</td>
</tr>
<tr>
<td>BCD::Decimal</td>
<td>in, return</td>
<td>Decimal by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Decimal by reference</td>
</tr>
<tr>
<td>BCD::Money</td>
<td>in, return</td>
<td>Decimal by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Decimal by reference</td>
</tr>
</tbody>
</table>
### .NET Language Interoperability

<table>
<thead>
<tr>
<th>CORBA IDL type</th>
<th>Mode</th>
<th>PowerScript type</th>
</tr>
</thead>
<tbody>
<tr>
<td>MJD::Date</td>
<td>in, return</td>
<td>Date by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Date by reference</td>
</tr>
<tr>
<td>MJD::Time</td>
<td>in, return</td>
<td>Time by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Time by reference</td>
</tr>
<tr>
<td>MJD::Timestamp</td>
<td>in, return</td>
<td>DateTime by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>DateTime by reference</td>
</tr>
<tr>
<td>TabularResults::ResultSet</td>
<td>in, return</td>
<td>ResultSet by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>ResultSet by reference</td>
</tr>
<tr>
<td>TabularResults::ResultSets</td>
<td>in, return</td>
<td>ResultSets by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>ResultSets by reference</td>
</tr>
<tr>
<td>Void</td>
<td>return</td>
<td>(None)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>CORBA IDL type</th>
<th>Mode</th>
<th>PowerScript type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>in</td>
<td>Bounded array by value</td>
</tr>
<tr>
<td></td>
<td>inout</td>
<td>Bounded array by reference</td>
</tr>
<tr>
<td>Sequence</td>
<td>in</td>
<td>Unbounded array by value</td>
</tr>
<tr>
<td></td>
<td>inout</td>
<td>Unbounded array by reference</td>
</tr>
<tr>
<td>Structure</td>
<td>in, return</td>
<td>Structure by value</td>
</tr>
<tr>
<td></td>
<td>out, inout</td>
<td>Structure by reference</td>
</tr>
</tbody>
</table>

### 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:

```power-script
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.

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

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 99.

**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:

```powershell
Connection myconnect
int rc
myconnect = create Connection
myconnect.Application = "pbtest"
myconnect.Driver = "jaguar"
myconnect.UserID = "admin@system"
```

---

.NET Language Interoperability

---

PowerBuilder
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 services**

For mutual authentication, PowerBuilder .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 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:

```powerbuilder
if b = 0 then
   label: ...
else
   ...
end if
goto label
```

This PowerScript translates conceptually into the following C# code:

```csharp
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:

```powerbuilder
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:

```csharp
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 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 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 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:

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

it is recommended that you alter it like this:

```c
#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 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 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
- 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.

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 — use only when absolutely necessary. Response windows and message boxes require more server-side resources than other kinds of windows.
- `Yield` — avoid whenever possible, because it requires additional server-side resources.
- Timers — use sparingly and avoid including them on forms that require data entry. Timers periodically generate postbacks and can impede 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.

Global properties also allow you to share data across application sessions.

These properties are listed in *Global Web Configuration Properties* on page 68.
Compiling, Debugging, and Troubleshooting

This part provides information about compiling, debugging, and troubleshooting .NET targets.

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 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 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, 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 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 redeploy its 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

<table>
<thead>
<tr>
<th>Selection</th>
<th>Effect when you click or select Run or Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ask me</td>
<td>(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.</td>
</tr>
<tr>
<td>Always redeploy</td>
<td>Always redeploy a project before running or debugging it. It first rebuilds the project using the rebuild scope set in the Project painter.</td>
</tr>
<tr>
<td>Never redeploy</td>
<td>Never redeploy 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.)</td>
</tr>
</tbody>
</table>

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 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:
• 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.

• Remote debugging — debugging of 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:

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

**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 75.

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:

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 59 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 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 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 81 for a list of datatype mappings from .NET to PowerBuilder.

If your application uses conditional compilation blocks, see Limitations on page 89 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 103 and Design-Level Considerations on page 106.

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 .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 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 18, 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:

```xml
<appSettings>
  <!-- The value could be "enabled" or "disabled"-->
  <add key = "PBTrace" value = "enabled"/>
  <!-- The target can be File, EventLog or File|EventLog -->
  <add key = "PBTraceTarget" value = "File"/>
  <!-- If the Target is File, PBTraceFileName should also be specified.-->
  <add key = "PBTraceFileName" value = "PBTrace.log"/>
</appSettings>
```
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 29.

- 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
Compiling, Debugging, and Troubleshooting

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

### Custom Permission Settings

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


### 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:

   ```xml
   <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 `GetContextKeywords` function. If you modify the setting to “Write” or “NoAccess”, `GetContextKeywords` will fail.

