SYBASE®
An SAP® Company

Features Guide

PowerBuilder® .NET 12.5.2

DOCUMENT ID: DC01261-01-1252-01
LAST REVISED: February 2013

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

Contents

About PowerBuilder .NET

The Sybase® PowerBuilder® setup program allows you to install two separate IDEs. The familiar PowerBuilder IDE has been rebranded as PowerBuilder Classic. The new IDE is named PowerBuilder .NET.

You start each IDE from separate items on the Start menu, and you can run multiple sessions of each PowerBuilder IDE simultaneously. The `PB125.EXE` file opens the PowerBuilder Classic IDE, and the `PBSHELL.EXE` opens the PowerBuilder .NET IDE.

A set of wizards in PowerBuilder .NET enable you to quickly create a variety of targets for different types of applications. For example, one wizard creates a new Windows Presentation Foundation (WPF) application or, if you prefer, converts an existing PowerBuilder Classic client/server or .NET Windows Forms target. Other wizards allow you to create .NET Assembly and WCF Service targets from nonvisual user objects. The PB Assembly wizard creates assembly components that you can include in other projects within PowerBuilder .NET. After the wizard generates a target you can modify its properties at any time.

Project painters let you specify how PowerBuilder generates executable applications from your target. You can define multiple projects for the same target, each for a different deployment scenario.

Building applications in PowerBuilder .NET lets you take advantage of language enhancements for fuller .NET compliance than applications designed in PowerBuilder Classic.

PowerBuilder .NET Architecture

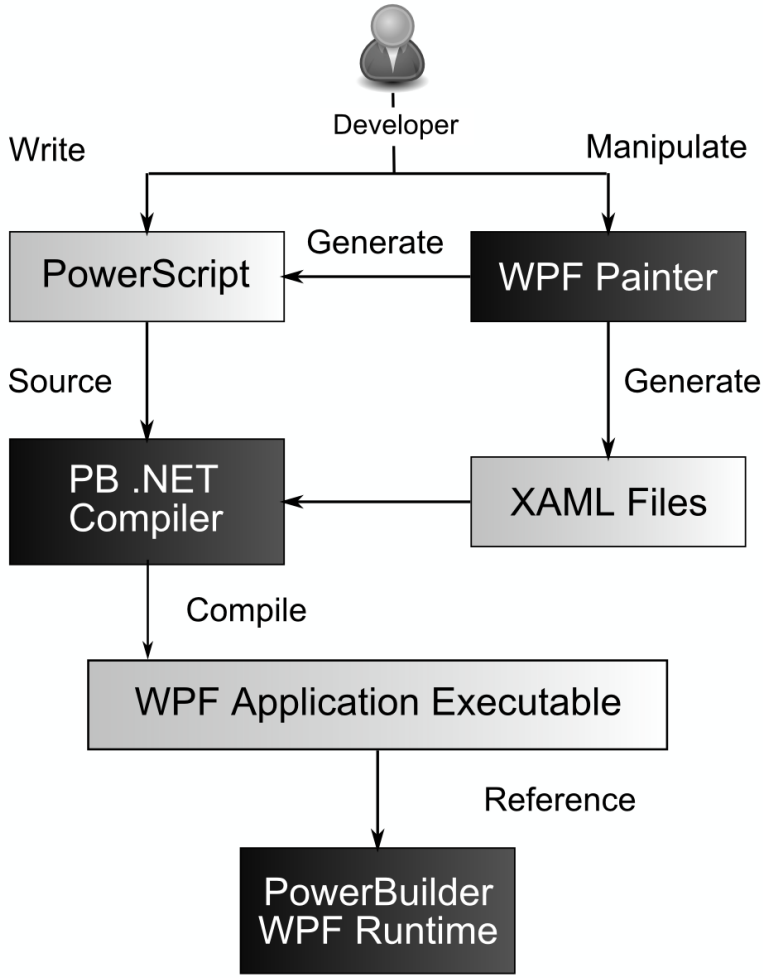
PowerBuilder .NET applications, including system classes and system functions, are built on top of the PowerBuilder WPF runtime library.

The WPF runtime library reuses the existing nonvisual part of the PowerBuilder Classic .NET runtime library, which has been enhanced for compliance with the extended Common Language Specification (CLS). The major difference between PowerBuilder .NET and PowerBuilder Classic is in the presentation layer.

A .NET assembly, `Sybase.PowerBuilder.WPF.controls.dll`, contains PowerBuilder versions of all WPF controls available in the presentation layer. (Implementations for the DataWindow®, EditMask, and RichText controls depend on additional assemblies.) WPF application development reuses the existing PowerBuilder to .NET compiler (**pb2cs**) for application compilation.

PowerBuilder Classic and PowerBuilder .NET share the same compiler, which has been modified to correctly generate WPF applications. For example, the WPF runtime library must link binary application markup language (BAML) to WPF controls. The modified version of the compiler accomplishes this linkage for PowerBuilder .NET applications, even though the .NET Windows Forms applications you develop in PowerBuilder Classic do not use BAML or WPF controls.

The following diagram shows the process used by PowerBuilder to create a new WPF application. Once you drag and drop a control or change something in the layout of the WPF Window painter, a corresponding XAML file is generated.



The WPF Window painter also generates PowerScript® code to work as the code-behind file to the XAML files. However, before it can be used as a code-behind file, the PowerScript code

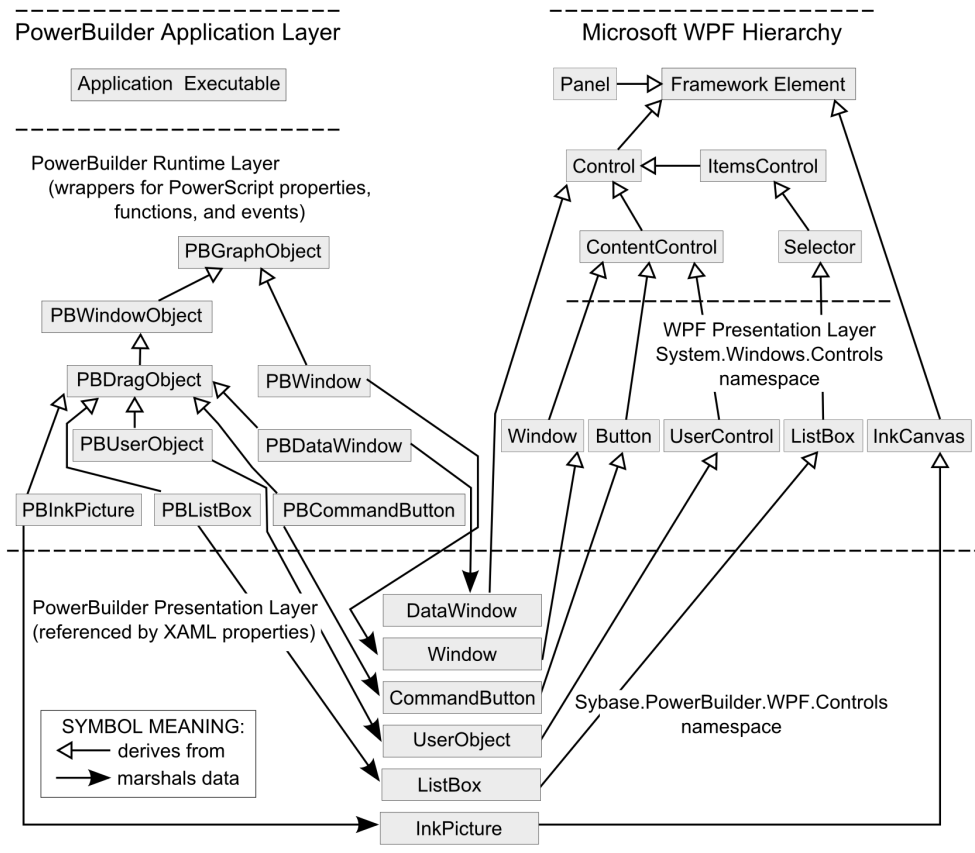
is translated and processed by the PowerBuilder to .NET compiler. The generated files are then compiled to a WPF application that contains embedded BAML. The WPF application also references the PowerBuilder WPF runtime engine that enables users to run the deployed application.

WPF Control Classes

The runtime classes in a PowerBuilder .NET application maintain the traditional PowerBuilder control hierarchy, but inherit from the Microsoft WPF control hierarchy.

When the runtime engine loads the application, the PowerBuilder .NET control classes (that are actually .NET wrappers around standard PowerScript controls) marshal data and connect to PowerBuilder versions of WPF controls in the presentation layer.

The following diagram shows the dual-dependency structure of controls in the PowerBuilder presentation layer. However, for simplicity, only a minimal set of control classes is shown in the diagram.



The PowerBuilder .NET controls internally use a "PB" prefix. This set of controls enables you to use PowerScript code to access control properties, methods, and events that marshal the data and pass it to the PowerBuilder version of WPF controls in the presentation layer.

The controls that you add to window objects in the PowerBuilder WPF Window painter, and that appear in the runtime presentation layer, are derived from control classes in the WPF Framework (System.Windows.Controls namespace). They are referenced at design time in XAML using the default **pbwpf** alias for the Sybase.PowerBuilder.WPF.Controls namespace, and at runtime, in binary application markup language (BAML).

Semantic Differences

There are a number of semantic differences between PowerBuilder .NET and PowerBuilder Classic.

The main semantic differences are listed below:

- A "solution" in PowerBuilder .NET is equivalent to a "workspace" in PowerBuilder Classic. A solution file is saved with a `PBWX` extension, rather than the `PBW` extension used in PowerBuilder Classic. The Solution Explorer in PowerBuilder .NET allows you to peruse all the objects in a solution, and is equivalent to the System Tree in PowerBuilder Classic.
- A "target" in PowerBuilder .NET is saved with a `PBTX` extension, rather than the `PBT` extension used in PowerBuilder Classic. A target file in both PowerBuilder IDEs is equivalent to a Visual Studio "project." The Find and Replace dialog box that PowerBuilder .NET uses from the Visual Studio isolated shell has a "current project" selection that actually applies to the current PowerBuilder target, not the current project. Project objects in PowerBuilder are contained inside targets.
- A "library" or "PBL" in PowerBuilder .NET is a folder containing the source files for all objects in a PowerBuilder library, while in PowerBuilder Classic, the PBL is a binary file containing the compiled code as well as the source code for all objects in the library. The PBL directory in PowerBuilder .NET must have a `PBL` extension, and it must contain a `PBLX` file. Although you can add other files, including PowerBuilder source files, to the library directories using file management tools, the PowerBuilder .NET Solution Explorer recognizes only the files registered in the `PBLX`. You can register PowerBuilder source files either by creating them in the IDE or by importing them, as in PowerBuilder Classic.
- A "Window control" in WPF applications can host only a single control, while in PowerBuilder Classic, a window is a container control that can host multiple controls. By default, to work around this limitation, PowerBuilder .NET places a Grid container control in every WPF Window control that you create. However, PowerBuilder .NET also lets you replace the Grid control with other panel types, and you can change the default by making a different container control selection in the WPF Window wizard. The default panel type for windows that you migrate from PowerBuilder Classic is a Canvas container control. As a corollary to this syntactic difference, controls that are inside a GroupBox in PowerBuilder .NET belong to the GroupBox and not to the container of the GroupBox.

When you migrate a PowerBuilder Classic application with radio button controls inside a GroupBox, the migration wizard automatically attributes the radio buttons to the GroupBox in the migrated application. Therefore, if you make the GroupBox invisible at runtime, the radio buttons will also be invisible.

For resize events in applications that you migrate to PowerBuilder .NET, this difference in processing may require you to comment out any script that changes the position of the radio button controls inside a GroupBox. However, the migration wizard does not attribute other types of controls to a GroupBox, only radio buttons. The other types of controls are attributed to a panel control during migration.

Runtime Requirements for PowerBuilder .NET

The applications you create in PowerBuilder .NET automatically reference several .NET assemblies that must be present in the global assembly cache (GAC) on the design time and runtime computers.

The PowerBuilder setup program installs the required assemblies in the GAC on the design time computer, but you must also deploy or install them in the GAC on each runtime computer.

The Microsoft assemblies (and their .NET Framework version numbers) that are installed with PowerBuilder and can be redistributed to runtime computers with the PowerBuilder Runtime Packager include:

- System.dll (v 2.0)
- PresentationCore.dll (v 3.0)
- PresentationFramework.dll (v 3.0)
- WindowsBase.dll (v 3.0)
- System.Xml.Linq.dll (v 3.5)

The Runtime Packager also deploys these Sybase assemblies that are required on runtime computers:

- Sybase.PowerBuilder.WPF.dll
- Sybase.PowerBuilder.Common.dll
- Sybase.PowerBuilder.Interop.dll
- Sybase.PowerBuilder.Core.dll

Advantages of WPF Applications

WPF applications allow you to take advantage of the best attributes of diverse systems, such as DirectX (3-D and hardware acceleration), Adobe Flash (animation support), and HTML (declarative markup and easy deployment).

The advantages of WPF applications include:

- Tight multimedia integration – to use 3-D graphics, video, speech, and rich document viewing in Windows 32 or Windows Forms applications, you would need to learn several independent technologies and blend them together without much built-in support. WPF applications allow you to use all these features with a consistent programming model.
- Resolution independence – WPF lets you shrink or enlarge elements on the screen, independent of the screen's resolution. It uses vector graphics to make your applications resolution-independent.
- Hardware acceleration – WPF is built on top of Direct3D, which offloads work to graphics processing units (GPUs) instead of central processor units (CPUs). This provides WPF applications with the benefit of hardware acceleration, permitting smoother graphics and enhanced performance.
- Declarative programming – WPF uses Extensible Application Markup Language (XAML) declarative programming to define the layout of application objects and to represent 3-D models, among other things. This allows graphic designers to directly contribute to the look and feel of WPF applications.
- Rich composition and customization – WPF controls are easily customizable. You need not write any code to customize controls in very unique ways. WPF also lets you create skins for applications that have radically different looks.
- Easy deployment – WPF provides options for deploying traditional Windows applications (using Windows Installer or ClickOnce) . This feature is not unique to WPF, but is still an important component of the technology.
- Culturally aware controls – static text in controls and the return data for the String function are modified according to the culture and language specified by the end user's operating system.

WPF is also more suitable for applications with rich media content than Windows Forms applications. This includes applications using:

- Multimedia and animation with DirectX technology
- HD video playback
- XPS documentation for high quality printing
- Control rotation (Windows Forms applications support text rotation only)

Modified and Unsupported Features in PowerBuilder .NET

Some features in PowerBuilder Classic are not supported in PowerBuilder .NET. Other features are partially supported or use a different method to obtain similar results or to display similar functionality.

Several differences are the result of .NET restrictions, or the Visual Studio isolated shell designer and functionality that PowerBuilder .NET leverages. The following features are not currently supported or are only partially supported in PowerBuilder .NET:

- Target location – you can save workspaces and libraries to the root directories of computer drives, but do not save PowerBuilder .NET target files (PBTX) to root directories.
- Event sequence – the difference in event sequence in WPF applications may affect applications that are dependent on a strict order of triggered events. You must manually refactor migrated applications to take into account the order of triggered events. Events must be handled immediately after they fire.
- User-defined custom events (partial support) – PowerBuilder .NET uses a different event model than PowerBuilder Classic. For a list of supported custom events in PowerBuilder .NET, see *Supported Custom Events*.
- Fonts that are not TrueType or OpenType fonts – although you can only select installed TrueType or OpenType fonts in PowerBuilder .NET, you can still enter the names of other fonts for the FaceName property of controls that display text. You can also migrate applications that use non-TrueType fonts from PowerBuilder Classic. However, at runtime, the .NET Framework replaces fonts that are not TrueType or OpenType fonts, causing unexpected changes to the way the text appears to end users of PowerBuilder .NET applications.
- Coding restrictions – several PowerScript coding practices, such as dashes in identifiers or colons for microsecond separators in time functions, are not permitted in .NET environments. For more information, see *Coding Restrictions*.
- Section 508 support – uses UI Automation (UIA) to implement accessibility instead of the Microsoft Active Accessibility (MSAA) used by PowerBuilder Classic. UIA implements a UIA-to-MSAA bridge, so you can still use MSAA tools for application accessibility in PowerBuilder .NET. You can also use newer tools based on UIA for this purpose.
  UIA supports the AccessibleName property, but does not support the AccessibleRole and AccessibleDescription properties on windows and controls that inherit from DragObject.
- Window previews – do not work for untitled windows. You must save a window before you can preview it at design time.
- Control handles – in PowerBuilder Classic applications, each control has its own window handle, and many operations depend on the window handle. In PowerBuilder .NET, there is only a single "big" handle for a window, but the controls nested in a window do not have their own handles. You must refactor migrated applications that use API calls relying on control handles before importing them to PowerBuilder .NET.
- Design-time list views – the Control List, Non-Visual Object List, Function List, and Event List views are replaced in PowerBuilder .NET by the Document Outline and the PB Object Outline.
- Library painter and the (PowerScript) Browser – most of the functionality of the Library painter and the Browser is available from Solution Explorer context menu items and the PB Object Outline.
- External visual objects – are not supported in PowerBuilder .NET. You must remove these objects manually before you migrate applications from PowerBuilder Classic. Although the Migration wizard reports unsupported system features, it does not report user-defined features, such as external visual objects, that are unsupported.
- Resource files – are added directly to a target or a folder under the target. PowerBuilder .NET does not use PBR files, although if you convert a target from

PowerBuilder Classic, the new target wizard lets you select resources that are listed in a PBR file associated with the target you are converting.

- The Plug-in Manager – in the PowerBuilder Classic IDE is replaced in PowerBuilder .NET by the Add-in Manager from the Visual Studio isolated shell. To enable add-ins, see *Add-ins in the PowerBuilder .NET Environment*.
- Pipeline and query objects – cannot be created from the New dialog box. In PowerBuilder .NET, you can create query objects only in the SQL Dialog invoked from the DataWindow painter, although you can also import or migrate them from PowerBuilder Classic. Pipeline objects are not currently supported in PowerBuilder .NET, and if you import or migrate these objects, they are ignored at runtime.
- OLE controls – only partially supported in PowerBuilder .NET.
- PBNI (including PBDOM objects) – are not supported in PowerBuilder .NET.
- Profile- and trace-related objects – are not supported in PowerBuilder .NET.
- EAServer components – are not supported in PowerBuilder .NET.
- EMF and WMF image formats – these formats are not supported by WPF because they are more susceptible to security vulnerabilities than other image formats.
- Syntax and semantic checking – occurs in the background, and you do not need to save or compile a script to detect any scripting errors. The Script Editor inserts a red caret under the errors that it detects, and you can also view the list of detected errors in the Error List window. In semantic checking, the parser uses type information to validate the script.
- Passing strings by reference – before passing a string to an external function by reference in PowerBuilder .NET, allocate memory for the string by calling the **Space** system function. This enables you to pass the same string to the function in subsequent calls to the function, even if the string becomes empty.
- Global variables and global external functions – can only be declared in the Application object of a WPF Window Application target.
- **Timer** system function – may require the WindowName parameter to trigger the Timer event, particularly if several windows are open. This parameter is optional in PowerBuilder Classic. Also, there is a slight timing interval difference (typically about 1 millisecond) due to the threading model in WPF applications.
- Debugger – is now based on the Visual Studio debugger. The *Debugger Changes* topic summarizes differences with the PowerBuilder Classic debugger.
- Source control – connections to source control use completely different mechanisms in PowerBuilder .NET and PowerBuilder Classic. For more information, see *Source Control in PowerBuilder .NET*.
- Resource objects – all DataWindow and query objects in WPF targets are compiled in a separate DLL (*appName*.resource.dll) rather than in the application executable.
- Look and feel differences – the size of WPF controls is determined by the controls' contents.

For a list of unsupported PowerScript properties, events, and functions in PowerBuilder .NET, see *Unsupported Properties, Events, and Functions*.

Behavior Changes for Runtime Controls

The .NET and WPF environments affect the behavior of several PowerBuilder controls at runtime.

This table highlights some of the important changes to the runtime behavior of PowerBuilder controls in WPF applications:

Control property or functionality	PowerBuilder Classic behavior	PowerBuilder .NET behavior
All controls focus sequence when current control has a TabOrder value of 0	Pressing Tab or the Shift+Tab keys changes focus to the control with the lowest TabOrder value (other than 0).	Pressing Tab changes focus to the control with the lowest TabOrder value (other than 0), but pressing Shift+Tab changes focus to the control with the highest TabOrder value.
DatePicker background and text colors	Properties are not available to set the colors in the text box part of the control. The CalendarBackColor and CalendarTextColor properties set the colors in the drop-down calendar part of the control.	BackColor and TextColor properties set the colors in the text box part of the control, and the CalendarBackColor and CalendarTextColor properties set the colors in the drop-down calendar part of the control.
DatePicker font size	You cannot save negative font sizes at design time, but positive font sizes are saved as negative values. If positive values are set at runtime, the actual text size for the text box part of the control is converted from PowerBuilder units to units for a logical device (such as a screen), and is relatively smaller than the actual font size that appears when you set the same values preceded by a minus sign. However, this calculation is not performed for the calendar part of the control, where text sizes appear as they would if the set values were preceded by a minus sign.	As in PowerBuilder Classic, you cannot save negative font sizes at design time, but positve font sizes can be saved as positive values if you set these in the XAML editor. At runtime, PowerBuilder .NET uses the same calculation for converting positive values as PowerBuilder Classic, but the converted values apply to both the text box and calendar parts of the control. For more precise font sizes that do not depend on unit conversion, you can set negative values at runtime.

Control property or functionality	PowerBuilder Classic behavior	PowerBuilder .NET behavior
DatePicker ValueChanged event	Date entry by user triggers event automatically.	User must press Enter key or change focus before the event can be triggered.
EditMask default mask when decimal mask is invalid	No default mask. Behavior is undefined.	EditMask uses ##,###.## as its default mask.
EditMask empty mask for Date, Time, and DateTime datatypes	Uses static masks: dd/mm/yyyy, hh:mm:ss:fff, and dd/mm/yyyy hh:mm:ss:fff.	Uses masks based on the culture set for the runtime operating system.
EditMask with percent sign in mask string	The percent sign is appended to any numeric value entered. For example, for a 0.00% mask, if you set the Text property to 5, the resulting value is 5.00%.	The numeric value set for the Text property is first multiplied by 100 before the percent sign is added. For a 0.00% mask, if you set the Text property to 5, the resulting value is 500.00%. This matches the behavior of DataWindow edit formats that change calculated decimal values into percentages.
Graph control availability	Standalone graph controls can be placed on any window or user object, as well as in DataWindow objects.	Graph controls are supported in the DataWindow painter only. By default, there is no WPF Graph control in the Toolbox for inclusion on WPF windows or visual user objects. Although you can migrate PowerBuilder Classic, applications with independent Graph controls, the Graph controls do not appear in WPF applications, and calls to these controls do not work.
GroupBox assembled controls	Controls that are inside a GroupBox belong to the container of the GroupBox.	Controls that are inside a GroupBox belong to the GroupBox and not to the container of the GroupBox. If you make the GroupBox invisible at runtime, its inner controls will also be invisible.

Control property or functionality	PowerBuilder Classic behavior	PowerBuilder .NET behavior
Menu focus	Menus are dependent on windows or other container objects, and do not gain or lose focus. The MenuFocusable property has no meaning in PowerBuilder Classic.	Menus are independent controls in PowerBuilder .NET. By default, if a WPF control has focus when you click a menu item, the LoseFocus event of the control is triggered, and if you make a menu selection that returns focus to the WPF control, the GetFocus event of the control is triggered. You can prevent these events from being triggered by setting the MenuFocusable property to false. However this also prevents application users from being able to scroll through the menu using arrow keys.
MonthCalendar **SetDateLimits** function	Makes dates outside the limits unselectable.	Makes dates outside the limits invisible.
MonthCalendar DateSelected event	Clicking in blank area of control does not trigger event.	Clicking in blank area of control triggers event.
MonthCalendar RButtonDown and GetFocus events	Right-clicking a MonthCalendar that does not have focus triggers only the RButtonDown event.	Right-clicking a MonthCalendar that does not have focus triggers the RButtonDown and GetFocus events.
RichTextEdit Selected Text Object dialog box	Users can format selected text through selections in the Selected Text Object dialog box.	RichTextEdit controls take advantage of WPF functionality to render rich text. Although the Selected Text Object dialog box is not available to application users, they can still modify the format of selected text from the control toolbar at runtime.

Control property or functionality	PowerBuilder Classic behavior	PowerBuilder .NET behavior
Tab control TabPosition values	TabPosition values and meaning: • TabsOnBottomAndTop! – tabs before the selected tab are on top; the selected tab, and tabs after it, are on the bottom. • TabsOnLeftAndRight! – tabs before the selected tab and the selected tab are on the left; tabs after the selected tab are on the right. • TabsOnRightAndLeft! – tabs before the selected tab are on the left; the selected tab and tabs after it are on the right. • TabsOnTopAndBottom! – tabs before the selected tab, and the selected tab, are on top; tabs after the selected tab are on the bottom.	TabPosition values and meaning: • TabsOnBottomAndTop! – tabs are at the bottom. • TabsOnLeftAndRight! – tabs are on the left. • TabsOnRightAndLeft! – tabs are on the right. • TabsOnTopAndBottom! – tabs are on top.
TabPage control header	Tab pages appear in a single line.	When there are several tab pages, they may appear in more than one line. This can affect the appearance of controls in a tab page.
Window MDI appearance	MDI windows appear as sheet windows. Sheet windows can be arranged in layers, in vertical or horizontal tiling patterns, or in cascades.	MDI windows appear as tabbed documents.
Window MDI behavior	If there is a position parameter in the OpenSheet call, sheet windows are added to the specified menu item when they are opened. Users can select the sheet window they want to access from that menu.	If the user opens enough tabs that they do not fit in the frame window, there is a drop-down menu on the right-hand side that allows the user to select a sheet.

Control property or functionality	PowerBuilder Classic behavior	PowerBuilder .NET behavior
Window Resize event	Calling the window **Hide** function does not affect the Resize event.	Calls to the **Hide** function causes a window to lose focus, and a subsequent Resize event fails. You must remove **Hide** and **Show** scripts to trigger the Resize event successfully.

For behavior changes in DataWindow Runtime controls, see *Behavior Changes for DataWindow Objects*.

Conditional Compilation in PowerBuilder .NET Targets

Although you do not need to use conditional code blocks in PowerBuilder .NET to reference .NET classes as in PowerBuilder Classic, you can still use the PBDOTNET and DEBUG preprocessor symbols for conditional compilation in PowerBuilder .NET targets.

PowerBuilder .NET also recognizes the PBWPF preprocessor symbol, which lets you conditionally code blocks of script exclusively for WPF Window Application targets. The code in these blocks is not parsed for other target types that share objects with WPF Window Application targets.

PowerBuilder .NET processes code inside PBDOTNET code blocks for WPF Window Application, NVO .NET Assembly, WCF Service, and PB Assembly targets. When the Enable Debug Symbol check box is selected in the Project painter, PowerBuilder .NET also processes code that is bracketed by the DEBUG preprocessor symbol.

PowerBuilder .NET ignores all code in PBNATIVE, PBWINFORM, PBWEBSERVICE, and PBWEBFORM code blocks in the targets that you migrate from PowerBuilder Classic. The conditional code blocks in migrated targets are kept in place, and enabled only for those symbols that are valid in PowerBuilder .NET.

PowerBuilder .NET can process code in blocks with the NOT operator, even if the preprocessor symbol used is not normally valid in PowerBuilder .NET targets. You can also enable invalid code blocks in migrated applications by changing the preprocessor symbols to symbols that are valid for your current target.

For more information on conditional compilation, see *Referencing .NET Classes in PowerScript* in the PowerBuilder Classic HTML help.

Some preprocessor use cases in PowerBuilder .NET include:

- ```
  #if Defined NOT PBNATIVE  then
              /// code will be executed
  #end if
  ```
- ```
  #if Defined PBWINFORM
       /// code will be ignored
  ```

```
#elseif defined PBWEBFORM then
     /// code will be ignored
#else
    /// code will be executed
#end if
```

- ```
  #if Defined PBWPF then
            /// code will be executed
  #end if
  ```

Memory Tuning for Large Applications

OutOfMemory errors can occur when you migrate, compile, or build a large PowerBuilder .NET application. To prevent these errors, you can turn on 3G memory support for your operating system.

You also must make the .NET Framework build and compile executables large address aware.

1. Turn on the /3GB switch for your operating system:
   - On Windows XP, add the "/3GB" switch immediately after the "/fastdetect" switch in the system `boot.ini` file.

     See *http://www.fmepedia.com/index.php/How_do_I_turn_on_the_3GB_Switch* or *http://technet.microsoft.com/en-us/library/bb124810(EXCHG.65).aspx* for information on editing the `boot.ini` file.
   - On Windows Vista or Windows 7, right-click Command Prompt in the Accessories program group of the Start menu, select Run as Administrator, then enter "bcdedit /set IncreaseUserVa 3072" and restart the computer.

     You can later remove the /3GB switch by entering "bcdedit /deletevalue IncreaseUserVa" in the same Command Prompt dialog box and restarting the computer.
2. Download and install a tool to enable applications to be large address aware.
3. Use the tool to enable `msbuild.exe` and `csc.exe` to be large address aware.

   These build and compile executables are under `C:\Windows\Microsoft.NET\Framework`, in the subdirectory for the current version. You must enable the executables in separate operations.
  ```

Graphic User Interface

Although PowerBuilder .NET and PowerBuilder Classic share many GUI elements, there are also a number of differences.

Visual Studio Shell Features

Because PowerBuilder .NET is built on the Visual Studio 2010 platform, some of its features are different from those in PowerBuilder Classic.

The features in the following table are supplied with the Microsoft Visual Studio shell. You can use the F1 key from a Visual Studio shell feature to open the H2 help, and in Visual Studio shell dialog boxes, you can click the question mark icon in the title bar for the H2 help. The help for these features is also supplied by Microsoft.

Visual Studio shell feature	Description or comment
XAML editor	The eXtensible Application Markup Language (XAML) allows you to code presentation aspects of your applications separately from the business logic that you code in the PowerBuilder .NET painter Script views. Changes that you make in a painter Layout view (Design pane) are reflected in the XAML editor code, and changes you make in the XAML editor can be seen in the Design pane. You can view the XAML editor for a painter only when the Design pane is also open, although you can collapse either of these panes and alternately display the XAML editor or the Design pane by selecting the corresponding tab.
Object Browser	The Object Browser allows you to view all .NET Framework classes. It is accessible from the View menu. The PowerBuilder Classic Browser is not currently available in PowerBuilder .NET.
Source control	Source control functionality is available from the **File > Source Control** menu of PowerBuilder .NET. PBG files are no longer required, because exported objects are not compiled into a binary PBL file as in PowerBuilder Classic. The Visual Studio shell source control functionality also does not use PBC files.
Code Snippets Manager	The Code Snippets Manager replaces the PowerBuilder Classic Clip window, and allows you to add XML snippets as well as PowerScript snippets. You can insert snippets from the context menu in painter Script views. The Code Snippet Manager is available from the Tools menu.

Visual Studio shell feature	Description or comment
Command window	Open the Command window from the **View > Other Windows** menu. It allows you to execute commands from the design time environment. IntelliSense is active in the Command window to assist you in entering commands for your targets.
Document Outline	The Document Outline view enables you to see a list of controls available in the current painter. It replaces the Control List in PowerBuilder Classic, but it also shows the control hierarchy. You can open the Document Outline from the **View > Other Windows** cascading menu.
Toolbox	Replaces the tool selection facility in the painter toolbars and Insert menu in PowerBuilder Classic. Open the toolbox from the PowerBuilder .NET View menu. Add or remove toolbox items from the **Tools > Choose Toolbox Items** menu.
IntelliSense	Replaces PowerBuilder Classic autoscript, but also provides selections for classes, methods, and members of all assemblies included in target references. IntelliSense is available in painter Script views, in the XAML editor, and from the Edit menu.
Properties window	The Properties window replaces the Properties view in PowerBuilder Classic painters. It allows you to sort properties by category or alphabetically, and depending on the active painter, can include a search filter or a status line that provides descriptions for certain selected properties. You open the Properties window from the **View > Properties Window** menu.
Find and Replace	The PowerBuilder .NET Find and Replace dialog box has tabs for Quick Find, Quick Replace, Find in Files, and Replace in Files. It is available from the **Edit > Find and Replace** menu.
Layout view	The painter Layout views use the Visual Studio designer and are closely associated with the XAML editor. A magnifier bar in the upper-left corner of each Layout view allows you to change the magnification of objects and controls at design time without changing their size at runtime.
Options dialog box	The Options dialog box replaces the System Options dialog box in PowerBuilder Classic. Instead of tabs, it uses a single navigation pane to navigate pages, which are organized in groups. It is available from the Tools menu.

Visual Studio shell feature	Description or comment
File editor	The Open With dialog box replaces the File Editor in PowerBuilder Classic. It allows you to select an editor for selected items in the Solution Explorer. It also allows you to add editors that you want to use to view these files, and set a different editor as the default file viewing selection. Use Solution Explorer context menus to open this dialog box.
Text formatting	In PowerBuilder .NET, you can format text in painter Script views and in the XAML editor with the **Edit > Advanced** cascading menu items.
Task list	Replaces the PowerBuilder Classic To-Do List. Open the Task list by selecting the **View > Other Windows > Task List** menu item.

Other features, such as the Start page, were inspired by similar features in Visual Studio 2008. The Start page opens in the painter area of the PowerBuilder .NET IDE and can remain open while you work in object painters and editors. It displays a news channel (that you can set on the Startup and Shutdown page of the Options dialog box) and a list of recentPowerBuilder .NET solutions. You can open the Start page at any time by selecting **View > Other Windows > Start Page** from the menu.

Solution Explorer in PowerBuilder .NET

The Solution Explorer in PowerBuilder .NET works in many ways like the System Tree in PowerBuilder Classic, although for some functionality it appears more similar to the Solution Explorer in Microsoft Visual Studio.

The System Tree is typically the central point of action in PowerBuilder programming, providing access to all the objects in a PowerBuilder workspace.

The System Tree in PowerBuilder Classic displays objects in PowerBuilder libraries (PBLs), whereas in PowerBuilder .NET, PBLs have been converted to directories, and the Solution Explorer displays the contents of the PBL directories only as separate files. The implementation of the System Tree as a Solution Explorer extension in PowerBuilder .NET has or enables:

- Integration with the Visual Studio shell "Find" subsystem
- Provision of a more natural framework for opening editors
- Replacement of an ad hoc framework for source control
- Provision of a more natural framework for workspace and project persistence
- Use of extensions for actions like code refactoring
- Ability to drag and drop files from the Windows Explorer to targets and libraries
- Listings of all assemblies used by a target in a single References folder

- Ability to select a set of nodes to perform a common action, such as applying a comment or changing a common writeable property
- Ability to host targets with multiple application objects and to set one of those objects as the current application object.

Some functionality operates similarly in the PowerBuilder Classic System Tree and in the PowerBuilder .NET Solution Explorer. Double-clicking an item in the Solution Explorer executes the default action for that item. Right-clicking an item shows the context menu for that item.

The Open and Open With context menu items in the PowerBuilder .NET Solution Explorer deliver the same functionality as the Edit and Edit Source context menu item labels in the PowerBuilder Classic System Tree. However, the Open With menu allows you to select the editor you want to use to view the object. The PowerBuilder .NET Solution Explorer also has separate Open Layout and Open Script context menu items for windows and visual user objects.

The following context menu items on a PowerBuilder Classic System Tree object are not available in the Solution Explorer in PowerBuilder .NET:

- PBNI Extensions – PowerBuilder .NET library nodes do not have the Import PowerBuilder Extension context menu item for importing PBNI extensions.
- Optimize – in PowerBuilder Classic, the Optimize menu item provides a way to clean up deleted objects and items that have been accumulated in target PBLs. This is not necessary in PowerBuilder .NET.
- Migrate – the PowerBuilder Classic migrate capability assumes you can view a target in the System Tree that does not list objects in an expanded PBL format. In PowerBuilder .NET, objects are always listed in an expanded format, so the Migrate context menu item is not needed

  The other use of the Migrate context menu is to regenerate Pcode from the sources for an object; this is also not needed in PowerBuilder .NET.
- Regenerate – like the Optimize and Migrate context menu items, the Regenerate menu item is not needed in the PowerBuilder .NET Solution Explorer because objects are always listed in an expanded format.

PowerBuilder .NET Toolbox

The Toolbox displays icons for controls that you can add to applications.

The Toolbox contains a list of icons representing application controls and similar components. You can drag a Toolbox item onto a design surface (such as a window painter), or copy and it paste into a code editor, to create an instance of the item. You can tailor the Toolbox by adding and deleting items in the list. The Toolbox is available from the **View** menu.

Depending on the type of design window that is open, the Toolbox might be divided into one or more of these tabs:

- General – this tab is always available. Use the General tab to copy items from your application and store them for easy reuse, including:
  - Controls
  - Custom or third-party components (for WPF applications)
  - Reusable text or code snippets

  For example, you can highlight some code in a script window and drag it to the General tab to store it there.
- WPF Controls – this tab is available with WPF window painters. It is preloaded with icons for standard WPF controls, like ListBox and Tab.
- Layouts – this tab is available with WPF window painters. It contains icons for PresentationFramework layouts, like Canvas and Grid layouts, which control the sizes and positions of elements in your WPF window.
- DataWindow Controls – this tab is available with the DataWindow painter. It contains icons for controls that you can add to DataWindows.

See the Visual Studio help for details about basic Toolbox usage and customization.

Document Outline

The Document Outline provides a single view for navigating the controls in the selected object.

The Document Outline window displays information when certain PowerBuilder objects are open in Layout view. It is similar to the Control List in PowerBuilder Classic, except that it shows the controls in a hierarchy. For DataWindow objects, the controls are also grouped by band (header, detail, summary, and footer).

To open the Document Outline, click**View > Other Windows > Document Outline**.

You can navigate the Document Outline by expanding and collapsing the levels in the hierarchy. When you select the control in the Document Outline, it is selected in the Designer.

The Document Outline is a Visual Studio feature; refer to the Visual Studio isolated shell help.

PB Object Outline

The PB Object Outline provides a convenient, single view for navigating and manipulating object components.

The PB Object Outline shows all the components of a PowerBuilder object that is open for editing. The components are grouped into categories such as Controls, Events, and Functions. If you open or switch to a different painter, the PB Object Outline changes to show the components in the painter. If no PowerBuilder object is open, the window contains no outline.

In the outline, you can:

- Expand and collapse individual folders, or use the buttons at the top to expand or collapse all folders.
- Expand a nonvisual object that is open in a nonvisual object painter, or used in a window or visual user object, to show its events and functions.
- Expand interfaces to show any indexers, events, functions, and .NET properties. Interfaces can extend other interfaces. When an object in the outline extends other interfaces, those interfaces expand to the same information as the interface object. You can expand each level recursively until there are no more extended interfaces.
- Expand controls to view nested controls and control scripts.
- Expand enumeration types into the enumerated values.
- Expand structure types into the structure elements.

Object icons

Each object in the PB Object Outline has an icon representing its object type.

Events, functions, indexers, and .NET properties have an additional state icon that indicates the location of the object's script: either locally (with the object), in an ancestor only, or both locally and in an ancestor. The indexer and .NET property state icons reflect the combined get and set states, because an element can represent both methods.

An element that is under source control displays the same source control state icon in the outline and in the system tree.

Sorting and filtering options

The context menu for scripts includes options for sorting and filtering the outline.

For indexers, functions, events, and .NET properties, only local scripted objects are shown, by default, in the outline. Context menu options let you include unscripted and scripted ancestor objects.

The default sorting scheme groups all locally scripted objects first, followed by objects that are scripted in an ancestor, followed by unscripted objects. Context menu options let you change the sorting scheme.

Items in menu and global structure outlines appear in the order defined in the menu, unless you change the outline sort order.

When you use context menu options to change sorting or filtering, you override the global settings that are defined in the Script Lists page of the Options dialog box, which apply to all instances of the PB Object Outline. Changes to the current instance of the PB Object Outline do not affect any other instances.

Context menu actions

Context menus for items in the outline can include one or more of these actions:

- New *item* – opens a new script for an indexer, event, function, .NET property, enumeration, or structure.

For example, in the PB Object Outline for a window object, if you right-click the Events folder and select **New Event**, the window's Script view opens with a New Event tab, where you can complete the event definition.

- Nonvisual Object actions – the context menu for nonvisual objects provides two methods for adding a nonvisual object:
  - New – displays a submenu of built-in nonvisual class types to choose from, and it creates a new nonvisual instance of the type you select. This method is convenient, but you cannot insert a named nonvisual object from your library.
  - Add Nonvisual Object – opens the Select UserObject dialog box, from which you can insert a previously created nonvisual user object into the current object.
- Go To Declare *object* – opens a Declare script for a using, interface, instance variable, shared variable, global variable, or external function.
- Go To Script – opens the Script view tab of the selected outline item.
- Edit Namespace – opens the Script view Namespace \ Usings tab for the selected namespace.
- Remove – deletes the selected element.
- Properties – opens the Properties view for the selected element.

Toolbars in the IDE

The toolbars in the PowerBuilder .NET IDE are somewhat different from the toolbars in the PowerBuilder Classic IDE.

In PowerBuilder .NET toolbars:

- The PowerBar is now called the Standard toolbar, and includes additional functionality (navigation and search tools) from the Visual Studio isolated shell.
- The Standard toolbar includes buttons for building and deploying the current target in place of the buttons for building and deploying the workspace.
- PainterBars are now called by the name of the painters they support.
- StyleBar functionality is replaced by properties in the PowerBuilder .NET Window and DataWindow painters.
- There is a new Help toolbar that is supported by the Visual Studio isolated shell.
- There are two toolbars for the PowerBuilder .NET Debugger; one provides debug commands, and the other lists, and allows you to select, the process, thread, and stack frame of the application you are debugging. These toolbars use button images and functionality from the Visual Studio isolated shell.
- You can show all toolbars at any time, whether or not they are supported by the current painter.
- Toolbars uses drop-down lists for selecting targets to run and debug.
- The Toolbar Options drop-down arrow at the rightmost end of each toolbar allows you to add or remove buttons from a toolbar, reset a toolbar, and open the Customize dialog box.

You can also right-click anywhere in the menu bar or in any of the toolbars to select a toolbar to display and open the Customize dialog box.

For information on the Customize dialog box, see *http://msdn.microsoft.com/en-us/library/52y65fyz(VS.71).aspx*.l

- You can display text labels for individual toolbar items and change, edit, or reset their associated button images from the button context menus while the Customize dialog box is open.

  The same functionality is available by clicking the Modify Selection button on the Commands page of the Customize dialog box while a toolbar button is selected.

  The ability to display text labels on individual toolbar items provides for finer control of the user interface than in PowerBuilder Classic, where the Show Text command in toolbar context menus applies to all toolbar buttons and all toolbars.

Options Dialog Box

The Options dialog box provides settings that let you tailor your development environment.

PowerBuilder .NET options are set in the the Options dialog box from the Visual Studio shell, instead of the System Options dialog box that PowerBuilder Classic uses. The two dialog boxes are different in several ways:

- Unlike the System Options dialog box, the Options dialog box is not tabbed. Instead, options are contained in pages. Pages are grouped into categories in a single navigation pane.
- The Options dialog box includes some pages contributed by Visual Studio. The Visual Studio shell provides help for those pages.
- Some options are not implemented in the Options dialog box, because they are not applicable in . For example, there is no Automatically Clear Output Window option, because that function is implemented elsewhere by the Visual Studio shell.
- Options available in both IDEs might be organized differently. For example, options in the System Font tab of the System Options dialog box appear in the Font and Colors page of the Options dialog box.

The following sections describe only the pages and options that PowerBuilder adds to the Options dialog box. Press F1 for Microsoft online help on the Visual Studio options and pages that are not documented here.

Environment > Documents

The options in this page control the display of documents in PowerBuilder .NET, and enable you to manage external changes to documents and files:

- Detect when file is changed outside the environment - a message immediately notifies you of changes to an open file that have been made by an editor outside the IDE. The message allows you to reload the file from storage. Unselected by default.

- Auto-load changes, if saved - if Detect when file is changed outside the environment is checked and an open file in the IDE changes outside the IDE, a warning message is generated by default. If this option is enabled, no warning will appear and the document will be reloaded in the IDE to pick up the external changes.
- Allow scripts with syntax errors to be saved - selected by default.

Environment > Firewall

The options on this page control proxy server settings when you add a Web service at design time and your development machine is behind a firewall.

- Host - Name of the proxy server that you use to access Web pages
- Port - Port used for connecting to the proxy server
- UserName - User name for accessing the proxy server
- Password - Password for the user accessing the proxy server
- Use above values as system defaults

See *About Web services* in the PowerBuilder Classic online help for more information about using a proxy server behind a firewall.

Environment > Startup and Shutdown

The Startup and Shutdown page in the Environment category contains these options:

- Initialization path – specifies the initialization path for PowerBuilder. By default, the initialization path is either `c:\documents and settings\`*`userName`*`\local settings\application data\sybase\powerbuilder `*`version`* (Windows XP) or `c:\users\`*`userName`*`\AppData\Local\Sybase \PowerBuilder `*`version`* (Vista and later), where *userName* is the user name for the currently logged in user.

  Notes: If you change the initialization path, be sure to copy all INI, property, and license files from the original initialization path to the path that you enter.

  The Initialization path setting is shared between PowerBuilder .NET and PowerBuilder Classic.

- Load last loaded solution – at start-up, opens the last saved solution in its previous state. Any files that were open in the solution when it was last closed are opened and appear when you start . If no solution is loaded when you exit the product, no solution is loaded when you return.
- Show Start Page – displays the Start Page when you open PowerBuilder .NET.
- Open Home Page – at start-up, displays the default Web page specified by the Home page option in the Environment > Web Browser page.
- Start Page news channel – specifies the RSS feed for the Start Page.

- Download content every *n* minutes – specifies how often the IDE checks for new RSS feed content and product headlines for the Start Page. If you do not select this option, the Start Page does not display RSS feed content and product headlines.
- Free database driver libraries on disconnect – releases libraries that are held in memory after PowerBuilder disconnects from a database in the development environment.

  Notes: Selecting this option reduces memory use, but at the cost of performance loss and possible problems with some database management systems associated with the freeing and subsequent reloading of libraries. To free libraries held in memory at runtime, set the FreeDBLibraries property to true on the General category of the application's Properties view in the Application painter.

  The Free database setting is shared between PowerBuilder .NET and PowerBuilder Classic.

Solutions > General

The Solutions General page has these options:

- Show layout item for visual objects in Solution Explorer - displays the XAML layout file node for any SRW and visual standard user object files.
- Track active item in Solution Explorer - tracks the object (not necessarily an individual event or function) that currently has focus in an editor.
- Automatically set the current target - automatically changes the current target as you change focus in the Solution Explorer and open painters. The current target establishes context for some PowerBuilder .NET operations, and is displayed in bold in the Solution Explorer.
- Save the open document state to facilitate reloading the open documents when next opening the solution - opens documents from the previous session that were in read-write mode and does not open documents that were in read-only mode.

Solutions > Build and Run

Most of the options on this page are described in the Visual Studio help that is displayed when you press F1. PowerBuilder .NET adds the following options, both selected by default:

- Always show Error List if build finishes with errors
- Show Output window when build starts

Note: Output verbosity for .NET build corresponds to the MSBuild project build output verbosity option in the Visual Studio help.

Solutions > Show Folders

The options in this page show or hide specific folders within objects in the Solution Explorer. By default, all folders are shown. For example, selecting the Indexers option makes the Indexers folder visible after you reload the solution.

Solutions > Show Objects
The options in this page specify the types of PowerBuilder objects and other resources to show in the Solution Explorer. By default, all object types are shown. For example, you can set the DataWindows option to hide DataWindows when you reload the solution.

Text Editor > PowerScript
The options in this page apply to the PowerScript editor. They do not apply to editors for other languages.

- General – for information about these options, see the Visual Studio help for the *Text Editor > All Languages* page.
- Tabs – for information about these options, see the Visual Studio help for the *Text Editor > All Languages > Tabs* page.
- Formatting – specify the text case that is used for PowerScript keywords:

  lower case
  UPPER CASE
  Title Case –
- Miscellaneous – specify whether PowerBuilder .NET opens scripts in the current script view, or in a new script view.

DataWindow Painter > General
These options control the layout of DataWindows:

- Snap to Grid – makes controls snap to a grid position when you place them or move them.
- Show Grid – displays a grid in the painter.
- Show Ruler – displays a ruler. The ruler uses the unit of measurement specified in the DataWindow object's Unit property.
- X – specifies the width of the grid cells.
- Y – specifies the height of the grid cells.
- Show Edges – displays the boundaries of each control in the DataWindow object.
- Retrieve on Preview – retrieves all the rows from the connected data source when a DataWindow object opens in the Preview view.

DataWindow Painter > Prefixes
The options on this page set a default prefix for each type of DataWindow object you create.

Menu Painter
This page contains a single option that sets the default prefix for menu objects.

Object Prefixes
The options on this page set the default prefixes for window and user objects.

Property Editors
The options on this page enable you to control the display of images in the property drop-down lists.

Script Lists
The filter and sort options on this page enable you to configure the script lists in the PB Object Outline, Script Editor, or Solution Explorer.

The filter options specify types of scripts to include or exclude in lists:

- Show inherited scripts and controls – includes events, functions, controls, and nonvisual objects in the list that are referenced only in ancestors.
- Show unscripted system events – includes events that are defined as system-supplied, like Activate. Scripted versions are listed regardless of this setting.
- Show unscripted system functions – includes functions that are defined as system-supplied, like ArrangeSheets. Scripted versions are listed regardless of this setting.
- Show unscripted .NET events and functions – includes .NET events and functions that are built-in, like the Equals method. Scripted versions are listed regardless of this setting.

The sort options specify how to group and sort list items:

- Alphabetically – sorts scripts by event or function name.
- Scripted local first – groups all locally scripted events or functions first, sorted alphabetically within each group.
- Scripted local, scripted ancestor, unscripted – groups all locally scripted events or functions first, followed by events or functions scripted in an ancestor, followed by events or functions that are not scripted.
- Scripted anywhere, unscripted – sorts all scripted events or functions together, regardless of where they are scripted.

New and Inherit From Object Dialog Boxes

The New and Inherit dialog boxes have been changed for PowerBuilder .NET.

The New dialog box enables you to access wizards to create most of the objects you need for your application. It is similar to the same dialog box in PowerBuilder Classic, with these differences:

- Integrated object wizards – instead of tabbed pages, the PowerBuilder .NET New dialog box has a single page where you can select an object type and launch its wizard.
- Filter – before selecting an object type, you can simplify the list of available objects by entering a filter expression.
  A simple filter expression is a string of characters that matches only a subset of objects. For advanced filtering, you can include any of the special **Match** function characters. Expressions are not case-sensitive.

Examples:

- `wpf` matches WPF and wpf.
- `listbox` matches include DropDownListBox and DropDownPictureListBox.
- `^listbox` matches only Listbox.
- `Button$` matches CommandButton, RadioButton, and PictureButton.
- `...x` finds any three characters that are followed by x.

See the online help on the **Match** function for details about special characters.

- Object tree – the main part of the window contains a tree view of all the object types that you can create, organized into a hierarchy of folders. When you select an object, a brief description of it appears below the view.
- Wizard navigation buttons - In addition to enabling you to step back and forth through its pages, the navigation buttons provide two different methods for creating an object:
  - **Next** opens a new wizard page, where you can specify a name and other properties for the new object.
  - **Finish** creates a new, untitled object and closes the wizard. You can finish configuring the object later.

    Note: The **Finish** button is enabled as soon as you have entered enough information to create the selected object.

The Inherit From Object dialog box has these changes:

- Libraries – in addition to PBLs, enables you to choose one referenced assemblies in the target.
- Objects – shows the objects in the PBLs that you select in the Libraries list. Shows classes and controls for selected assemblies.
- Object types – Classes and Controls are added to the drop-down list.

Customizing the New Dialog Box

You can change the items that appear in the New dialog box.

To customize the New dialog box:

- To remove an item, right-click its entry in the object view and choose **Remove**.
- To restore items that have been removed, right-click a category folder and select:
  - **Reset PB Object Category** to restore the objects that have been removed from the category.
  - **Reset All** to restore all previously removed objects to the view.

PowerBuilder .NET Painters

In PowerBuilder .NET, as in PowerBuilder Classic, edit objects—such as applications, windows, and menus—in painters.

PowerBuilder .NET includes painters that are not available in PowerBuilder Classic, such as the Interface and Enumeration painters. You can also use most object painters in PowerBuilder .NET to create indexers and .NET properties, and to declare namespaces and using namespace directives.

About the Enumeration Painter

The Enumeration painter is similar to the Structure painter. You can use the painter to define global enumerations for your applications or components, and object painters to define local enumerations for objects in your applications or components.

Each row in the Enumeration painter represents an item in the enumeration you are defining. The rows contain editable text cells under three columns: Name, Value, and Comments. Specify a name for each item to include in the enumeration. Enumeration values must be whole numbers.

If you select the Flags option in the Enumeration painter, the values for each item in the enumeration are treated as bit fields.

The Enumeration painter for local enumerations includes the Enumeration Name field, where you enter the name for the local enumeration. This field does not appear in the painter for global enumerations, as a global enumeration is saved as a file when you select **File > Save**. The extension for a global user-defined enumeration that you save in the Enumeration painter is `.sre`.

System.Enum is the base class for all PowerBuilder .NET enumerations. You can assign enumerations using System.Enum, System.Object, ANY, or integral (int, unit, byte, sbyte, long, ulong, longlong, and ulonglong) datatypes.

For example, the following code assigns an integer to an enumeration named "myenum":