**Table 16. EnvironmentPermission required in Windows Forms**

<table>
<thead>
<tr>
<th>System function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetContextKeywords</td>
<td>Read</td>
</tr>
</tbody>
</table>

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

```xml
<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.

Set this property on the Configuration tab of the Project painter for Web Service targets after deployment in the `appname.exe.config` file (where `appname` is the name of the executable file).

**Table 17. EventLogPermission required in .NET targets**

<table>
<thead>
<tr>
<th>Global .NET property</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>PBTraceTarget when the value is set to file</td>
<td>Windows Forms: Pbtrace.log FileIO permission asserted by default in runtime library</td>
</tr>
<tr>
<td>PBTraceTarget when the value is set to EventLog</td>
<td>Administer</td>
</tr>
</tbody>
</table>

In Windows 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:

```xml
<IPermission class=" EventLogPermission" version="1">
```
FileDialogPermission

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

<table>
<thead>
<tr>
<th>System function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetFileOpenName</td>
<td>Open or OpenSave (unrestricted FileIOPermission also required)</td>
</tr>
<tr>
<td>GetFileSaveName</td>
<td>Save or OpenSave (unrestricted FileIOPermission also required)</td>
</tr>
</tbody>
</table>

FileIOPermission

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

**Permission requirements for Windows Forms**

<table>
<thead>
<tr>
<th>System function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileWrite, FileWriteEx, RemoveDirectory, LibraryCreate, LibraryDelete, and LibraryImport</td>
<td>Write</td>
</tr>
<tr>
<td>FileDelete</td>
<td>Read and Write</td>
</tr>
<tr>
<td>System function</td>
<td>Permission required</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------</td>
</tr>
</tbody>
</table>
| 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.

<table>
<thead>
<tr>
<th>Object or control</th>
<th>Function or property</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animation</td>
<td>AnimationName</td>
<td>Read</td>
</tr>
<tr>
<td>DataWindow</td>
<td>SaveAsAscii, SaveAsFormattedText, SaveInk, SaveInkPicture</td>
<td>Write</td>
</tr>
<tr>
<td>ImportFile</td>
<td></td>
<td>Read permission if the (usually second) file name argument is supplied; if file name argument is empty or null, requires OpenFileDialogPermission and Unrestricted FileIOPermission</td>
</tr>
<tr>
<td>Object or control</td>
<td>Function or property</td>
<td>Permission required</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td></td>
<td>SaveAs</td>
<td>Write permission if the (usually second) file name argument is supplied; if file name argument is empty or null, requires OpenFileDialogPermission and Unrestricted FileIOPermission</td>
</tr>
<tr>
<td>DataWindow (RichText only)</td>
<td>InsertDocument</td>
<td>Read</td>
</tr>
<tr>
<td>DataStore</td>
<td>ImportFile</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>SaveAs, SaveAsAscii, SaveAsFormattedText, SavInk, SaveInkPicture</td>
<td>Write</td>
</tr>
<tr>
<td>DragObject</td>
<td>DragIcon property</td>
<td>Read</td>
</tr>
<tr>
<td>DropDownListBox, ListBox</td>
<td>DirList (string s, uint filetype)</td>
<td>Read and PathDiscovery for the file specification (first) argument</td>
</tr>
<tr>
<td></td>
<td>AddPicture, and PictureName property</td>
<td>Read</td>
</tr>
<tr>
<td>DropDownPictureListBox, PictureListBox</td>
<td>DirList (string s, uint filetype)</td>
<td>Read and PathDiscovery for the file specification (first) argument</td>
</tr>
<tr>
<td>Graph</td>
<td>ImportFile</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>SaveAs</td>
<td>Unrestricted for function with no arguments; Write on the file name (first) argument for function with arguments</td>
</tr>
<tr>
<td>InkPicture</td>
<td>PictureFileName property</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>LoadInk</td>
<td>Read and Write for the file (first) argument</td>
</tr>
<tr>
<td></td>
<td>LoadPicture</td>
<td>Read for the file (first) argument</td>
</tr>
</tbody>
</table>
### 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**

<table>
<thead>
<tr>
<th>System function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Print, PrintBitmap, PrintCancel, PrintClose, PrintDataWindow, PrintDefineFont, PrintGetPrinter, PrintScreen, PrintSend, PrintSetFont, PrintSetSpacing, PrintLine, PrintOpen, PrintOval, PrintPage, PrintRect, PrintRoundRect, PrintSetupPrinter, PrintText, PrintWidth, PrintX, PrintY</td>
<td>DefaultPrinting or AllPrinting</td>
</tr>
<tr>
<td>System function</td>
<td>Permission required</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>PrintGetPrinters, PrintSetPrinter, PrintSetup</td>
<td>AllPrinting</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Object or control</th>
<th>Function or property</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataWindow</td>
<td>Print with no arguments</td>
<td>DefaultPrinting or AllPrinting</td>
</tr>
<tr>
<td></td>
<td>Print (cancelDialog, true)</td>
<td>AllPrinting</td>
</tr>
<tr>
<td>DataStore</td>
<td>Print</td>
<td>DefaultPrinting or AllPrinting</td>
</tr>
<tr>
<td>DragObject</td>
<td>Print</td>
<td>DefaultPrinting or AllPrinting</td>
</tr>
<tr>
<td>RichTextEdit</td>
<td>PrintEx (cancelDialog)</td>
<td>DefaultPrinting or AllPrinting</td>
</tr>
<tr>
<td>Window</td>
<td>Print</td>
<td>DefaultPrinting or AllPrinting</td>
</tr>
</tbody>
</table>

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