```
int a = 1
myenum e
e=a
```

When the Enumeration painter is active, its context menu includes Insert Field and Delete Field menu items. Insert Field adds an empty row before the current row and the sequential values in the rows after the empty row are automatically adjusted upwards. Delete Field removes the current row and the values in the rows following the deleted row are adjusted downwards sequentially.

About the Interface Painter

You can create interfaces in the Interface painter. It is similar to the Custom Class (nonvisual object) painter, with a few notable differences.

You can define functions, events, indexers, and .NET properties for interfaces using the prototype pane of the Interface painter. However, you cannot edit the scripting area of the painter, since scripting is not allowed for the members of an interface. You must do the scripting in the objects that implement the interface.

The fields in the prototype pane are similar to those for other painters. However, for functions that you create in the Interface painter, the Access field is not available. Access is always public for interfaces. Also, you cannot select Enumerations or Structures for interfaces.

When you select Declare from the drop-down list at the upper left of the Interface painter, you can select Global Variables, Global External Functions, Namespace, and Interfaces from the second drop-down list. You cannot declare instance or shared variables for an interface.

Project Painter User Interface

Use the Project painter to generate compiled resources—such as EXE and .NET assembly files—for runtime applications and components.

You can also locally execute the generated files from the design time environment by selecting Run from the Design menu of the Project painter, or from target or project context menus in the Solution Explorer. For WCF Client Proxy projects, you can select Generate Proxy from the Design menu of the Project painter, or from the project context menu.

The PowerBuilder .NET Project painter displays horizontally stacked tabs separating functional areas of a Project object rather than the PowerBuilder Classic painter style that uses top row tabs.

The options for full and incremental builds are not included on the General tab of the Project painter as in PowerBuilder Classic. Instead, you can specify the build scope by selecting Full Build or Incremental Build from the target and project context menus in the Solution Explorer. You can also select the build scope for the current target from the Build menu, or for the current project from the Design menu of the Project painter. If you open a WPF project, the WPF Project Painter Toolbar provides options for building and debugging the project, as well as checking dependencies between output assemblies.

The Resources tab in the Project painter for PowerBuilder Classic is not available in PowerBuilder .NET. Instead, you add resources to a target from a context menu in the Solution Explorer, or in the wizard when you create the target. The build process copies resource files to the project output directory.

PowerBuilder .NET does not use PBD files, so the Library Files tab in the Project painter has only a list box for Win32 dynamic library files; it does not have a separate list for PowerBuilder library files. Although Windows Forms projects in PowerBuilder Classic

require PBD files for applications with DataWindow, Query, or Pipeline objects, when you migrate a Windows Forms target to PowerBuilder .NET, the list of PowerBuilder library files on the project's Library Files tab is not processed during the migration.

Window Painter in PowerBuilder .NET

Design windows in the Window painter, which has several views where you specify how a window looks and behaves.

The default layout for the Window painter workspace has two views:

- The Layout view, where you design the appearance of the window and view the XAML. This view has buttons that let you split the Design and XAML panes either vertically or horizontally, and allows you to hide one of the panes. You can also swap the pane positions by clicking the double arrow icon between the Design and XAML panes.

- The Script view, where you modify behavior by coding window and control scripts.

You can also use the context menu in the Solution Explorer to open only one of these tabs.

Building a New WPF Window

You can build a window from scratch, for example, to create windows that are not based on existing windows.

When you build a window, you:

- Specify the appearance and behavior of the window by setting its properties
- Add controls to the window
- Build scripts that determine how to respond to events in the window and its controls

  To support these scripts, you can define new events for the window and its controls, and declare functions, indexers, properties, enumerations, structures, and variables for the window.

Creating a New WPF Window

Create a WPF Window from scratch.

Note: At any point during the creation process (if it is enabled) to create the window by accepting the default settings for the later steps.

1. Open the **New** dialog box.
2. In the PB Object node, select **WPF Window**.
3. Click **Next**.
4. Enter a title and a name for the window. Select the library in which to save the window. Click the Next button.

5. Select the layout for the window, then click **Next**.

Canvas	Defines an area within which you can explicitly position child elements by specifying coordinates relative to the canvas area.
Dock Panel	Defines an area within which you can arrange child elements either horizontally or vertically, relative to each other.
Grid	Defines a flexible grid area consisting of columns and rows. Use the Margin property to precisely position child elements in a grid.
Stack Panel	Arranges child elements into a single line that can be oriented horizontally or vertically.
Wrap Panel	Places child elements in sequential position from left to right, breaking content to the next line at the edge of the containing box.

6. (Optional) Specify a namespace, namespace shortcuts, and any interfaces, then click **Next**.
7. Review the WPF window characteristics, then click **Finish**.

The Window painter opens. The new window appears in the Window painter's Layout view and the XAML defining the Window shows in the XAML view.

Defining the WPF Window Properties

Every window and control has a style that determines how it appears to the user. Define a window's style by choosing settings in the Window painter's Properties view.

A window's style encompasses its:

- Type
- Basic appearance
- Initial position on the screen
- Icon when minimized
- Pointer

Note: As in PowerBuilder Classic, use the Properties view to define window properties. You can edit the properties in the XAML view as well, but editing the name in the XAML view may cause object corruption.

1. Select the window object to display the window's properties in the Properties view.
2. Choose the category appropriate to the property you want to specify.

To specify the window's	Choose
Name, type, state, color, and whether a menu is associated with it	PBGeneral

To specify the window's	Choose
Icon to represent the window when you minimize it	PBGeneral
Skin	PBGeneral
Transparency	Appearance
Position and size when it displays at runtime	PBOther
Default cursor whenever the mouse moves over the window	PBOther
Horizontal and vertical scroll bar placement	PBScroll
Toolbar placement	PBToolbar

Adding Controls to WPF Windows

You can add controls to a WPF Window by using the Toolbox.

When you build a window, place controls in the window to request and receive information from the user and to present information to the user.

Note: Although you can add, edit, or remove controls using the XAML view, it is not recommended because it may cause object corruption.

1. In the Toolbox, select the control you want to insert.
2. In the Layout view, specify the location of the control.

After you insert the control, you can size it, move it, define its appearance and behavior, and create scripts for its events.

MDI Applications in PowerBuilder .NET

Multiple Document Interface (MDI) is an application style you can use to open multiple windows (called sheets) in a single window and move among the sheets.

To build an MDI application, define a window with a type of MDI Frame, then open other windows as sheets within the frame. In a WPF application, these sheets open as tabbed documents.

You can find some general information about MDI applications here. For more information, see *Application Techniques*.

MDI frame windows

An MDI frame window has several parts: a menu bar, a frame, a client area, sheets, and (usually) a status area, which can display MicroHelp (a short description of the current menu item or current activity).

Client area

In a standard frame window, PowerBuilder automatically sizes the client area and the open sheets appear within the client area. In custom frame windows that contain objects in the client

area, you must manually set the size of the client area. If you do not correctly size the client area, the sheets open, but may not be visible.

When you build an MDI frame window, PowerBuilder creates a control named MDI_1, which it uses to identify the client area of the frame window. In standard frames, PowerBuilder automatically manages MDI_1. For custom frames, write a script for the frame's Resize event to correctly set the size of MDI_1. Code is also added to the XAML for the window.

Tabbed interface

Sheets open as tabs in the client area of a WPF frame window. You can use any type of window except an MDI frame as a sheet in an MDI application. To open a sheet, use either the **OpenSheet** or **OpenSheetWithParm** function.

Creating an MDI Frame Window

When you create a new window in PowerBuilder, its default window type is Main. Select Mdi! or MdiHelp! in the WindowType property to change the window to an MDI frame window.

When you change the window type to MDI, you must associate a menu with the frame. Menus usually provide a way to open sheets in the frame and to close the frame if the user has closed all the sheets.

Note: A sheet can have its own menu but is not required to. When a sheet without a menu is opened, it uses the frame's menu.

1. In the PowerBar, click the **New** button.
2. Select **WPF Window** and click **Next**.
3. Enter a window title and name, then click **Next** (or **Finish**).
4. (Optional) Select the layout and click **Next** (or **Finish**).
5. (Optional) Select a namespace and click **Next** (or **Finish**).
6. Click **Finish.**
7. When the window object is selected, open the Properties view.
8. In the PBGeneral category, set the WindowType property to **Mdi!** or **MdiHelp!**.

Using Sheets in a PowerBuilder .NET Application

In an MDI frame window, users can open sheets to perform activities. All sheets can be open simultaneously and the user can move among the sheets, performing different activities in each one.

Note: A sheet can have its own menu but is not required to. When a sheet without a menu is opened, it uses the frame's menu.

To open a sheet in the client area of an MDI frame, use the **OpenSheet** function in a script for an event in a menu item, an event in another sheet, or an event in any object in the frame.

For more information about **OpenSheet**, see the *PowerScript Reference*.

Menus and Toolbars for PowerBuilder .NET Applications

Adding customized menus and toolbars to your applications makes it easy and intuitive for your users to select commands and options.

Menus and Menu Items

Menus are lists of related commands or options that a user can select in the currently active window. Each choice in a menu is called a menu item.

Usually, all windows in an application have menus except child and response windows.

Menu items appear in a menu bar or in drop-down or cascading menus. Each item in a menu is defined as a Menu object in PowerBuilder.

Using menus

You can use menus in two ways:

- In the menu bar of windows - Menu bar menus are associated with a window in the Window painter and appear whenever the window is opened.
- As pop-up menus - Pop-up menus appear only when a script executes the PopMenu function.

Designing menus

PowerBuilder gives you complete freedom in designing menus, however, Sybase recommends that you follow conventions to make your applications easy to use. For example, keep menus simple and consistent; group related items in a drop-down menu; make sparing use of cascading menus, and restrict them to one or two levels.

A full discussion of menu design is beyond the scope of this documentation. Acquire information that specifically addresses design guidelines for graphical applications and apply those guidelines when you use PowerBuilder to create your menus.

Building menus

When you build a menu, you:

- Specify the appearance of the menu items by setting their properties.
- Build scripts that determine how to respond to events in the menu items. To support these scripts, you can declare functions, structures, and variables for the menu.

There are two ways to build a menu. You can:

- Build a new menu from scratch. See *Building a New Menu*.
- Build a menu that inherits its style, functions, structures, variables, and scripts from an existing menu, thereby saving yourself coding and time. See *Using Inheritance to Build a Menu*.

Menu Painter in PowerBuilder .NET

The Menu painter has several views where you can specify menu items and how they look and behave. You can also customize the style of a toolbar associated with a menu.

Menu Painter Views

There are several views that you use to create a menu.

You use two views to specify menu items that appear in the menu bar and under menu bar items:

This view	Displays
Tree Menu view	All menu items at the same time when the tree is fully expanded.
WYSIWYG Menu view	The menu as you will see it and use it in your application, with the exception of disabled menu items, unless you set Show Invisibles.

The Tree Menu view and the WYSIWYG Menu view are nearly equivalent. You can use either to insert new menu items on the menu bar or on drop-down or cascading menus, or to modify existing menu items. However, you cannot enter text in the WYSIWYG Menu view. You can enter the menu text in the Tree Menu view or the Properties view.

The menus in both views change when you make a change in either view. Unlike PowerBuilder Classic, any disabled menu items will not appear in the WYSIWYG view unless you enable **Edit > Show Invisibles**.

The two views are separated by a grid splitter. You can resize the views as needed, or hide one view entirely.

Specify menu properties in two views:

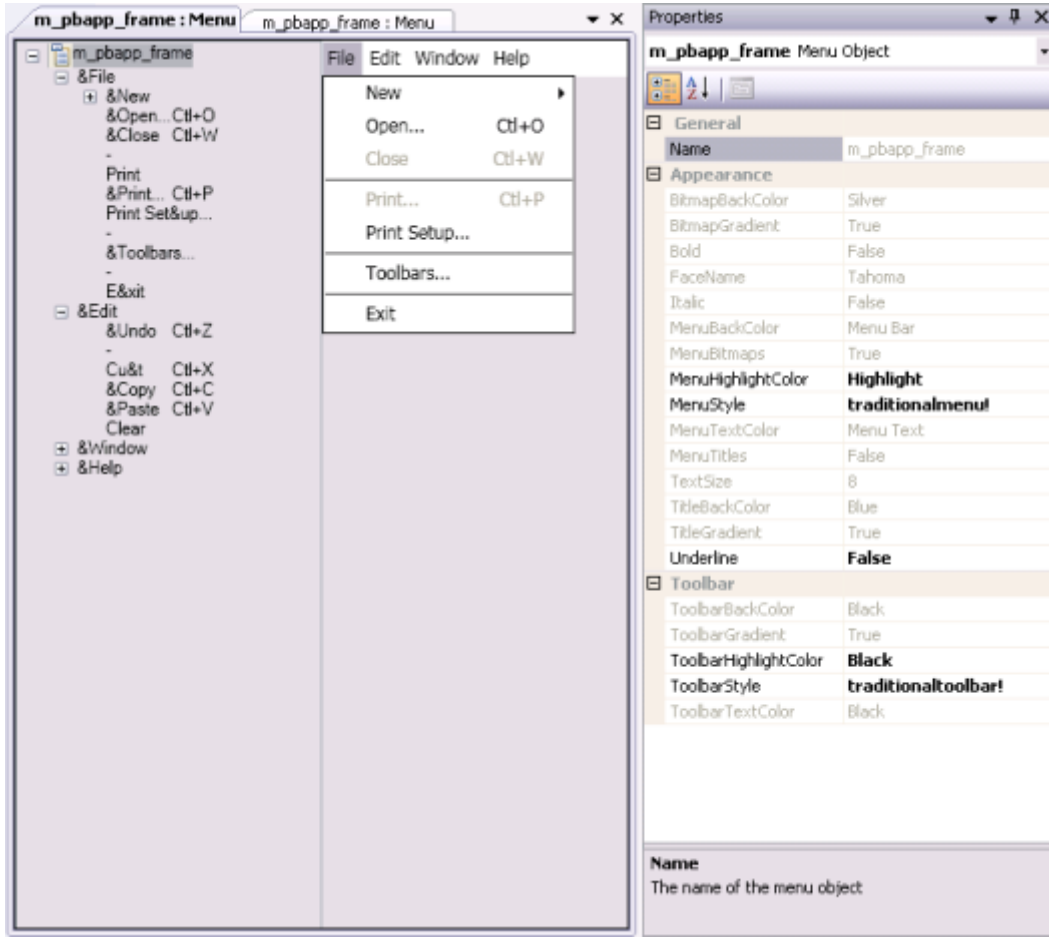
Use this view	To specify
Properties view (for the top-level menu object)	General, Appearance, and Toolbar categories for setting menu-wide properties
Properties view (for menu items)	Name, Menu Item, and Toolbar Item categories for setting properties for submenu items and toolbars

Note: When you select the menu items in the Tree Menu view, you can set their properties.

Views for the Top Level Menu Object

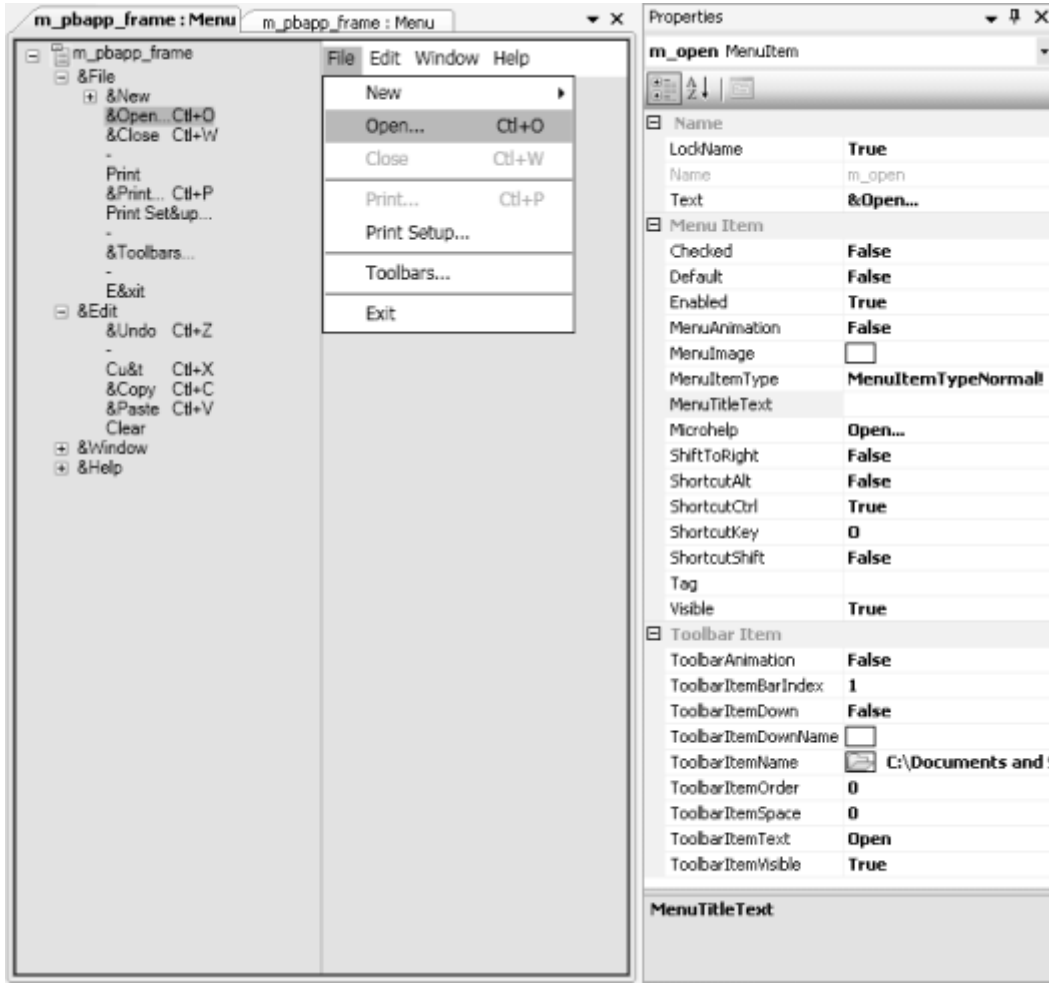
When you select the top-level menu item in the Tree Menu view, you can set its properties.

In the Menu painter layout for the top-level menu object, the Tree Menu view is on the left and the WYSIWYG Menu view is on the right. The Properties view displays the General, Appearance and Toolbar categories.



Views for Submenu Items

In the Menu painter layout for a menu item under the top level, the Tree Menu view is on the left and the WYSIWYG Menu view is on the right. The Name, Menu Item, and Toolbar Item categories are in the Properties view.



Menu Styles in PowerBuilder .NET

A menu can have a contemporary or traditional style.

By default, new menus use the contemporary menu style. Imported and migrated menus maintain their style.

Menu style	Description
Contemporary	A 3D-style menu similar to Microsoft Office 2003 and Visual Studio 2005 menus
Traditional	Window default menu style, which has a flat appearance

The contemporary menu style has a three-dimensional (3-D) appearance that can include images and menu title bands. With a contemporary menu, you can use scripts at runtime to set the MenuAnimation, MenuImage, and MenuTitleText.

Select a menu style in the Appearance category of the Properties view for the top-level menu object in the Menu painter. You must select the top-level menu object in the Tree Menu view of the Menu painter to display its Properties view.

If you select `contemporarymenu!` as the menu style, you can customize the display properties for that style, and apply them to all menu items in the current menu. If you select `traditionalmenu!` other menu style properties do not apply.

For more about using images for menus and toolbars, see the *PowerBuilder Users Guide*.

Building a New Menu in PowerBuilder .NET

You can build menus that are not based on existing menus.

Creating a New Menu

Build a new menu by creating a new Menu object and then working on it in the Menu painter.

To create a new menu:

1. In the PowerBar, click **New**.
2. In the PB Object category, select **Menu**.
   - To accept all of the defaults and create an untitled menu, click **Finish**.
   - To specify the menu details, click **Next**.
3. Accept the default menu name or enter a new one, and click **Next**.
4. Optional: Specify a namespace, using directives, or interface.
5. Click **Next**.
6. Click **Finish**.

The Menu painter opens and displays the Tree Menu view and the WYSIWYG view for defining the menu, and the General, Appearance and Toolbar categories in the Properties view for setting menu and toolbar properties.

Because you are creating a new menu and have not added menu items yet, the only content is an untitled top-level tree view item in the TreeMenu view.

Working with Menu Items in PowerBuilder .NET

A menu consists of at least one menu item on the menu bar and one or more items in a drop-down menu.

You can add menu items to:

- The Menu bar
- A Drop-down menu

- A Cascading menu

When you add a menu item, PowerBuilder assigns it a default name, which appears in the Name box in the Properties view. This is the name by which you refer to a menu item in a script.

The default name is `m_`*n* , where *n* is the lowest number that can be combined with the prefix to create a unique name. If you enter text when you create the menu item, the text you enter replaces the number. If the text you enter creates a duplicate name, PowerBuilder changes the name back to the number. Menu items in the Tree Menu view and WYSIWYG Menu view can have the same text, but they cannot have the same name in the Properties view.

Inserting Menu Items

There are three Insert choices in the context menu: Insert Menu Item, Insert Menu Item At End, and Insert Submenu Item. Use the first two to insert menu items in the same menu as the selected item, and use Insert Submenu Item to create a new drop-down or cascading menu for the selected item.

Suppose you have created a **File** menu on the menu bar with two menu items: **Open** and **Exit**. Here are the results of some insert operations:

- Select **File** and select **Insert Menu Item At End** from the pop-up menu
  A new item is added to the menu bar after the **File** menu.
- Select **Open** and select **Insert Menu Item** from the pop-up menu
  A new item is added to the **File** menu above **Open**.
- Select **Open** and select **Insert Menu Item At End** from the pop-up menu
  A new item is added to the **File** menu after **Exit**.
- Select **Open** and select **Insert Submenu Item** from the pop-up menu
  A new cascading menu is added to the **Open** menu item.

The first thing you do with a new menu item is add the first item to the menu bar. Then continue adding new items to the menu bar or to the menu bar item you just created. As you work, the changes you make appear in both the WYSIWYG and Tree Menu views.

Inserting the First Menu Bar Item in a New Menu

You must add an initial menu bar item.

Use this procedure if you want to add the menu bar item and then work on its drop-down menu.

1. In the Tree View view, select the first menu item, and then select **Insert Submenu Item** from the pop-up menu.
   PowerBuilder displays an empty box as a subitem in the Tree Menu view.
2. Type the text for the menu item and press **Enter**.
   The menu item shows the new text in the menu bar in the WYSIWYG Menu view and in the Tree Menu view.

After you have created the first menu bar item, you can add more items to the menu bar or start building drop-down and cascading menus.

Inserting Multiple Menu Bar Items in a New Menu

You can easily add multiple items to a menu.

1. In the Tree Menu view, select **Insert Submenu Item** from the pop-up menu.
   PowerBuilder displays an empty box as a submenu item in the Tree Menu view.
2. Type the text you want for the menu item. and press **Tab**.
   The text is added to the Tree Menu view and the WYSIWYG view, and PowerBuilder displays a new empty box as a submenu item in the Tree Menu view.
3. Repeat step 2 until you have added all the menu bar items you need.
4. Press **Enter** to save the last menu bar item.

Inserting Additional Menu Items on the Menu Bar

You can insert additional menu items to the end of the menu or before the selected item.

1. Do one of the following:
   - With any menu bar item selected, select **Insert Menu Item At End** from the pop-up menu to add an item to the end of the menu bar.
   - Select a menu bar item and select **Insert Menu Item** from the pop-up menu to add a menu bar item before the selected menu bar item.
2. Type the text you want for the menu bar item. Press **Enter**.

Adding a Drop-down Menu to an Item on the Menu Bar

After you have created the first menu bar item, you can build drop-down menus.

1. In the Tree Menu view, select the item for which you want to create a drop-down menu.
2. Select **Insert Submenu Item** from the pop-up menu.
   PowerBuilder displays an empty box.
3. Type the text for the menu item. Press **Tab**.
4. Repeat step 3 until you have added all the items to the drop-down menu.
5. Press **Enter** to save the last drop-down menu item.

Adding a Cascading Menu to an Item in a Drop-down Menu

After you have created the first menu bar item, you can start building cascading menus.

1. In the Tree Menu view, select the item in a drop-down menu for which you want to create a cascading menu.
2. Select **Insert Submenu Item** from the pop-up menu.

PowerBuilder displays an empty box.

3. Type the text for the menu item. Press **Tab**.
4. Repeat step 3 until you have added all the items on the cascading menu.
5. Press **Enter** to save the last cascading menu item.

Adding an Item to the End of a Menu

You can add items to the end of menus.

1. Select any item on the menu.
2. Select **Insert Menu Item At End** from the pop-up menu.
3. In the empty box, type the text for the new menu item. Press **Enter**.

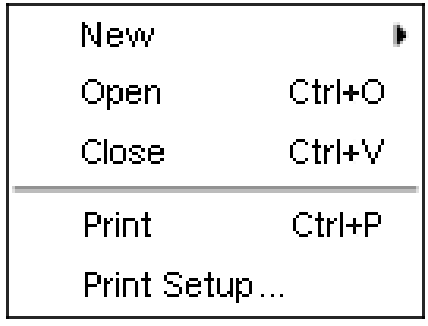
Inserting an Item in an Existing Menu

You can insert an item in a menu.

1. Select the item that should follow the new menu item.
2. Select **Insert Menu Item** from the pop-up menu.
3. In the empty box, enter the text for the menu item. Press **Enter**.

Creating Separation Lines in Menus

Use separation lines to separate groups of related menu items.



1. Insert a new menu item where you want the separation line to display.
2. Type a single dash (-) as the menu item text and press **Enter**.
   The separation line appears.

Duplicating Menu Items

Creating duplicates of existing menu items can save time.

A duplicate menu item has the same properties and script as the original menu item. For example, you might be able to modify a script slightly to make it work for a duplicate menu item.

1. Select the menu item or the submenu item to duplicate.
2. Do one of:

Action	Result
Select Duplicate from the pop-up menu	The duplicate item is added at the same level of the menu, after the original selected item.
Press and hold the Ctrl key while you drag and drop the item	The new menu item appears where you drop it.
Select Edit > Copy. Select the menu item you want the new item to follow, then select Edit > Paste.	The new menu item appears following the selected menu item.

   The text of the duplicate menu item is the same as the original but the name is unique. A copy of a menu item has the same properties and scripts as the original menu item.
3. Change the text of the duplicate menu item.
4. Modify the properties and script associated with the duplicate item as needed.

Changing Menu Item Text

You may need to change the text of a menu item.

1. Do one of the following:
   - In the Tree Menu view, click the item to select it, then click it again.
   - Select the item, then select **Edit Menu Text** from the pop-up menu.
   - Select the item and open the Name category in the Properties view.
2. Type the new text for the menu item in the box in the Tree Menu view or in the Text box in the Properties view.

Navigating in the Menu

Use the mouse or the keyboard to select items in the Tree Menu or WYSIWYG view.

As you work in a menu, move to another menu item by selecting it. You can also use the arrow keys on the keyboard to navigate.

Moving Menu Items

Use drag and drop to change the order of items in the menu bar or in a drop-down or cascading menu.

You can drag items to another location on the same level in a menu structure, or to another level. For example, you can drag an item in the menu bar to a drop-down menu or an item in a cascading menu to the menu bar.

Note: You can use drag and drop only in the Tree Menu view. The WYSIWYG view is primarily for previewing the runtime appearance of the menus.

1. Select the item.
2. Do one of:

Action	Result
Press and hold the left mouse button and drag the item to a new location. Release the mouse button to drop the item.	The menu item appears in the new location.
Select Edit > Cut, then select the menu item you want it to follow and select Edit > Paste.	The menu item appears after the selected menu item.

Saving the Menu

You can save the menu you are working on at any time. When you save a menu, PowerBuilder saves the compiled menu items and scripts in the library you specify.

The menu name can be any valid PowerBuilder identifier. See the *PowerScript Reference*.

A common convention is to use **m_** as a standard prefix, plus a suffix that helps you identify the particular menu. For example, you might name a menu used in a sales application **m_sales**.

1. Select **File > Save** from the menu bar.

   If you have previously saved the menu, PowerBuilder saves the new version in the same library and returns you to the Menu painter. If you have not previously saved the menu, PowerBuilder displays the Save Menu dialog box.
2. If you are saving a new menu, enter its name.
3. Add comments to describe the menu.
   These comments appear in the Select Menu dialog box and in the Library painter. Sybase recommends that you use comments to identify the menu's purpose.
4. Specify the library in which to save the menu and click **OK**.

Deleting Menu Items

You can delete menu items.

1. Select the menu item to delete.
2. Select **Delete** from the pop-up menu.

Menu Item Appearance and Behavior in PowerBuilder .NET

By setting menu properties, you can customize the display of menus in applications that you create with PowerBuilder.

You use the Menu painter to change the appearance and behavior of your menu and menu items by choosing different settings in the Properties view categories. For a list of all menu item properties, see *Objects and Controls*.

General Properties for Menu Items

There are properties you can set when you select a menu item and then select the Menu Item category in the Properties view. For information on using menu properties, see the PowerBuilder Users Guide.

Accelerator and Shortcut Keys in Menus

Accelerator and shortcut keys in menus enable your users to access menu items quickly.

Every menu item should have an accelerator key, also called a mnemonic access key, which allows users to select the item from the keyboard by pressing Alt+*key* when the menu is displayed. Accelerator keys display with an underline in the menu item text. See *Accelerator Characters in Control Labels*.

You can also define shortcut keys, which are combinations of keys that a user can press to select a menu item, whether or not the menu is displayed. See *Assigning a Shortcut Key to Menu Items* on page 44.

Sybase recommends that you adopt conventions for using accelerator and shortcut keys in your applications. All menu items should have accelerator keys, and commonly used menu items should have shortcut keys.

Assigning a Shortcut Key to Menu Items

You can assign shortcut keys to menu items.

If you specify the same shortcut for more than one MenuItem, the command that occurs later in the menu hierarchy takes precedence.

Some shortcut key combinations, such as Ctrl+C, Ctrl+V, and Ctrl+X, are commonly used by many applications. Avoid using these combinations when you assign shortcut keys for your application.

1. Select the menu item to which to assign a shortcut key.
2. In the Menu Item category in the Properties view, select a key from the ShortcutKey drop-down list.
3. Change ShortcutAlt, ShortcutCtrl, or Shortcut Shift to True to create a key combination. PowerBuilder displays the shortcut key next to the menu item name.

Accessibility of Menu Items

Use the Visible and Enabled properties to determine what menu items are available to users.

Using the Visible Property

Use the Visible property to determine whether menu items are hidden by default.

If the Visible property of a menu item is set to true, the menu item appears in the Menu painter's Tree and WYSIWYG views. If you want a menu item to be initially invisible, set the Visible property to false in the menu item's Properties view.

To display a control at runtime, set the Visible property to true:

```
menuitem.Visible=TRUE
```

To display hidden controls in the WYSIWYG view, select **Edit > Show Invisibles**.
In the WYSIWYG view, the menu item appears.

Using the Enabled Property

Enabled menu items can be selected; disabled menu items cannot.

If the Enabled property is set to true in the menu item's Property view, the menu item is active.

If you want a control to appear but be inactive, set the Enabled property to false. For example, a menu item might be active only after the user has selected a particular option. In this case, initially display the menu item as disabled, so that it appears dimmed. In the script for the menu item, enable the menu item when the user selects the appropriate option.

```
menuitem.Enabled=TRUE
```

MenuFocusable Property

You can open a menu using a clickable object. The MenuFocusable property, which applies only to Menu controls, determines whether the focus stays on a clicked item or the menu.

Syntax

PowerBuilder dot notation:

```
menuname.MenuFocusable
```

Modify and Describe argument:

```
menuname.MenuFocusable = {value}
```

Parameter	Description
value	Specifies the default behavior of the menu's focus. • true - Focus shifts from the button to the menu so that the user can use the keyboard arrow keys to navigate the menu. The button loses focus, triggering the LoseFocus event. • false - Focus remains on the button when the item is clicked.

Usage

In the Menu painter, Properties view, Appearance group, select the menu control and set the value.

You can also set the property in a script.

```
m_genapp_sheet.MenuFocusable = true
```

Toolbars for Applications

To make your application easier to use, add toolbars with buttons as shortcuts for menu items.

Adding Toolbars to a Window

Create toolbars for windows.

When you define an item in the Menu painter for a menu that will be associated with a frame window, sheet, or a main window, specify that you want to include the item in the toolbar using a specific picture. At runtime, PowerBuilder automatically generates a toolbar for the window containing the menu.

1. In the Menu painter, specify the display characteristics of the menu items to include on the toolbar.

   See *Toolbar item display characteristics* in the *PowerBuilder Users Guide*.
2. (Optional) In the Menu painter, specify drop-down toolbars for menu items.
3. In the Window painter, associate the menu with the window and turn on the display of the toolbar. (See *Adding a Menu Bar to a Window* on page 49.)
4. (Optional) In the Window painter, in the PBToolbar property category, specify other properties, such as the size and location of a floating toolbar.

   For information on setting the toolbar properties, see the *PowerBuilder Users Guide*.

Selecting a Toolbar Style

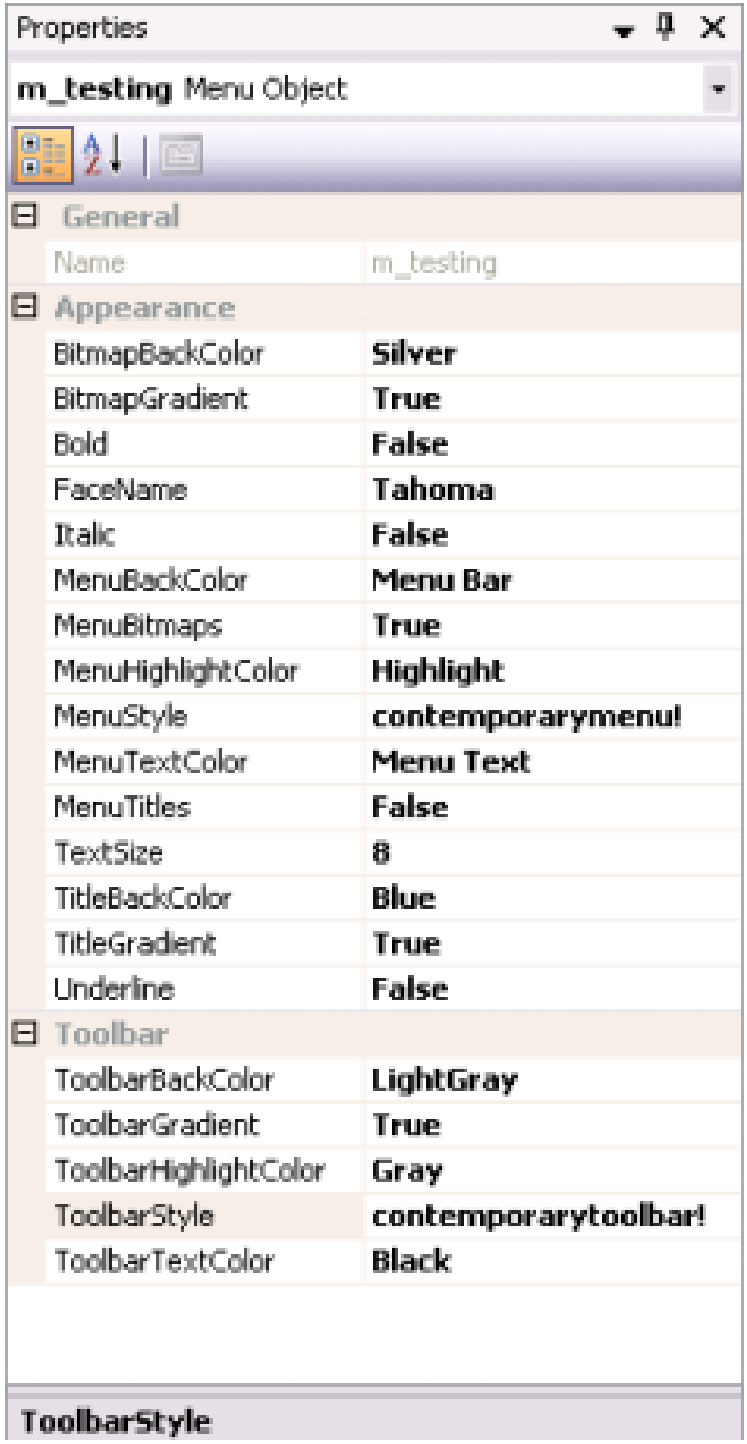
Select a toolbar style in the Toolbar category of the Properties view for the top-level menu object in the Menu painter.

Toolbars can have a contemporary or traditional style. Imported and migrated toolbars maintain their style.

Toolbar style	Description
Contemporary	A 3D-style toolbar similar to Microsoft Office 2003 and Visual Studio 2005
Traditional	A more traditional and older toolbar style

By default, new toolbars use the contemporary toolbar style.

1. Select the top-level menu object.
2. In the Toolbar category of the Properties view, select the toolbar style you want: **traditionaltoolbar!** or **contemporarytoolbar!**


Properties
m_testing Menu Object
General
Name
m_testing
Appearance
BitmapBackColor
Silver
BitmapGradient
True
Bold
False
FaceName
Tahoma
Italic
False
MenuBackColor
Menu Bar
MenuBitmaps
True
MenuHighlightColor
Highlight
MenuStyle
contemporarymenu!
MenuTextColor
Menu Text
MenuTitles
False
TextSize
8
TitleBackColor
Blue
TitleGradient
True
Underline
False
Toolbar
ToolbarBackColor
LightGray
ToolbarGradient
True
ToolbarHighlightColor
Gray
ToolbarStyle
contemporarytoolbar!
ToolbarTextColor
Black
ToolbarStyle

If you select **traditionaltoolbar!** you cannot then select other toolbar style properties. If you select **contemporarytoolbar!** style, you can customize the display properties for that style and all menu items with associated toolbar buttons in the current menu.

Unless you are using the traditional toolbar style for the current menu object, you can set the ToolbarAnimation property in the Properties view for each menu item. If you do not select an image for the ToolbarItemName property of a menu item, the selection you make for the ToolbarAnimation property is ignored.

Writing Scripts for Menu Items

Scripts specify what happens when users select a menu item.

Note: For information on writing scripts for menu item events, see the *PowerBuilder Users Guide*.

To write a script for a menu item:

Select Script from the menu item's pop-up menu.
You see the Script view for the clicked event.

Using Inheritance to Build a Menu

Build a menu that inherits its style, events, functions, structures, variables, and scripts from an existing menu to save coding time. All you have to do is modify the descendant object to meet the requirements of the current situation.

1. In the Solution Explorer, select the menu from which you want to inherit.
2. Right-click and select **Inherit from**.
   The new descendant menu opens in the Menu painter.
3. Make the necessary changes to the descedant menu.
   See *Using the inherited information* in the *PowerBuilder Users Guide*.
4. Save the menu under a new name.

Menus in Your PowerBuilder .NET Applications

You can use menus as the window menu bar or as pop-up menus.

Adding a Menu Bar to a Window

To include a menu bar in a window, use the Window painter to associate a menu with the window.

1. In the PowerBar, click **Open**, then select the window with which you want to associate the menu. Open the window.
2. In the Properties view, PBGeneral category, MenuName text box, do one of:
   - Enter the name of the menu.
   - Click the ellipsis button and select the menu from the Select Menu dialog box.

3. Click **Save** to associate the selected menu with the window.

Add and Change Menu Bars Using Scripts

You can use PowerScript to assign and change menu bars in Windows.

Identifying menu items in window scripts

Reference menu items in scripts in windows and controls using this syntax:

```
menu.menu item
```

You must always fully qualify the menu item with the name of the menu.

When referring to a menu item in a drop-down or cascading menu, specify each menu item on the path to the menu item you are referencing, using periods to separate the names.

For example, to refer to the Enabled property of menu item **m_open**, which is under the menu bar item **m_file** in the menu saved in the library as **m_menu**, use:

```
m_menu.m_file.m_open.Enabled
```

Changing a window's menu at runtime

Use the **ChangeMenu** function in a script to change the menu associated with a window at runtime.

Displaying Pop-up Menus in Windows

Display a pop-up menu in a window using the PopMenu function to identify the menu and location.

If the menu is currently associated with the window, you can simply call the PopMenu function.

For example, assuming that **m_appl** is associated with the window, this statement in a CommandButton script displays **m_appl.m_help** as a pop-up menu at the current pointer position:

```
m_appl.m_help.PopMenu(PointerX(), PointerY())
```

If the menu is not already associated with the window, first create an instance of the menu.

For example, assuming that m_new is not already associated with the window containing the script, create an instance of the menu **m_new** and position **mymenu.m_file** at the pointer location:

```
m_new  mymenu
mymenu = create m_new
mymenu.m_file.PopMenu(PointerX(), PointerY())
```

User Objects

Applications often have features in common. If you find you are repeatedly using the same application feature, define a user object in the User Object painter; you can then use the object in multiple places.

For example, you might often reuse features like the following:

- A processing package that calculates commissions or performs statistical analysis
- A Close button that performs a certain set of operations and then closes the window
- DataWindow controls that perform standard error checking
- A list that includes all departments
- A predefined file viewer that you plug into a window

Creating and using class and visual user objects are discussed in the *PowerBuilder Users Guide*. PowerBuilder .NET also allows you to use customized or third-party WPF controls. See *Third-Party and Custom Controls* on page 51.

Third-Party and Custom Controls

You can use custom and third-party WPF controls in PowerBuilder .NET, in WPF windows and DataWindow object. Using custom and third-party controls allows you to extend the software system to support a wider range of applications.

Note: WPF controls must be descendants of the FrameworkElement class. You can inherit visual third-party controls that are written in code (such as C# or Visual Basic .NET) and not created using XAML; XAML does not support visual object inheritance.

WPF controls in windows

If the custom control defines public events, they are dynamically added to the event list of the custom control. You can define your own event handler for any event of a WPF control.

WPF controls in DataWindow objects

In DataWindow objects, insert custom and third-party WPF controls using the CustomControl component.

A third-party control encapsulated in the CustomControl may have different capabilities of presenting and processing data. If the third-party control can present data (for example, a TextBlock), CustomControl feeds data to this third-party control; if the third-party control can modify data (such as a TextBox), CustomControl retrieves modified data from the third-party control.

XAML and third-party or custom controls

When working with third-party or custom controls:

- Add the third-party or custom WPF controls defined in an external XAML file before pasting the XAML file into the XAML view.

- Add controls from the Toolbox to third-party or custom control containers. Do not use the XAML view.
- Make sure there are no duplicate control names before pasting the code into the XAML view and saving the window. If there are duplicates, PowerBuilder .NET prompts you to change them before saving the window.

Adding WPF Controls to the Toolbox

You can specify the components that appear in the Toolbox.

The contents of the Toolbox are related to the object that has focus. Be sure that the appropriate object is selected before adding a WPF control to the Toolbox.

1. In the Toolbox, select the group to which you want to add the WPF control.
2. Right-click in the Toolbox group and select **Choose Items**.
3. Select the **WPF Components** tab.
4. Select a standard WPF control in the tab, or use **Browse** to select your own WPF control.
5. Click **OK**.

Adding Third-Party and Custom WPF Controls to Windows

You can add third-party and custom WPF controls to objects.

1. If you have not already done so, add the WPF control to the Toolbox.
   See *Adding WPF Controls to the Toolbox* on page 52.
2. In the Toolbox, select the control and then click in the window where you want to place it.
   The control is added to the window, and the XAML code for the control is added to the window code.

Adding Third-Party and Custom Controls to DataWindow Objects

Add third-party and custom controls to DataWindow objects using the Custom Control.

1. In the Toolbox, in the DataWindow Controls group, select **Custom Control**.
2. Click the location in the DataWindow where you want to place the custom control. The Select Custom Control dialog box opens.
3. Either:
   - Click the ellipsis button next to the From File field and navigate to the XAML file that defines the custom control.
   - Click the ellipsis button next to the From Assembly field and select the control.

   The XAML definition populates the Preview/Edit Xaml field.
4. Click **OK**.

Source Control in PowerBuilder .NET

PowerBuilder .NET leverages Visual Studio isolated shell features to provide direct connections to external source control systems.

Connections to source code control systems are very different in PowerBuilder .NET and PowerBuilder Classic.

In PowerBuilder .NET, set up source control connections from the Options dialog box rather than the workspace properties dialog box. The source control system you use must already be installed on your computer before you can select it on the Plug-in Selection page of the Options dialog box. The Options dialog box also has separate pages for configuring environment and plug-in settings for your source control system.

Source code functionality in PowerBuilder .NET depends on Microsoft hotfix VS90SP1-KB975247. The PowerBuilder setup program typically automatically this hotfix if it does not detect its presence on the computer where you are installing PowerBuilder. However, the hotfix is also available in the Support folder on the PowerBuilder setup CD. You can install it at any time before connecting to your source control system in PowerBuilder .NET.

Note: Use the F1 key in the Options dialog box to access Microsoft help for the settings on any of the dialog box pages, including those for source control.

The PBNative source code utility that is installed with PowerBuilder Classic does not work with PowerBuilder .NET.

Adding Solutions to Source Control

Configure the source control settings in the Options dialog box before you add a solution, and the objects in a solution, to source control.

You can add only one solution at a time to source control. When you add a solution, all the objects in the solution are automatically added.

Windows that you add to source control are saved as two separate files, an SRW file and a XAML file. These files are automatically checked in and checked out together when you perform these operations on window objects. However, you must save a window object before you can compare its XAML file with the file on the source control server. If you use an unsaved (dirty) XAML file for the comparison, you are likely to get incorrect results.

1. If source control settings are not already configured for your current solution, select **Tools > Options** to configure these settings, then close the Options dialog box.
   For Microsoft Visual Studio help on configuring source control settings, select any of the options under the Source Control node in the left pane of the Options dialog box, then press F1.
2. In the PowerBuilder .NET Solution Explorer, right-click the solution and select **Add Solution to Source Control**.

A dialog box opens for the source control system you selected in the Options dialog box.

3. Log in to the source control system and add the solution to source control in the same way you would add it from the manager of your source control system.

After you add a solution to source control, you can select context menu items in the Solution Explorer to perform all operations permitted by your source control system, on the solution itself, or on the objects that the solution contains. Adding a solution also enables the Pending Checkins window, which you can open from the View menu or from the context menus on targets in the Solution Explorer.

Add-ins in the PowerBuilder .NET Environment

Add-ins allow you to extend the functionality of the PowerBuilder .NET environment.

Add-ins can be written in any .NET language, including PowerBuilder .NET, and must follow Microsoft Visual Studio 2008 guidelines. The configuration file for each add-in must include PowerBuilder .NET as an allowed host.

These lines in the configuration file enable you to use an add-in in the PowerBuilder .NET IDE:

```
<HostApplication>
   <Name>PowerBuilder .NET</Name>
   <Version>12.0</Version>
</HostApplication>
```

For information about creating add-ins, see *http://msdn.microsoft.com/en-us/library/5abkeks7.aspx* and *http://www.c-sharpcorner.com/UploadFile/mgold/AddIns11292005015631AM/AddIns.aspx*. Microsoft help for enabling add-ins in an isolated shell environment is available from the Add-in/Macros Security page under the Environment node of the PowerBuilder .NET Options dialog box.

PowerBuilder .NET Targets and Projects

You work with one or more targets and projects in a PowerBuilder .NET workspace.

Just as in PowerBuilder Classic, each target specifies a type of of application. It can also contain documentation and resource files. Each target can have one or more associated projects, specifying how the executable version of the application is to be generated. You can configure separate projects for different runtime platforms and build types. However, there are some important differences. For example, PowerBuilder .NET targets have `PBTX` extensions, and you cannot save them to the root directory of a computer drive.

You can use these target and project types in PowerBuilder .NET.

Target	Associated Projects
WPF Window Application	WPF Window Application, WCF Client Proxy, or REST Client Proxy
PB Assembly	WCF Client Proxy, REST Client Proxy, or PB Assembly
.NET Assembly	WCF Client Proxy, REST Client Proxy, or .NET Assembly
WCF Service	WCF Client Proxy, REST Client Proxy, or WCF Service

Creating a WPF Window Target

Use the WPF Window Application target wizard to create a WPF Window Application target.

You can create a new WPF Window Application target, or convert a PowerBuilder Classic Win32 or Windows Forms target.

1. Select **File > New** from the menu or click the **New** button in the toolbar.
2. Expand the Target folder. Select the **WPF Window Application** target and click **Next**.
3. On the Create the application page, select one of these options and click **Next**:
   - Create a new application and target
   - Convert an existing Win32 or Windows Forms target to WPF

   Note: If the existing target was created in an earlier version of PowerBuilder, upgrade it in PowerBuilder Classic by following the guidelines in the section *Users Guide>Working with Libraries>Migrating targets* and the current *Release Bulletin>Migration Information*. You can then convert the upgraded version in PowerBuilder .NET using this procedure.

Migration to PowerBuilder .NET automatically makes some changes that might also require manual refactoring of application scripts. For example, hyphens are not allowed in identifiers in WPF applications, so by default the migration changes hyphens to the word "dash". If these identifiers are referenced in a script, you must modify the references. You must also refactor applications that rely on API calls using control handles before migrating them to PowerBuilder .NET.

4. (Optional) If you are creating the target from scratch, you can choose to generate an initial Window object on the "Specify an object to create" page.
5. Follow the remaining instructions in the WPF Window Appliation Target wizard and click **Finish**.

   You can skip wizard pages and create a target with just the values you have provided by clicking **Finish** any time it is enabled. You can add or change values at a later time in the Properties Pages dialog box for the target and in the Project painter for target projects.

The New WPF Target wizard creates a target, as well as a project object, which lets you deploy the application.

Next

After you create a WPF target, use the Project painter to:

- Modify the settings of the associated project
- Deploy the project
- Run the project
- Debug the project

Creating and Building a WPF Project

Use the WPF Project painter to define and test a WPF Window Application project.

1. Create a WPF project, using one of these methods:

   - Use the WPF Window Application Target wizard to create a target and a project.
   - Use the WPF Window Application Project wizard to create a project.

   The project is generated with its basic properties defined.
2. If you want change or add project properties, double-click the WPF project object In the Solution Explorer to open it in the Project painter.
3. In the General tab, specify the name and location of the executable file to be generated, and the optional build options.
4. In the Assemblies page, specify the output to be built from each PBL in the target library list:

To generate an assembly containing the current row's PBL	Choose or enter an assembly name. If you enter a new name instead of choosing an existing output assembly name, the new name is available in all other rows.
To build more than one PBL to the same output assembly	Specify the same assembly in multiple rows.
To build a PBL to the application executable instead of an assembly	Leave the assembly name blank.

For example, for a library named `pbexample.pbl`, you might choose the `pbexample.dll` assembly name.

5. Specify an assembly for global variables.
6. Optionally, specify whether to create a resource-only assembly, and its name.

Note: Before this feature was implemented, PowerBuilder automatically built a resource-only assembly containing all DataWindow, query, and pipeline objects. Now, you can specify the name of the resource-only assembly, or omit it from the build entirely.

7. In the Dependencies page, examine a summary list of output assemblies that you defined in the Assemblies page.

   This read-only page shows a tree view of the output assemblies. The structure indicates dependencies among assemblies and the order in which they will be built.
8. Click **Check Dependencies** in the WPF Project Painter Toolbar (or, in the context menu of the project object in the Solution Explorer). Any time you update the list of output assemblies, or refactor your scripts in such a way that affects dependencies, you should invoke the Check Dependencies command.
9. To generate and deploy the output assemblies, click **Full Build Project** or **Incremental Build Project** in the WPF Project Painter toolbar.
10. To run or debug your project after deploying it, click **Run** or **Debug** in the WPF Project Painter toolbar.

WPF Window Application Target and Project Properties

The WPF Window Application target and project wizards define basic properties for a WPF Window application. The Project painter lets you define additional properties for building and deploying the project. It also lets you check dependencies between output assemblies.

Basic WPF Window Application Properties

Property	Description
Application name	Name of the optional application object to be created. By default, the application name is als o used for the library and target. Example: `wpfapp`
Library	Name and location of the PowerBuilder library to be created.
Target	Name and location of the target to be created. By default, this includes the current solution path, the application name, and a PBTX extension. Example: `C:\Documents and Settings\janedoe\My Documents\pbnet\wpfapp.pbtx`
New location and target file	For existing targets only. By default, the wizard specifies a target with the same name as the existing target but with a `PBTX` extension. The default location is a `wpf` subfolder within the existing target's folder. Example: `C:\pbapps\wpf\myapp.pbtx` If you prefer a different target, change the default value.
Project name	Name of the project object to be created. Example: `p_wpfapp_wpf`
Product name	Product name to be associated with this target. Example: `My WPF Application`
Product executable filename	Name of the runnable file for this target.

Property	Description
Product version	Major, Minor, Build, and Revision version numbers to be associated with this target.
Resource files and directories	Identify resource files and folders to add to the target. (Optional.) Use the browser buttons to select individual files, directories, or files.
Publish as smart client application?	Select this option to configure smart client publishing properties in subsequent wizard pages. For example, select this option to publish the application to an FTP site.
Application running mode (for smart client only)	Choose one of these options: • **Yes –** the application can be run either locally (on the user's computer) or from a remote site. • **No –** the application will be installed on a Web site or shared path; not locally.
Install options (for smart client only)	Choose one of these options: • **From web site –** specify the Web site URL. Example: http://localhost/wpfapp • **From shared path –** specify the path where a user can find and install the application. • **From CD-ROM or DVD-ROM –** choose this option if the application will be provided on one of these media.
Update options (for smart client only)	Choose one of these options to specify how the application checks for the latest version and updates: • Never check for updates • Check for updates before application starts • Check for updates after application starts, and specify the update frequency

Property	Description
Update frequency (for smart client only)	Choose one of these options: • Every time application starts • At least *n interval* from last update. Enter a positive number for the frequency *n*. Choose days, hours, or weeks as the *interval*.

WPF Window Application Project Painter: General Tab

Property	Description
Executable	Name of the runnable file for this target. The file is created in the output directory. Default: application name and an `.EXE` extension.
Output Path	The full path where the executable is deployed.
Build type	Choose either **Debug** or **Release**. The wizard displays the build subfolder for the selected option.
Suppress Unsupported Feature warnings	Suppress the compiler warnings that PowerBuilder .NET normally displays in the Output window about features in your application that are unsupported in .NET. **Note:** Unsupported Feature warnings do not have IDs. See the next property to suppress errors by ID.
Suppress the following warnings (comma-separated)	Enter a comma-separated list of the IDs of PowerBuilder compiler warnings to suppress. Example: `C0003,C0016`
Publish as smart client application	Indicates that the application is configured as a smart client.
Enable DEBUG symbol	Choose this option if your application contains code with the DEBUG conditional compilation symbol, and you want this code to run.

WPF Window Application Project Painter: Assemblies Tab

This tab displays the target's WPF library list, and lets you specify what assemblies to build from the target's PBLs.

Property	Description
PBL and Output Assembly	The left column lists the PBLs in the target library list. The right column is a selection list of PBL names with `.DLL` appended. A valid assembly name indicates that an output assembly is to be built that includes the library on that row. If the output assembly field is blank, the PBL is built to the application's executable file instead.
Globals and Output Assembly	The Output Assembly field specifies the globals assembly DLL file to generate. It can be one of the output assemblies in the PBL list, or a different one.
Create resource-only assembly for executable	Indicates that resource objects from PBLs that are part of the application executable are to be built to a resource-only assembly, and specifies the assembly name.

Output Assembly Name Restrictions:

- An output assembly cannot have the same name as the executable. For example, if the executable name is `examples`, then no assembly can be named `examples.dll`.
- These characters are not allowed: (| " < > -)

If you enter an assembly name that does not conform to these restrictions, the Output Assembly control displays a red border and a tooltip error indicating the error.

WPF Window Application Project Painter: Dependencies Tab

This tab presents a tree view of output assemblies. It indicates dependencies among the assemblies, and their build order.

The view is read-only.

WPF Window Application Project Painter: Remaining Tabs

The remaining tabs in the Project painter are similar to tabs of the same name in the .NET Assembly Project painter.

Dependency Checker

The dependency checker examines the target PBLs and output assemblies in a WPF Window Application project, and displays information about the dependencies among the assemblies.

For any group of PBLs built to an output assembly, the dependency checker might detect single dependencies on other assemblies, circular dependencies, or no dependencies. A group of PBLs might be dependent, in turn, on a PBL that is built to the executable file.

The Dependencies tab of the WPF Application Project painter contains a list of output assemblies and their dependencies, in build order. If you change output assemblies in the Assemblies page, the dependency checker displays a note on the Dependencies tab, advising you to run **Check Dependencies** to update dependencies. This command is available in

- The WPF Project Painter toolbar when the WPF Window Project painter is open.
- The **Design** menu when the WPF Window Project painter is open.
- The context menu of a WPF project object in the Solution Explorer.

A project build uses the current build order. Therefore, if you launch a build after changes that affect dependencies, the build can fail with dependency errors. Dependencies can change because you changed assemblies in the project, or through code refactoring. You might try to resolve dependency errors by refactoring, or you can invoke the dependency checker to update the build order.

The dependency checker displays any compiler errors in the Output window and Error List. If there are no compiler errors, the Assembly Dependencies window opens, where you can examine dependencies in detail.

The top of the Assembly Dependencies window has a read-only, collapsible key view that explains the format of the information below. The main output area of the window lets you navigate dependencies in detail. It provides expander controls to show or hide any part of the dependency tree, and highlights any dependency errors. You can also double-click a script reference in the tree to navigate to its source.

The Output window notes the progress of dependency checking. For example:

```
-------- Checking dependencies using project settings
--------
Checking project settings...
Using Project p_examples_wpf
Dependency check complete.
Error: output assembly pbexamw1.dll has a dependency on
examples.exe
Error: There is a circular dependency between pbexamw1.dll and
pbexamuo.dll
Time elapsed for dependency check: 00:00:08.3593750
-------- Dependency check finished ---------
```

These output messages indicate the status of the dependency check:

- **All dependencies are OK. –** no circular or problematic dependencies are found.
- **Error: There is a circular dependency between *assembly* and *assembly*. –** circular dependencies are found.
- **Error: output assembly *assembly* has a dependency on *exe*. –** an output assembly is dependent on the executable

WPF Window Application Runtime Requirements

At runtime, the WPF applications that you generate from PowerBuilder .NET execute using the .NET Common Language Runtime (CLR).

For deployment of WPF Window Application targets, production servers or runtime computers must have:

- Windows XP SP3, Windows Vista SP2, Windows 2008 SP1, or Windows 7 operating system
- .NET Framework 4.0 or later
- The Microsoft Visual C++ runtime libraries `msvcp100.dll`, `msvcr100.dll`, `msvcr71.dll`, `msvcp71.dll`, and the Microsoft .NET Active Template Library (ATL) module, `atl71.dll`

Executing deployed WPF applications require additional PowerBuilder files and assemblies. When you select the PowerBuilder .NET Components option in the PowerBuilder Runtime Packager and click Create, the Runtime Packager generates an MSI or MSM package that contains required PowerBuilder files and supporting libraries for all targets. By default, the generated package is named `PBNETRT125.MSI`.

You can extract the generated MSI or MSM file to install the included files and assemblies to their required locations on a production server or target computer. The package that you generate from the Runtime Packager when the PowerBuilder .NET Components option is selected includes these files and assemblies:

- PowerBuilder runtime dynamic link libraries in the system path: `pbshr125.dll`, `pbrth125.dll`, and `pbdwm125.dll`.
- PowerBuilder .NET assemblies in the global assembly cache (GAC): `Sybase.PowerBuilder.ADO.dll`, `Sybase.PowerBuilder.Common.dll`, `Sybase.PowerBuilder.Core.dll`, `Sybase.PowerBuilder.DataSource.Db.dll`, `Sybase.PowerBuilder.DataSource.Db2.dll`, `Sybase.PowerBuilder.DataSource.dll`, `Sybase.PowerBuilder.DataSource.Sharing.dll`, `Sybase.PowerBuilder.DataSource.Trace.dll`, `Sybase.PowerBuilder.DataSource.WS.dll`, `Sybase.PowerBuilder.DataWindow.Interop.dll`, `Sybase.PowerBuilder.DataWindow.Web.dll`, `Sybase.PowerBuilder.DataWindow.Win.dll`,

Sybase.PowerBuilder.Db.dll, Sybase.PowerBuilder.DBExt.dll,
Sybase.PowerBuilder.EditMask.Win.dll,
Sybase.PowerBuilder.EditMask.Interop.dll,
Sybase.PowerBuilder.Graph.Core.dll,
Sybase.PowerBuilder.Graph.Interop.dll,
Sybase.PowerBuilder.Graph.Web.dll,
Sybase.PowerBuilder.Graph.Win.dll,
Sybase.PowerBuilder.Interop.dll,
Sybase.PowerBuilder.LibraryManager.dll,
Sybase.PowerBuilder.RTC.Interop.dll,
Sybase.PowerBuilder.RTC.Win.dll,
Sybase.PowerBuilder.Utility.dll,
Sybase.PowerBuilder.WCF.WSDL.dll,
Sybase.PowerBuilder.WCF.WSDLRemoteLoader.dll,
Sybase.PowerBuilder.WCF.Runtime.dll,
Sybase.PowerBuilder.Web.dll,
Sybase.PowerBuilder.Web.WebService.dll,
Sybase.Powerbuilder.WebService.Runtime.dll,
Sybase.PowerBuilder.WebService.RuntimeRemoteLoader.dll,
Sybase.PowerBuilder.Win.dll,
Sybase.PowerBuilder.WinWebDataWindowCommon.dll,
Sybase.PowerBuilder.WPF.Controls.dll,
Sybase.PowerBuilder.WPF.Controls.Skins.dll, and
Sybase.PowerBuilder.WPF.dll.

- DataWindow .NET assemblies in the GAC:
  Sybase.DataWindow.ADO.dll, Sybase.DataWindow.Common.dll,
  Sybase.DataWindow.Core.dll, Sybase.DataWindow.Db.dll,
  Sybase.DataWindow.DbExt.dll, Sybase.DataWindow.Interop.dll,
  Sybase.DataWindow.Shared.dll,
  Sybase.DataWindow.WebService.Runtime.dll,
  Sybase.DataWindow.WebService.RuntimeRemoteLoader.dll,
  Sybase.DataWindow.WebService.WSDL.dll, and
  Sybase.DataWindow.Wpf.dll.

For a complete list of files deployed by the Runtime Packager, see *Application Techniques > Deploying Applications and Components > PowerBuilder Runtime Packager* in the PowerBuilder collection on the Sybase Product Manuals Web site.

Creating a PB Assembly Target

Create PB Assembly targets to deploy one or more PBLs into an assembly for use in other PowerBuilder .NET targets.

1. Select **File > New**, or click **New** in from the toolbar.
2. From the Target category, select **PB Assembly** and then click **Next**.
3. Follow the remaining instructions in the PB Assembly wizard and click **Finish**.

   If you prefer to enter values for the PB Assembly target in the project painter, click **Finish** at any time it is enabled.

PB Assembly Target

Use the PB Assembly target to build a set of one or more PBLs into an assembly, which can be referenced by another target.

The PB Assembly target is similar to the runtime library (.PBD) in PowerBuilder Classic. The application object and all the global variables (system and user-defined) defined in the PB Assembly target are used only within the PB Assembly target itself; you cannot use them across targets.

The PB Assembly target:

- Can contain all PowerBuilder object types
- Exposes all PowerBuilder objects when a referenced PB Assembly is expanded in the Solution Explorer
- Outputs a DLL assembly
- Need not be built by any referencing targets once it has been built
- Is used directly in PowerBuilder (unlike the .NET Assembly target)

The global objects in the PB Assembly are not instantiated. They require a reference to the instantiated variables from the application. Pass the global variable instantiated in the application into the PB Assembly and assign it to the global variable defined in the PB Assembly target.

PB Assembly Target and Project Properties

Set properties for PB Assembly targets and projects in the PB Assembly wizard, and for PB Assembly projects in the Project painter.

Table 1. PB Assembly Target Properties

Wizard field	Description
Project name	Name of the project object.
Library	Name of the library directory. By default, this includes the current solution path and takes the name you enter for the project object with a PBL extension.
Target	Name of the target. By default, this includes the current solution path and takes the name you enter for the project object with a PBTX extension.
Library search path	Lets you add PBL directories to the search path for the new target.
Assembly file name	Name of the assembly the target creates when you build the target. By default, the assembly file name takes the project name with a DLL suffix.
Resource files	List of resource files, or directories containing resource files, to deploy with the project. Use the buttons to add files or directories to the list box. When you add a directory, a check box appears next to the directory path in the list box. You can select this check box to add resources from subdirectories of the listed directory path. Select a file or directory in the list and click **Delete** to remove that file or directory from the list.
Win32 dynamic library file list	Specifies any Win32 DLLs to include with your project. Click **Add** 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 for the PB Assembly project. Copy this MSI file to client computers, then double-click the files to install the PB Assembly on those computers.

Modify target properties in the Properties dialog box for the target. Enter and modify properties for projects in the Project painter.

Creating a .NET Assembly Target

Create .NET Assembly targets from scratch or from an existing target that contains at least one nonvisual custom class object. You can also migrate .NET Assembly targets from PowerBuilder Classic to PowerBuilder .NET.

Use the .NET Assembly wizard in the PowerBuilder .NET New dialog box to create a .NET Assembly target.

Note: If you prefer to enter values for the .NET Assembly target in the Project painter, click **Finish** at any time it is enabled.

1. Select **File > New**, or open the New dialog box from the toolbar.
2. From the Target category, select **.NET Assembly** and click **Next**.
3. Specify how to create the .NET assembly:
   - From scratch
   - By converting a PowerBuilder Classic .NET Assembly target
4. Optional: If you are creating the target from scratch, you can choose whether to generate an initial object—Custom Visual, Custom Nonvisual, standard nonvisual objects, or None—on the page Specify an object to create.
5. Follow the remaining instructions in the .NET Assembly wizard and click **Finish**.

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 you can use it to create a .NET assembly.

If you selected 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 at least one NVO that has at least one public method. The public method must be implemented by the NVO or inherited from a parent. The AutoInstantiate property of the NVO must be set to false.

.NET Assembly Target and Project Properties

Set properties for .NET Assembly targets and projects in the .NET Assembly wizard, and for .NET Assembly projects, in the Project painter.

Properties for New Target

For a new .NET Assembly target that you create in a wizard, provide the information described in this table:

Table 2. Basic .NET Assembly Properties

Property	Description
Project name	Name of the project object the wizard creates.
Library	Name of the library directory the wizard creates. By default, this includes the current solution path and takes the name you enter for the project object with a PBL extension.
Target	Name of the target the wizard creates. By default, this includes the current solution path and takes the name you enter for the project object with a PBTX extension.
Library search path	Lets you add PBL directories to the search path for the new target.
Default namespace	Assigns a namespace for all objects in the target for which you do not explicitly specify a different namespace. This namespace is saved in the project created by the wizard, but is not visible in the UI for the objects that you subsequently add to the target and deploy with this project. The default namespace is not available for selection from lists for objects in the target, and can be applied only to those objects by the compiler. Other projects in the same target can have different default namespaces that apply to the same objects.
Assembly file name	Name of the assembly that the wizard creates. By default, the assembly file name takes the namespace name with a DLL suffix.
PowerBuilder objects	An object to be created with the target, or none.

Property	Description
Resource files	List of resource files, or directories containing resource files, that you want to deploy with the project. Use the buttons to add files or directories to the list box. When you add a directory, a check box appears next to the directory path in the list box. You can select this check box to add resources from subdirectories of the listed directory path. Select a file or directory in the list and click **Delete** to remove that file or directory from the list. Resource files that you add are automatically included under the Resources folder for the target in the Solution Explorer.
Win32 dynamic library file list	Specifies any Win32 DLLs you want to include with your project. Click **Add** 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 for the .NET Assembly project. Copy this MSI file to client computers, then double-click the files to install the .NET assembly on those computers.

Properties for New Target Converted from an Existing Target

The New .NET Assembly wizard lets you convert an existing .NET Assembly target that was created earlier in PowerBuilder Classic.

The **Use the library list from an existing target** option lets you select a source target from the current solution only. When you select this option, the wizard prompts you for the same information as when you create a new target, but omits the PowerBuilder object name, description, 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. However, the wizard does present the additional fields in the table below, which are not available when you create a new target:

Table 3. Additional Wizard Fields for Targets Created from Existing Target Library List

Field	Description
Target	A target in the current solution to be copied for the new target. After selecting a source target, specify a new target name, a project name, and a project library, as described in the table listing wizard fields for a new .NET Assembly target.
Choose objects to be deployed	The objects to be deployed with the project. You can expand the library nodes and select check boxes next to each of the objects you want to deploy.
Use .NET nullable types	Maps PowerScript 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.

If you choose **Convert an existing .NET assembly target**, the .NET Assembly target wizard has only two editable fields—one to select the source target for conversion, and the other to assign the location and name of the new .NET Assembly target. The wizard also displays a list of the libraries from the source target that you select.

.NET Assembly Project Painter Properties

The .NET Assembly Project painter lists all properties. You can modify fields that were set in the wizard, and add additional properties that are not available in the wizard.

Table 4. General Tab

Property	Description
Namespace	Name of the runnable file for this target. The file is created in the output directory. Default: application name and an `.EXE` extension.
Assembly file name	
Output Directory	The full path where the executable is deployed.
Setup file name	

Property	Description
Build type	Choose either **Debug** or **Release**. The wizard displays the build subfolder for the selected option.
Suppress Unsupported Feature warnings	Suppress the compiler warnings that PowerBuilder .NET normally displays in the Output window about features in your application that are unsupported in .NET. **Note:** Unsupported Feature warnings do not have IDs. See the next property to suppress errors by ID.
Suppress the following warnings (comma-separated)	Enter a comma-separated list of the IDs of PowerBuilder compiler errors to suppress. Example: `C0003,C0016`
Use .NET nullable types	Maps PowerScript 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.
Enable DEBUG symbol	Choose this option if your application contains code with the DEBUG conditional compilation symbol, and you want this code to run.
Verify CLS compliance	PowerBuilder .NET issues warnings when you deploy objects that are not compliant with the Common Language Specification (CLS).

Before PowerBuilder .NET Version 2.5, .NET Assemply targets could deploy only public NVO functions. In the current version, the Objects tab enables you to deploy multiple objects using additional language elements, including visual objects.

Table 5. Objects Tab

Property	Description
Custom class objects	A tree listing the libraries currently included in the Assembly. Navigate the tree and select a library to display its objects in the table of objects.
Object name	The name of the currently selected object.

Property	Description
Class name	The class name to be used for the object.
Description	Optional text describing the object.
Namespace	The namespace under which the Assembly will be deployed (read-only, defined in the General tab).
Table of object	A list of objects in the library, organized by type. Select the objects to deploy in the Assembly.

The remaining .NET Assembly Project painter tabs define additional properties (some also defined in the wizards):

- **Win32 DLLs –** Specifies any Win32 DLLs you want to include with your project.
- **Version –** Defines the product name, version numbers, copyright text, and related properties that you want to be associated with your deployed assembly.
- **Post-build –** Specifies command line properties for running and debugging the project.
- **Run –** Defines arguments for running the application executable, including its path, runtime arguments, and start location.
- **Sign –** Lets you sign the Assembly.

.NET Assembly Deployment

At runtime, the .NET assemblies that you generate from .NET Assembly targets execute using the .NET Common Language Runtime (CLR).

For deployment of .NET Assembly targets, production servers or target computers must have:

- Windows XP SP3, Windows Vista SP2, Windows 2008 SP1, or Windows 7 operating system
- .NET Framework 4.0 or later
- The Microsoft Visual C++ runtime libraries `msvcr71.dll` and `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

  For more information on the required runtime files, see "Deploying PowerBuilder runtime files" in the *Deploying Applications to .NET* book for PowerBuilder Classic.

Support for CVUOs in .NET Assemblies

The .NET Assembly target exposes custom visual user objects (CVUOs).

When you use the output assembly in a .NET development environment, the visual objects are available as WPF user controls that can be added to WPF windows or user controls.

This support is similar to the support for nonvisual objects (NVOs) in .NET assemblies, providing the same wizard and project painter interface. Functions, events, properties, .NET properties, instance variables, and indexers are exposed through the Objects tab in the project painter. You can also provide Visual Studio IntelliSense descriptions for classes and methods.

In the .NET Assembly Project painter, you can deploy only a CVUO by selecting the Export only the CVO option in the Objects tab. When you choose this option, no other elements are exported: to deploy other objects, you must first unselect Export only the CVO.

Limitations

- You can use standard visual user objects (SVUOs) in .NET assemblies, but they are not directly exposed.
- PowerBuilder exposes only customized events for a CVUO, not events for the inner control.

Creating a WCF Service Target

Create a WCF Service target from scratch, or migrate a WCF Service target from PowerBuilder Classic to PowerBuilder .NET.

Note: To create a target with minimal properties, click **Finish** at any time the button is enabled. You can then use the Project painter to specify WCF Service target properties.

1. Click **File > New**, or open the New dialog box from the toolbar.
2. From the Target category, select **WCF Service**.
3. Specify how to create the target:
   - Create a new WCF Service target
   - Convert a PowerBuilder Classic .NET Web Service target
4. Continue stepping through the New WCF Service pages, following the instructions on each page.

   Optional: If you are creating the target from scratch, you can choose whether to generate an initial object—Custom Nonvisual, standard nonvisual objects, or None—on the page Specify an object to create.
5. On the Hosting Options page, choose the type of host on which you plan to deploy your service:

Hosting Environment	Common Scenarios
IIS	Running a WCF service side-by-side with ASP.NET content on the Internet using the HTTP, HTTPS, TPS, or Named Pipes protocols (if the Web Administration Service component is installed).
Console	Console applications used during development, supporting transport protocols like HTTP, HTTPS, TCP, and Named Pipes.

6. If you chose the IIS hosting option, the Specify Virtual Directory Name page lets you specify the IIS virtual directory name for the WCF service.

   The Web service virtual directory name is the most important setting for a .NET Web service. The default value is the application name of the current target.

   The Web service URL preview automatically displays the URL that will be used to access the service, based on the virtual directory name you provide.

7. In the Specify Deployment Options page, choose one of:

Generate setup file	If you enter a file name with no path information, the file is created in the directory where the PowerBuilder target file exists. If you enter a file name with the path, make sure that the directory exists before deployment begins.
Directly deploy to IIS	Enter the machine name or IP address of an IIS server machine.

8. If you specified the console hosting option, the New WCF Service page enables you to specify console information:

Application file name	Enter the console application file name with an `.exe` extension. The file is created in the output directory. **Note:** The console application file name cannot be the same as the assembly file name.
Base address	Enter a base address for deployed WCF services. For example: `http://localhost:8001`

9. Review the Summary page and click **Finish** to create the target.

When you use the WCF Service target wizard to create a target, the wizard 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 you can use it to create a WCF Service.

Next
After you create a WCF Service target, use the Project painter to:

- Modify the project settings
- Deploy the project
- Run the project
- Debug the project

About WCF Services

Windows Communication Foundation (WCF) is a Microsoft programming model for building service-oriented applications. It enables developers to build secure, reliable, transacted solutions that integrate across platforms and interoperate with existing investments. When compared to a traditional Web service, a WCF Service supports more transport protocols, including HTTP, HTTPS, TCP, MSMQ, and Named Pipes. It also provides both transport and message level security, and supports other Web service enhancement specifications such as WS-ReliableMessaging and WS-Transactions.

Earlier versions of Powerbuilder allowed you to create only traditional Web services that were conformant to the WS-I profile. The traditional ASP.NET Web Service has limited functionalities: it supports only HTTP and HTTPS transport protocols, and XML message encoding and transport-level security. Beginning with Version 12.5, PowerBuilder provides a WCF Services target that can more fully implement the WCF programming model.

Differences With PowerBuilder Classic Web Services