```xml
<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

<table>
<thead>
<tr>
<th>System function or object</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindClassDefinition, FindTypeDefinition</td>
<td>TypeInfo</td>
</tr>
<tr>
<td>ScriptDefinition object</td>
<td>TypeInfo</td>
</tr>
</tbody>
</table>

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

```xml
<IPermission class="System.Security.Permissions.ReflectionPermission, mscorlib, Version=4.0.0.0, Culture=neutral,
```
RegistryPermission

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

Table 23. Required RegistryPermission settings for system functions

<table>
<thead>
<tr>
<th>System function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryGet, RegistryKeys, Registry-Values</td>
<td>Read</td>
</tr>
<tr>
<td>RegistrySet</td>
<td>Write; if registry key does not exist, requires Create</td>
</tr>
<tr>
<td>RegistryDelete</td>
<td>Read and Write</td>
</tr>
</tbody>
</table>

This table shows the required RegistryPermission settings for MLSync object functions in .NET targets:

Table 24. Required RegistryPermission settings for MLSync functions

<table>
<thead>
<tr>
<th>MLSync function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetObjectRevisionFromRegistry, GetsSyncRegistryProperties</td>
<td>Read on HKEY_CURRENT_USER registry key</td>
</tr>
<tr>
<td>GetDBMLSyncPath</td>
<td>Read on the Software\Sybase\SQL Anywhere registry keys under HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE</td>
</tr>
<tr>
<td>SetsSyncRegistryProperties</td>
<td>Unrestricted on HKEY_CURRENT_USER registry key</td>
</tr>
</tbody>
</table>

This example for a Windows Forms application grants read permission for the HKEY_CURRENT_USER registry key, which extends to its subkeys:

```xml
<IPermission class="System.Security.Permissions.RegistryPermission, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" 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

<table>
<thead>
<tr>
<th>Function, object, property, or feature</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLEControl</td>
<td>Unrestricted (or the Full Trust option)</td>
</tr>
<tr>
<td>ChangeDirectory, Handle, Post, Restart, Run, Send</td>
<td>UnmanagedCode</td>
</tr>
<tr>
<td>URL (PictureHyperlink and StaticHyperlink property),</td>
<td>UnmanagedCode</td>
</tr>
<tr>
<td>HyperlinkToURL (Inet property)</td>
<td>UnmanagedCode</td>
</tr>
<tr>
<td>Language interoperation feature</td>
<td>Variable permissions required, depending on .NET function called or property accessed</td>
</tr>
<tr>
<td>Win32 API feature</td>
<td>UnmanagedCode</td>
</tr>
</tbody>
</table>

This example sets required security permissions for Windows Forms targets:

```xml
<IPermission class="System.Security.Permissions.SecurityPermission, mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1" Flags="Assertion, Execution, BindingRedirects, UnmanagedCode" />```

SMTPPermission

An SMTPPermission setting is required for the MailSession object log on function in .NET targets.