Programmers who have worked with PowerBuilder Classic Web Services will notice some differences developing and deploying WCF Services. For example:

PowerBuilder Classic Web Services	PowerBuilder WCF Service
PowerBuilder Classic Web Services process file operations using the PowerBuilder virtual file system.	WCF Service does not use the PowerBuilder virtual file system. Instead, you can use .NET Framework functions to find absolute paths of virtual directories for read and write file operations.

WCF Concepts

Host Process – Every WCF Service object must be hosted in a Windows process; it cannot exist independently. The process that hosts the WCF Service is called the *host process* or *host*. A host process can be any Windows process, such as an IIS process, a console application, or a Windows Service.

Binding – A binding is a set of options for the transport protocol, message encoding, communication pattern, reliability, security, transaction propagation, and interoperability of the service. You can use predefined bindings provided in the WCF framework, you can modify their properties, or you can write your own custom bindings.

endpoint – Every service is associated with an address that defines where the service is, a binding that defines how to communicate with the service, and a contract that defines what the service does. These three associations are easy to remember as the ABC of the service. WCF formalizes the relationship as an endpoint, concatenating the address, contract, and binding.

WCF Service Target and Project Properties

You define basic properties for a WCF Services project when you create its target in the WCF Service wizard. You can define and modify all its properties later using the Project painter.

WCF Service Project Painter: General Tab

Most of the settings in this tab are also defined in the WCF Service target wizard.

Setting	Description
Project name	Name of the project object the wizard creates.
Library	Name of the library directory the wizard creates. By default, this includes the current solution path and takes the name you enter for the project object with a PBL extension.
Target	Name of the target the wizard creates. By default, this includes the current solution path and takes the name you enter for the project object with a PBTX extension.
Library search path	Lets you add PBL directories to the search path for the new target.
WCF service assembly name	File name of the WCF service assembly. The WCF service assembly is created in the output directory. For an IIS host, it is also copied to the virtual directoy.
Output directory	The full path where the service is deployed.
Application file name	The console application file name. This `.exe` file is created in the output directory. **Note:** Do not specify the same name as the assembly file.

Setting	Description

Base address	For console hosts only. The base URL for deployed WCF Services. Use this syntax:
	[*transport protocol*]://[*machine name* or *domain name*][: *port number*]
	For example:
	`http://localhost:8001`
	`net.tcp://localhost:8002`
	`net.pipe://localhost`
	You can deploy multiple WCF Services under the base address. For example:
	`http://localhost:8001/MyService`
	`http://localhost:8001/MyOther-Service`

Web service virtual directory name	The virtual directory name that the IIS-hosted WCF Service is deployed to.
Service URL preview	Automatically displays the URL accesses the service, based on the virtual directory name.
Build type	Choose either Debug or Release.

Setting	Description
Options	Any of these settings: **Use .NET nullable types –** Maps PowerScript standard datatypes to .NET nullable datatypes when selected. 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. **Enable DEBUG symbol –** Choose this option if the nonvisual objects you deploy as a Web service contain code with the DEBUG conditional compilation symbol, and you want this code to run. **Verify CLS compliance –** If you choose this option, the PowerBuilder to .NET compiler (pb2cs) sets CLSCompliantAttribute to true for the component, and the C# compiler applies Common Language Specification (CLS) rules to the public methods of the component. If you set this value for a component that is not CLS-compliant, the compiler issues the appropriate warnings.
Deployment options	Choose one of these deployment modes, also specified in the wizard: **Setup file name –** If you enter a file name with no path information, the file is created in the directory where the PowerBuilder target file exists. If you enter a file name with the path, make sure that the directory exists before deployment begins. **Deploy directly to IIS –** Enter the machine name or IP address of an IIS server.

WCF Service Project Painter: Objects Tab

To implement a WCF Service, you must define its properties in the Objects tab. The tree view on the left of the tab lists the available NVOs. Select an NVO to set its Web service name and target namespace, and to specify which functions to expose as Web service methods.

NVO icons in the tree view indicate which NVOs are configured and to be deployed; NVOs without the icon are not deployed.

Setting	Description
Web service name	The name of the WCF Web service for the selected NVO. Default: the NVO name.
Target namespace	The targetNamespace attribute of the root descriptions element of a Web Services Description Language (`.WSDL`) XML file. Default: `http://tempurl.org`.
Web service URL	The URL of the Web service. Click **Run Web Service** to open a Web browser to this URL.
Web service WSDL	The URL of the Web service. Click **View WSDL** to open a Web browser to view the WSDL in the browser window **Note:** No WSDL is available when the binding is set to WebHttpBinding. In this case, you can provide information about calling a RESTful Web service by specifying this URL: `http://localhost/test/service-name.svc/help”`
Service Attributes	These attributes control important behavior such as the InstanceContextMode property. **Service Attribute –** Configure service attributes for the WCF Service class. **Operation Attribute –** Configure operation attributes for a function that you select in the Functions Table (described in the next row).

Setting	Description
Functions Table	This table displays the functions available to be exposed as Web services. You must select and configure at least one function for an NVO, otherwise it cannot be deployed. In each row, specify and modify the message name, its function prototype, and operation attribute. • By default, the message name is the same as the NVO function name. However, PowerBuilder modifies message names when two functions have the same name, because the Web service standard does not allow two methods to have the same name.

WCF Service Project Painter: Remaining Tabs

The remaining tabs in the Project painter are identical to the same tabs in the .NET Assembly Project painter.

WCF Service Class Attributes

The Service Attribute editor lists the WCF service class attributes that you can configure in the Objects tab of the WCF Service Project painter.

For detailed information about each attribute, please see the Microsoft *MSDN library* .

Table 6. ServiceContract

Member Name	Description
ProtectionLevel	Specifies whether the binding for the contract must support the value of the ProtectionLevel property.
SessionMode	Gets or sets whether sessions are allowed, not allowed, or required.

Table 7. ServiceBehavior

Member Name	Description
InstanceContextMode	Gets or sets the value that indicates when new service objects are created.

Member Name	Description
AddressFilterMode	Gets or sets the AddressFilterMode that is used by the dispatcher to route incoming messages to the correct endpoint.
AutomaticSessionShutdown	Specifies whether to automatically close a session when a client closes an output session.
ConcurrencyMode	Gets or sets whether a service supports one thread, multiple threads, or reentrant calls. **Note:** Because PowerBuilder is not thread-safe, do not set `ConcurrencyMode=Multiple`.
IncludeExceptionDetailInFaults	Gets or sets a value that specifies that general unhandled execution exceptions are to be converted into System.ServiceModel.FaultException*TDetail* messages of type System.ServiceModel.ExceptionDetail, and sent as fault messages. Set to true only to troubleshoot a service during development.
IgnoreExtensionDataObject	Gets or sets a value that specifies whether to send unknown serialization data onto the wire.
MaxItemsInObjectGraph	Gets or sets the maximum number of items allowed in a serialized object. Type: Int32. Default: 65536 bytes (64KB).
Name	Gets or sets the value of the name attribute in the service element in Web Services Description Language (WSDL).
Namespace	Gets or sets the value of the target namespace for the service in WSDL.
ReleaseServiceInstanceOnTransactionComplete	Gets or sets a value that specifies whether the service object is released when the current transaction completes.
TransactionAutoCompleteOnSessionClose	Gets or sets a value that specifies whether pending transactions are completed when the current session closes without error.

Member Name	Description
TransactionIsolationLevel	Specifies the transaction isolation level for new transactions created inside the service, and incoming transactions that flow from a client.
TransactionTimeout	Gets or sets the period within which a transaction must complete. Type: String. The format is d.hh:mm:ss.ff, where *d* is days, *hh* is hours as measured on a 24-hour clock, *mm* is minutes, *ss* is seconds, and *ff* is fractions of a second.
UseSynchronizationContext	Gets or sets a value that specifies whether to use the current synchronization context to choose the execution thread.
ValidateMustUnderstand	Gets or sets a value that specifies whether the system or the application enforces SOAP MustUnderstand header processing.

Table 8. AspNetCompatibilityRequirements

Member Name	Description
RequirementsMode	Gets or sets the level of ASP.NET compatibility required by the service.

WCF Service Operation Attributes

Use the Operation Attribute editor to define operation attributes for WCF messages.

The Operation Attribute editor lets you define attributes for any selected message in the Objects tab of the WCF Service Project Painter. For detailed information about each attribute, please see the Microsoft *MSDN library*.

Table 9. OperationContract

Member Name	Description
Action	Gets or sets the WS-Addressing action of the request message. Type: String.
AsyncPattern	Indicates that an operation is implemented asynchronously using a Begin*methodName* and End*methodName* method pair in a service contract.

Member Name	Description
IsInitiating	Gets or sets a value that indicates whether the method implements an operation that can initiate a session on the server.
IsOneWay	Gets or sets a value that indicates whether an operation returns a reply message.
IsTerminating	Gets or sets a value that indicates whether the service operation causes the server to close the session after any reply message is sent.
Name	Gets or sets the name of the operation.
ProtectionLevel	Gets or sets a value that specifies whether the messages of an operation must be encrypted, signed, or both.
ReplyAction	Gets or sets the value of the SOAP action for the reply message of the operation. Type: String.

Table 10. OperationBehavior

Member Name	Description
ReleaseInstanceMode	Gets or sets a value that indicates when in the course of an operation invocation to recycle the service object.
TransactionAutoComplete	Gets or sets a value that indicates whether to automatically complete the current transaction scope if no unhandled exceptions occur.
TransactionScopeRequired	Gets or sets a value that indicates whether the method requires a transaction scope for its execution.
AutoDisposeParameters	Gets or sets whether parameters are to be automatically disposed.
Impersonation	Gets or sets a value that indicates the level of caller impersonation that the operation supports.

Table 11. WebInvoke

Member Name	Description
BodyStyle	Gets and sets the body style of the messages that are sent to and from the service operation.
Method	Gets and sets the protocol method (for example, HTTP) that the service operation responds to. Type: String. Values can be POST, GET, DELETE, PUT, HEAD, OPTIONS, TRACE, and CONNECT.
RequestFormat	Gets and sets the RequestFormat property.
ResponseFormat	Gets and sets the ResponseFormat property.
UriTemplate	The Uniform Resource Identifier (URI) template for the service operation. Type: String.

Table 12. WebGet

Member Name	Description
BodyStyle	Gets and sets the body style of the messages that are sent to and from the service operation.
RequestFormat	Gets and sets the RequestFormat property.
ResponseFormat	Gets and sets the ResponseFormat property.
UriTemplate	Gets and sets the Uniform Resource Identifier (URI) template for the service operation. Type: String.

STAOperationBehavior

It is a PB customized operation attribute. Some functions like SaveDocument or Print involve many UI components, which require a thread running in STA (Single Thread Apartment) mode. But for IIS environment, the default thread mode of the WCF service operation is MTA (Multiple Thread Apartment). If a WCF operation uses such functions, you should define the "STAOperationBehavior" attribute for the operation.

Modifying the Configuration File

You can modify the default .NET Framework configuration settings for a WCF Service.

Settings for a WCF service target are defined in a configuration file that the PowerBuilder .NET target wizard automatically generates in the application directory. The XML file, named *project*`.config`, includes the basic service, endpoint, and binding elements required by the service, based on your selections in the wizard.

When you deploy your application to an IIS host, the generated file is copied to the output directory and renamed `Web.config`; if you deploy it to a console host, the file is renamed *`ApplicationFileName`*`.config`. Example: `WCFService_host.exe.config`

If you need a more complex configuration than the generated one, you can edit the `.config` file directly. Or, if you have installed the Microsoft .NET Framework SDK, you can use the WCF Service configuration editor in PB.NET to edit the file.

To open the WCF Service configuration editor directly in PowerBuilder .NET, first add it to the IDE's program selection list:

1. In the Solution Explorer, right-click your configuration file and choose **Open With**.
2. In the Open With dialog, click **Add** . Navigate to `svcconfigeditor.exe` in your Microsoft .NET Framework SDK installation and click **Open** to add the file to the list. For example:

   ```
   C:\Program Files\Microsoft SDKs\Windows\v6.0\Bin
   \SvcConfigEditor.exe
   ```
3. Select the configuration editor from the list and click **OK**.

WCF Service Configuration File Attributes

The WCF Service configuration file defines important service attributes.

The *`project`*`.config` file that PowerBuilder generates contains attributes based on your settings in the WCF Service target wizard or Project painter. This topic documents bindings and security attributes that can be defined in the WCF Service configuration file.

Table 13. Binding Attributes

Binding	Binding Tag	Description
BasicHttpBinding	*basicHttpBinding*	A binding that is suitable for communicating with Web services that conform to the WS-Basic Profile, such as ASP.NET Web services (ASMX)-based services. This binding uses HTTP as the transport and text/xml as the default message encoding.
WSHttpBinding	*wsHttpBinding*	A secure and interoperable binding that is suitable for non-duplex service contracts.

Binding	Binding Tag	Description
NetTcpBinding	*netTcpBinding*	A secure and optimized binding suitable for cross-machine communication between WCF applications.
NetNamedPipeBinding	*netNamedPipeBinding*	A secure, reliable, optimized binding that is suitable for on-machine communication between WCF applications.
WebHttpBinding	*WebHttpBinding*	The WCF Web Programming Model allows developers to expose WCF Web services through HTTP requests that use "plain old XML" (POX) style messaging instead of SOAP-based messaging. For clients to communicate with a service using HTTP requests, an endpoint of the service must be configured with the *wsHttpBinding* that has the *WebHttpBehavior* attached to it.

Table 14. Security Mode Attributes

Security Mode	Description
None	Security is disabled.
Transport	Security is provided using a secure transport (for example, HTTPS).
Message	Security is provided using SOAP message security.
TransportWithMessageCredential	A secure transport (for example, HTTPS) provides integrity, confidentiality, and authentication while SOAP message security provides client authentication.

Security Mode	Description
TransportCredentialOnly	This mode does not provide message integrity and confidentiality. It provides only HTTP-based client authentication. Use this mode with caution, in environments where the transfer security is provided by other means (such as IPSec), and where the infrastructure provides only client authentication.

Table 15. Security Mode Bindings

Binding	Supported Security Mode
BasicHttpBinding	None, Transport, Message (default), TransportWithMessageCredential, TransportCredentialOnly
WSHttpBinding	None, Transport, Message (default), TransportWithMessageCredential
NetTcpBinding	None, Transport (default), Message, TransportWithMessageCredential
NetNamedPipeBinding	None, Transport (default)
WebHttpBinding	None (default), Transport, TransportCredentialOnly

PowerBuilder .NET Datatype Mapping

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

PowerBuilder	C#
boolean	bool
boolean[]	bool[]
blob	byte[]
blob[]	byte[][]
byte	byte
byte[]	byte[]
int	short
int[]	short[]

PowerBuilder	C#
uint	ushort
uint[]	ushort[]
long	int
long[]	int[]
ulong	uint
ulong[]	uint[]
longlong	long
longlong[]	long[]
decimal	decimal
decimal[]	decimal[]
real	float
real[]	float[]
double	double
double[]	double[]
string	string
string[]	string[]
User-defined structure	class [1]
User-defined structure[]	class [] [1]
User-defined NVO object	class
User-defined NVO object[]	class [] [1]
enumerated	enum
enumerated[]	enum[]
Time	Not supported [2]

[1] Data container class

[2] PowerBuilder Date and Time variables need to be converted to DateTime when calling a C# assembly. You can call PowerBuilder assemblies with the Date and Time variables and perform the conversion within the assemblies.

PowerBuilder	C#
Time[]	Not supported [2]
Date	Not supported [2]
Date[]	Not supported [2]
Datetime	DateTime
Datetime[]	DateTime[]
any	Not supported
Multidimension array	Not supported
PB system structure	Not supported
PB system object type	Not supported
PB system nvo type	Not supported
User-defined non-NVO object	Not supported
ResultSet and its descendants	Not supported
Returning any	Array type
Dot Net serializable type (with Serializable or DataContract attributes)	Dot.Net serializable type
Dot Net primitive type	Dot Net primitive type

WCF Service Project Deployment

What happens when you deploy a WCF Service project:

During deployment, PowerBuilder .NET

- For an IIS host, deploys the WCF service assembly file to the virtual directory and output directory specified in the target wizard or Project painter.
- Copies the configuration file to the output directory and (for an IIS host only) virtual directory, and renames it to one of these names:
  - `web.config` if the WCF service is hosted in IIS.
  - *`ApplicationFileName`*`.config` if the WCF Service is self-hosted in a console application. *ApplicationFileName* is the console application file. You can also specify the name in the target wizard and Project painter.
- If the WCF service is hosted in IIS, copies *`ServiceName`*`.svc` file to the virtual directory of each WCF service, where *ServiceName* is the NVO's Web service name.

- If the WCF service is hosted in a console application, creates the hosting application file in the output directory, as specified in the target wizard or Project painter.

Creating a WCF Client

Create a WCF client by building a WCF Client Proxy project, which you can add to any WPF Window Application target.

1. Under the Project node in the New dialog box, select **WCF Client Proxy** to start the wizard.
2. Click **Next** and complete the wizard, entering or selecting values for the project name and library, the service URL or file name, the proxy namespace and assembly, and the library for the proxy. In the wizard, you can also review the services and data classes defined in the service that you select.

   For the service URL or file name, you can enter or choose files with the `WSDL`, `ASMX`, `XML`, or `SVC` extension.

   Note: If your development computer is behind a proxy server, the PowerBuilder WCF Client Proxy wizard uses the proxy server settings from Internet Explorer.
3. Click **Finish** to create the project object.
4. Right-click the new project object in the Solution Explorer and select **Generate Proxy**. PowerBuilder loads the WCF client assembly and creates a proxy NVO that represents a class defined by the service.

   If a service has endpoints with different bindings, PowerBuilder .NET generates one proxy NVO for each binding.

   Each generated proxy has one public property, wcfConnectionObject, which is assigned to a default instance of WCFConnection.
5. (Optional) Change default values of WCFConnection, or create a nondefault instance of WCFConnection, and set all of its binding and binding-related properties.

   - Change specific properties of the WCFConnection object assigned to the wcfConnectionObject property of the generated proxy, such as the endpoint and timeout, as shown in this example:

     ```
     ProxyNVO.wcfConnectionObject.EndpointAddress.URL="xxx"
     ProxyNVO.wcfConnectionObject.Timeout="00:05:00"
     ```
   - Create a WCFConnection object and manually set all its properties, then assign the object to the wcfConnectionObject property of the generated proxy:

     ```
     PBWCF.WCFConnection obj
     obj= create PBWCF.WCFConnection
     obj.EndpointAddress.URL="xxx"
     obj.Timeout="00:10:00"
     ```