Table 26. SMTPPermission required in .NET targets

<table>
<thead>
<tr>
<th>MailSession object function</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>MailLogon</td>
<td>Connect (if using default port) or ConnectToUnrestrictedPort</td>
</tr>
</tbody>
</table>

This permission setting allows a Windows Forms application to log onto a mail session and receive mail through a default port:

```xml
<IPermission class="System.Net.Mail.SmtpPermission, System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" version="1" Access="Connect"/>
```
SocketPermission

A SocketPermission setting is required for the Connection object `ConnectToServer` function in .NET targets.


| Table 27. SocketPermission required in .NET targets |
|----------------|----------------|
| **Connection object function** | **Permission required** |
| ConnectToServer | Connect |

This permission setting allows a Windows Forms application to get or set a network access method:

```xml
<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>
```

SQLClientPermission

A SocketPermission setting is required for the database connection feature in .NET targets.

| Table 28. 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:

```xml
<IPermission class="System.Data.SqlClient.SqlClientPermission, System.Data, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" 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.
**WebPermission**

WebPermission settings are required for features and functions in .NET targets.


<table>
<thead>
<tr>
<th>Function or feature</th>
<th>Permission required</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetURL (Inet function)</td>
<td>Connect for <em>urlname</em> argument</td>
</tr>
<tr>
<td>PostURL (Inet function)</td>
<td>Connect for <em>urlname</em> and <em>serverport</em> arguments</td>
</tr>
<tr>
<td>Web Service call feature</td>
<td>Unrestricted=&quot;true&quot;</td>
</tr>
</tbody>
</table>

This permission setting allows a Windows Forms application to connect to the Sybase Web site:

```xml
<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>
```

**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

.NET applications
  bitwise operator support 84
coding restrictions 103
.NET assemblies
  importing 18, 80
  strong names 14
.NET Assembly
  component project 57
  component target wizard 55
.NET calls
  PowerScript syntax for 78
.NET classes and interfaces
  limitations in conditional code 89
.NET compiler 16
.NET enumerations 85
.NET environment
  debugging 113, 116
  handling exceptions 90
  support for language features 82
  synchronization 87
.NET Framework SDK 18
.NET generics classes
  unsupported 89
.NET language features
  support for 82
.NET modules 110
.NET primitive types 86
.NET Web Service
  component project 64
  component target wizard 62
.NET Windows Forms
  See Windows Forms applications

A
access permissions
  ASP.NET 8
application
  design-level considerations 106
directory structure 32
  installing 27, 31
  manifest file for smart client 29
  publishing 26
  running mode 27, 31
  update mode 27, 33
  updating 32
arrays 79, 89
ASP.NET
  configuring 6
  setting user permissions 8
  user permissions 73
  version 7
assemblies
  See .NET assemblies
AutoScript
  does not support .NET classes 89

B
best practices 16
bitwise operator support
  .NET applications 84
bootstrapper
  about 35
  customizing 35
build directories 109
builds
  Debug and Release 114
  incremental 109
  PBD generation 110

C
case sensitivity 80
ClickOnce technology 26
company name
  setting 20, 29
compound statements 78
conditional compilation 75
configuring
  ASP.NET 6, 68
  SQL Anywhere database connection 8
Connection object
  connecting to EAServer 94
  using with a .NET client 94
controls
  supported in Windows Forms 44
CORBA
  supported datatypes 97
CORBACurrent object 96
Index

CORBAObject object 96

D
data
  synchronizing 37
data exchange 80
datatype mapping 60, 81
Debug builds 114
DEBUG preprocessor symbol 114, 117
debugging
  .NET applications 113
  .NET components 116
  attaching to a running process 113
  restrictions with .NET targets 113
  Windows Forms 113
declaring
  arrays 79
  enumerations 85
deploying
  .NET Assembly project 61
  .NET Web Service project 70
  Web Service projects 71
  Windows Forms projects 25
deployment
  checklist for production server 11
  troubleshooting 117
deployment manifest file for smart clients 29
digital certificates 28
directory structure, on server 32
DLLs
  deploying 11, 20
DYNAMIC keyword
  unsupported with .NET methods 89

E
EAServer
  .NET clients 94
  using the Connection object 94
  using the JaguarORB object 95
enumerated types
  function calls on 86
enumerations 85
exceptions
  handling in .NET environment 90

F
file server
  setting up 26
files
  runtime 11
fonts
  using TrueType in controls in Windows Forms 38
FTP server
  setting up 26
Full Trust required for smart client 29

G
GAC (Global Assembly Cache) 15
generic .NET classes
  unsupported 89
Global Assembly Cache (GAC) 15
global properties
  and .NET Web Service targets 68
  list of 68
  taking advantage of 108

H
handling exceptions
  in .NET environment 90

I
IIS
  installing 7
images
  deploying 23
  for Windows Forms targets 32
incremental builds 109, 112
Indexes
  for .NET classes 87
  instantiating a .NET class 78
  intelligent notifier 34
  intelligent update 33
interoperability
  datatype mappings 81
  referencing .NET classes 75
  support for .NET language features 82
  writing code in a .NET block 77

J
JaguarORB object
  connecting to EAServer 95
K
keywords 79

L
library files 20
line return characters 79
log file
  pbiupub.log 119
  pbtrace.log 9

M
mandatory updates 34
manifest files
  for smart client application 29
  for smart client deployment 29
  for Windows Forms 24
  security tab 20
  signing with digital certificates 28
migration
  runtime files 11
MobiLink synchronization
  for smart clients 37
multithreading .NET applications
  support for 87
mutual authentication 102

N
notifier
  icon 34
  options 34
nullable
  unsupported 89

O
online only 31

P
PATH environment variable 18
PBDs
  deploying 20
PBLs
  deploying 20
PBTrace.log file 9, 118
permissions
  adding in .NET Framework configuration tool 121
  adding manually for copied files 73
  ASP.NET 6
  EnvironmentPermission 122
  error messages 4
  EventLogPermission 122
  FileDialogPermission 123
  FileIOPermission 123
  for Web service components 70
  Full Trust required for smart client 29
  granting from command line 73
  PrintingPermission 126
  ReflectionPermission 127
  RegistryPermission 128
  SecurityPermission 128
  SMTPPermission 129, 130
  Sybase directories 8
  troubleshooting Windows Forms 118
  UIPermission 130
  WebPermission 131
POST keyword
  unsupported with .NET methods 89
post-build commands 20
PowerBuilder runtime files
  deploying 20
PowerScript
  keywords 79
  unsupported events in Windows Forms 51
  unsupported functions in Windows Forms 49
  unsupported properties in Windows Forms 51
preprocessor statements
  pasting into script 78
preprocessor symbols
  about 75
  DEBUG 114
  list of 75
prerequisites
  for application 29, 35
  for deployment 10
  for development 18
primitive types
  function calls on 86
projects
  out-of-date message 111
properties
  global 68
publish page
  link to server 26
  prerequisites 35
  view of 29
publishing an application 26

R
rebuild scope 110
Release builds 114
requirements
  system 18
resource files
  for .NET assembly targets 59
  for .NET Web service targets 66
  for .NET Windows Forms targets 23
resources
  deploying 23
running an application 25
runtime files
  deploying 11

S
security 29
  manifest files for Windows Forms 24
  settings 3
server authentication 98
Sign tab 5
signing manifest files 28
smart client
  rolling back 37
SQL Anywhere
  setting up database connection 8
SSL connection support 98
Start menu
  adding to 31
string matching 107
strong-named assemblies 5, 14
structures
  supported 38
system functions
  unsupported in Windows Forms 44
system objects
  supported 38
system options
  redeployment 111
system requirements 18
System.Nullable
  unsupported 89
T
troubleshooting
  conditional code 89
  deployment errors 117
  tips for Windows Forms applications 118
TrueType fonts
  using in controls in Windows Forms 38
trust options 3
U
updates
  checking for 33, 34
  mandatory 34
  online and offline 33
  online only 33
  polling for 34
V
Vista
  See Windows Vista
visual controls
  supported in Windows Forms 44
W
Web browser
  default start page 25
Web server
  setting up 26
Windows Forms Application project 18
Windows Forms Application wizard 16
Windows Forms applications
  advantages of 1, 16
  supported controls 46–49
  supported objects 42, 43
Windows Vista
  additional requirements for Windows Forms 24