```
   obj.BindingType=PBWCF.WCFBindingType.BasicHTTPBinding!
   obj.BasicHTTPBinding=create PBWCF.WCFBasicHTTPBinding
   obj.BasicHttpBinding.PBWCF.TransferMode.Streamed!
   ...
   ProxyNVO.wcfConnectionObject = obj
```

6. Call a method of the instance of the proxy NVO.

   You cannot use PowerScript exception objects to catch errors from a WCF service call. You must catch any exceptions using the .NET System.Exception class, as in this example, or by using a class that inherits from System.Exception:

```
   ns_test_wsdwSoapClient svc
   svc = create ns_test_wsdwSoapClient

   try

svc.wcfConnectionObject.ClientCredential.UserName.UserName="xx"

svc.wcfConnectionObject.ClientCredential.UserName.Password="xxx"

      string ret
      ret = svc.getProverb()
      messagebox("ok", ret)
   catch (System.Exception ee)
         // Error handling
   End try
```

   Note: If your development computer is behind a proxy server, and you do not want the WCF client to use the default proxy server settings from Internet Explorer, you can change the settings with code like this:

```
svc.wcfConnectionObject.ProxyServer.UseDefaultWebProxy = false
svc.wcfConnectionObject.ProxyServer.ProxyAddress =
  "http://hostname:9090"
```

   Otherwise, you can change the proxy server settings at runtime through an instance of the WCFProxyServer class that you assign to the WCFConnection object instance.

7. Deploy and run the WPF application with the WCF project.

About WCF Clients

Windows Communication Foundation (WCF) is a runtime engine and set of APIs that allow you to send messages between clients and services using security settings and a communications protocol defined by service binding properties.

The WCF Client Proxy Project painter generates WCF client proxy objects that consume Web or WCF services from a PowerBuilder .NET application.

Note: You must install the Windows SDK for .NET Framework to use the WCF feature. The WCF Client Proxy project uses a tool from the SDK to access a service contract (`svc` or `wsdl`). This tool (**SvcUtil.exe**) parses the service contract and generates C# files. The C#

compiler (**CSC.exe** from .NET Framework) then builds the newly generated C# code into a private assembly, which is automatically referenced by the project.

The nonvisual user object (NVO) proxy generated by the WCF Client Proxy project allows PowerBuilder .NET application users to consume a Web service or a WCF service. The proxy NVO represents the service class defined by the service contract, which is also incorporated in a private assembly generated by the WCF client project.

The proxy NVO can connect to services with bindings of type: basicHttpBinding, wsHttpBinding, netTcpBinding, and netNamedPipeBinding. If there are any other types of bindings specified in the service contract, the proxy NVO fails at compilation time.

The generated proxy layer uses PowerScript and .NET interoperability to pass the call to the assembly and get a return value from the service. This is different from the approach used by the Web service engine in PowerBuilder Classic, where the generated proxy uses the PowerBuilder Native Interface (PBNI) for data exchange. For this reason, you cannot migrate existing Web service clients from PowerBuilder Classic to the WCF engine.

The WCF client engine allows PowerBuilder .NET application users to consume WSE (Web Service Enhancement) extensions in addition to other Web services. The WSE extensions cannot be accessed by the EasySoap and .NET SOAP engines available in PowerBuilder Classic.

Creating a REST Client

Build a proxy nonvisual user objects that represents a remote Representational State Transfer (REST) service.

Prerequisites

The REST Client Proxy wizard and project require the following installed software:

- .NET Framework 4.0 runtime
- Microsoft Windows SDK for Windows 7 and .NET Framework 4

Task

1. Click **File > New**, or open the New dialog box from the toolbar. **REST Client Proxy** to start the wizard.
2. From the Project category, select **REST Client Proxy**. Follow the instructions on each page of the wizard.
3. In the first page of the wizard, specify the project name and library.
4. In the **Select Service URL** page, specify the URL where the service can be accessed on the Web, and the HTTP method action the service uses.
   The action you select also determines the next page the wizard displays:

Action	Next Wizard Page
HTTP POST or HTTP PUT	Specify request message type
HTTP GET or HTTP DELETE	Specify response message type

5. In the **Specify request message type** page, choose an action to perform for request messages, and provide any additional information the action requires:

Option	Additional Information Required
Skip this step	None. Appropriate for messages that are limited to primitive types and simple arrays.
Use provided schema/ sample data	Specify a schema or sample data that PowerBuilder uses to generate an assembly. • In the box below, enter or paste the schema, or a sample that instantiates the specified datatype. Or, click the **Browse** button to choose a file that contains the schema or data. • Under **Assembly name**, specify the name of the assembly to generate. The wizard automatically prefixes the library path.
Use provided assembly	Under **Assembly name**, specify or select an assembly from the list for the request or response datatype.

   If you choose the second option, PowerBuilder creates the assembly as specified. When the assembly is created successfully, the wizard displays the next page. If the assembly creation fails, the page remains open and displays an error message.

6. In the **Specify request message format and type** page:

   - If your service uses no request message, choose a format of None.
   - Otherwise, choose a format and datatype for request messages.
   - Choose the **Datatype is array** option if appropriate.

   This page is not displayed if the method type is HTTP GET or HTTP DELETE.

7. In the **Specify response message type** page, choose an action to perform for response messages, and provide any additional information the action requires:

Option	Additional Information Required
Skip this step	None. Appropriate for messages that are limited to primitive types and simple arrays.

Option	Additional Information Required
Use provided schema/ sample data	Specify a schema or sample data that PowerBuilder uses to generate an assembly. • In the box below, enter or paste the schema, or a sample that instantiates the specified datatype. Or, click the **Browse** button to choose a file that contains the schema or data. • Under **Assembly name**, specify the name of the assembly to generate. The wizard automatically prefixes the library path.
Use provided assembly	Under **Assembly name**, specify or select an assembly from the list for the request or response datatype.

8. In the **Specify response message format and type** page:
   - If your service uses no response message, choose a format of None.
   - Otherwise, choose a format and datatype for response messages.
   - Choose the **Datatype is array** option if appropriate.
9. In the **Specify proxy information** page, enter the proxy name, library. Optionally, enter a namespace associated with the proxy.
10. Click **Finish** to review the summary page and create the REST client project.
    PowerBuilder .NET generates the project object in the library that you specified in the first wizard page.
11. Double-click the project to open the Project painter. Use the Project painter to define REST service properties, and, optionally, to change properties you defined in the wizard.
12. Deploy the project: in the Solution Explorer, right-click the project object and choose **Generate Proxy**.
    PowerBuilder loads the REST client assembly and creates a proxy NVO that represents a class defined by the service.
13. At runtime, instantiate the NVO and invoke its function, passing any parameters required to get the result.
    For example:

```
test1 svc
svc = create test1
int ret
try
  ret = svc.GetMessage()
  messagebox("ok", ret.ToString())
catch (System.Exception ee)
  messagebox("Failure", "Error")
end try
```

About REST Clients

PowerBuilder .NET supports building proxy nonvisual user object (NVOs) that represent remote RESTful services, and then calling the RESTful services through the proxy objects.

At design time, a **REST Client Proxy** wizard and Project painter enable you to gather the inputs PowerBuilder needs to generate an NVO:

- The URI of the service at runtime.
- A standard HTTP method by which the proxy communicates with the server, such as GET, POST, PUT, or DELETE.
- The message format for exchanging data, such as XML or JSON

At run time, client applications access remote RESTful services using a network URI, and client and server communicate by sending HTTP request and response messages.

REST Client Project Properties

The Project painter lets you modify properties defined in the REST Client Proxy wizard, and define other required service properties

REST Client Project Painter: General Tab

Setting	Description
Deployment PBL	The full path of the library into which PowerBuilder .NET generates the proxy NVO. Example: `C:\MyWPFapp\mytest.pbl`
Library Comments	Any comments about this service that are useful for other developers or maintainers.
Clear deployment PBL on proxy generation	Deletes user objects from the deployment PBL when you generate the proxy.
Confirm before removing user objects from deployment PBL	Gives you a chance to cancel the operation before generating the proxy.
Proxy namespace	An optional namespace for the proxy object.
Proxy name	The file name of the proxy to be generated in the deployment PBL. The file extension is `PBL`.

REST Client Project Painter: REST Services Tab

Setting	Description
Service URL	The URL of the service on the Web. The method URL specification can contain variables that are replaced by constant values at runtime.
Web Method	The HTTP method action the service uses: • HTTP POST (most commonly used action to modify data) • HTTP PUT • HTTP GET • HTTP DELETE
The remaining options in this table are set independently for request messages and response messages.	

Setting	Description
Message Format	The format for message data. The available format options are: • **None –** no message is used (default), and no other message options are available. • **DataContract –** an XML format defined in a WCF data contract, a serializer that abstracts .NET types from schemas and XML data. Typically, RESTful services with data contracts are developed in WCF using a WebHttpBinding endpoint. • **XML –** well-formed XML. • **JSON –** JavaScript Object Notation. Unless you choose None, these additional options are available: • **Use primitive datatypes –** The proxy uses only primitive types and simple arrays. • **Use provided schema/sample data to create assembly –** The generated assembly defines the the specified datatype using a schema or instance that you reference. • **Use datatype from referenced assembly –** PowerBuilder .NET uses an existing assembly that defines the specified datatype instead of generating a new assembly.
Schema (or sample data for request or response message)	A schema or data sample that defines the required datatype. Optionally, click the **Browse** button to choose a file that contains the schema or data. Available only if you choose the option, **Use provided schema/sample data to create assembly**.

Setting	Description
Assembly Name	Name of the data assembly file to generate for the message. Example: `C:\pbtest\WebGetxbNest-Struct.dll` Not available if you choose the option, **Use primitive datatype**. If you choose **Use provided schema/sample data to create assembly**, you can click the **Generate** button to create the specified data assembly file.
Argument Type	Any of the primitive or complex types that you select, or an array of the specified type (if you also select **Array type**).

Using Variables in REST Client Method URLs

When you specify the method URL for a REST client, you can include one or more variables that are resolved at runtime.

Variables give you some flexibility in specifying the method URL. Using a variable involves two steps

1. Insert one or more variables, delimited with braces, in the method URL.

   Variable expressions in the URL must be strings or string variables.

   For example, this URL specification includes two variables:

   ```
   http://myorg.com:7000/webHttpNone/Service.svc/xb/{format}/{type}
   ```

2. Provide the replacement values as parameters to the method that you use to call the service.

   This sample code defines parameters for the two variables shown in the preceding URL:

   ```
   TestService svc
   svc = create TestService
   string  ret
   ret = svc.GetMessage("XML","string")
   ```

The RestService engine resolves the variables at runtime. In the preceding example, the URL is resolved as:`http://myorg.com:7000/webHttpNone/Service.svc/xb/XML/string`.

REST Client Deployment

When you deploy a WCF Service project:

- PowerBuilder creates a proxy NVO. Each NVO uses one of these methods, based on the method you choose in the wizard or painter:

  GetMessage
  PostMessage
  PutMessage
  DeleteMessage
- Assigns the public property, `RestConnectionObject`, to a default instance of WebConnection. You can use this property to set many service items such as the endpoint URI, message format, and client credentials.
- Automatically adds any generated assemblies for the request or response message to the target reference.
- If the response type defined in the wizard is RSS2.0 or ATOM1.0, the associated method is `System.ServiceModel.Syndication.SyndicationFeed`. This class is defined in the `System.ServiceModel.dll` assembly, which is required to compile the generated proxy.

REST Service Classes

The generated REST client NVO uses the PowerBuilder RestService class and several supporting classes.

WebConnection Class

The `WebConnection` class enables you to customize request properties. You can use it to set the endpoint URI, message format, and client credential.

Property or Method	Return Type	Read/Write	Description
Endpoint	String	RW	Gets and sets the method URI. Default: null.
RequestMessageFormat	WebMessageFormat (enum)	RW	Gets and sets one of these request message formats: XML (default) JSON DataContract

Property or Method	Return Type	Read/Write	Description
ResponseMessageFormat	WebMessageFormat (enum)	RW	Gets and sets one of these request message formats: XML (default) JSON DataContract
ClientCredential	WebClientCredential	RW	Gets and set the client credential items (for example, security mode, user name, and password).
ProxyServer	WebProxyServer (class)	RW	Gets and sets the client firewall.
MaxMessageLength	System.UInt32 (ulong in PowerScript)	RW	Gets and set the maximum length of the message, in bytes. Default: 262144 (256 KB)

WebMessageFormat enum

The `WebMessageFormat` enum allows three formats for the message body:

- **XML –** well-formed XML.

  String Example:

  ```
  <string>You have entered (Xml, Bare): 12345</string>
  ```

  Int Array Example:

  ```
  <ArrayOfint>
    <int>0</int>
    <int>1</int>
    <int>2</int>
  </ArrayOfint>
  ```

  Structure Example:

  ```
  <myStruct>
    <i1>10</i1>
    <str1>Xml, Bare</str1>
  </myStruct>
  ```

- **DataContract –** DataContract data is an XML format defined in a WCF data contract. By default, data contracts are assigned a namespace that comes from the common language runtime (CLR) namespace of the datatype.

String Example:

```
<string xmlns="http://schemas.microsoft.com/2003/10/
Serialization/">
  You have entered (Xml, Bare): 12345
</string>
```

Int Array Example:

```
<ArrayOfint xmlns="http://schemas.microsoft.com/2003/10/
Serialization/Arrays"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <int>0</int>
  <int>1</int>
  <int>2</int>
</ArrayOfint>
```

Structure Example:

```
<myStruct xmlns="http://schemas.datacontract.org/2004/07/None"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <i1>10</i1>
  <str1>Xml, Bare</str1>
</myStruct>
```

- **JSON –** data in the JavaScript Object Notation format. Examples of string, array, and structure messages:

  String Example:

  ```
  "You have entered (Json, Bare): 12345"
  ```

  Int Array Example:

  ```
  [0,1,2]
  ```

  Structure Example:

  ```
  {"i1":30,"str1":"Json, Bare"}
  ```

WebClientCredential Class

This class specifies properties for REST service client credentials. When called, the PowerBuilder REST client uses credentials to communicate to the service. It supports all transport-level authentications, including Windows authentication, Basic, Digest, and certificate authentication.

Property or Method	Return Type	Read/Write	Description
AccessAuthentication	AuthenticationMode (enum)	RW	Gets and sets one of these authentication modes: Anonymous (default) Basic Digest
Domain	String	RW	Gets and sets the domain that the user is based on. Ignored when AccessAuthentication is set to Anonymous.
Username	String	RW	Gets and sets the user name. Ignored when AccessAuthentication is set to Anonymous.
Password	String	RW	Gets and sets the password. Ignored when AccessAuthentication is set to Anonymous.
UseIntegratedWindowsAuthentication	Boolean	RW	Get and sets using Integrated Windows authentication.
Certificate	WebClientCertificate (class)	RW	Gets and sets the client certificate.

WebClientCertificate Class

This class specifies a certificate from a certificate store or a certificate file. When called, the REST client exchanges the certificate with the server for authentication.

Property or Method	Return Type	Read/Write	Description
StoreName	CertStoreName (enum)	RW	Gets and sets the certificate store name, using one of: AddressBook AuthRoot CertificateAuthority Disallowed My (default) Root TrustedPeople TrustedPublisher
StoreLocation	CertStoreLocation (enum)	RW	Gets and sets the certificate location, using one of: CurrentUser (default) LocalMachine

Property or Method	Return Type	Read/Write	Description
FindType	CertFindType (enum)	RW	Specifies how to find the certificate in the cert store, using one of: FindByThumbprint FindBySubjectName (default) FindBySubjectDistinguishedName FindByIssuerName FindByIssuerDistinguishedName FindBySerialNumber FindByTimeValid FindByTimeNotYetValid FindByTimeExpired FindByTemplateName FindByApplicationPolicy FindByCertificatePolicy FindByExtension FindByKeyUsage FindBySubjectKeyIdentifier
FindValue	string	RW	Specify the value to find.
CertFileName	string	RW	Specify one or more certificate files, using a semicolon-separated list.

WebProxyServer

This class specifies properties for a client firewall. When called, the REST client communicates with the server through the firewall.

Property or Method	Return Type	Read/Write	Description
Address	string	RW	Gets and sets the URL of the firewall. Default: null. Example: `http://www.x12345.com :8000`
BypassProxyOnLocal	bool	RW	Specifies whether the client uses the proxy server when the services are on the local machine. False (default): all Internet requests are made through the proxy server. True: local Internet resources do not use the proxy server. Local requests are identified by the absence of a period in the URI (for example, `http://webserver/`), or by a URI that explicitly addresses a local resource (for example, `http://localhost`, `http://loopback`, or `http://127.0.0.1`).

Property or Method	Return Type	Read/Write	Description
UseDefaultCredentials	bool	RW	Specifies how the server authenticates proxy requests for access. False (default): does not require the user to provide credentials. Instead, uses the value set in the `Credential` property for authentication. True: requires the server to authenticate requests using credentials provided by the user. Ignores the `Credential` property setting.
Credential	WebProxyCredential (class)	RW	Gets or sets the credentials to submit to the proxy server for authentication. The Credential property contains the authentication credential to send to the proxy server. Default: null.

WebProxyCredential

This class specifies the credential of the proxy server. When called, the REST client uses the credential to communicate with the server.

Property or Method	Return Type	Read/Write	Description
Domain	string	RW	Gets and sets the user's domain.

Property or Method	Return Type	Read/Write	Description
Username	string	RW	Gets and sets the user name.
Password	string	RW	Gets and sets the password.

Adding Resources to a Target

When you create a new target in PowerBuilder .NET, you can use the Select Resources page of the new target wizard to include resources such as pictures or text files. After the target is created, you use the context menu items on the target or its subfolders to include new or existing resource files.

The new target wizard adds the resources and resource directories you select on its Select Resources page to a Resources folder under the target directory. If you do not use the wizard to create a Resources folder, you can add resources directly to the target directory, or you can add a folder under the target directory to contain the resource files that you add.

Note: Resources are external to the target in PowerBuilder Classic, but not in PowerBuilder .NET. When you migrate an application and select its resource files or directories in the new target wizard, the resource files are copied relative to the new target path, based on their location relative to the original target.

If the application that you migrated did not reference the resource files relative to the target location (for example, in a path relative to the Application object, when that is not the same as the target location), you can drag and drop the resource directory to the correct location in the Solution Explorer after migration.

You can drag and drop resource files or directories from the Windows Explorer directly onto the target object or a folder under the target object in the Solution Explorer, or you can use this procedure:

1. (Optional) Right-click the target object in the Solution Explorer, select **New Folder**, then type a name for the folder.
2. Right-click the new folder or the target object, and select:

Add New Item	Select the type of resource you want to add, click **Add**, then double-click the new item in the Solution Explorer to edit the new object in its associated editor.
Add Existing Item	Select the file for the resource you want to add, and click **Add**. You can view or edit the object by double-clicking it in the Solution Explorer.

If you right-click a folder, the item you add is included under the folder; if you right-click the target, the item you add is included in the target directory. Either way, the resource item is deployed and available to the target when you build a target project.

Batch Command Processing

The **pbshell** command supports batch building and deployment without direct intervention.

The command uses this syntax:

Pbshell.exe /pbcommand *command-file-path*

command-file-path is a full path that identifies a file containing batch commands. For example:

```
pbshell.exe /pbcommand "C:\MyProjects\MyCommandFile.cmd"
```

In the file, each line contains a single command. These commands are available:

Full Build Solution
Full Build Target
Incremental Build Solution
Incremental Build Target
Deploy Solution
Deploy Target
Output File
Comment
Solution
Target

Painter (form, query, project, database)
Library
Open
New
Run
RunOnly
Target

The Output File command enables you to specify a file where output for each command is appended, with a date time stamp.

The shell closes when all commands have been processed.

Scripts and Code Fundamentals

PowerBuilder applications are event driven. Write a script to specify the processing that takes place when an event occurs.

Script View in PowerBuilder .NET

Use the Script view to code functions and events, define your own funtions and events, and declare variables and external functions.

Script views are part of the default layout in the Application, Window, User Object, Menu, and Function painters.

There are three drop-down lists at the top of the Script view.

In the first list, you can select the object, control, or menu item for which you want to write a script. You can also select Functions, Indexers, Properties, Enumerations, Structures to edit those, or Declare to declare variables.

The second list lets you select the specific item you want to edit or the kind of declaration you want to make, based on the selection in the first list. A script icon next to an event name indicates there is a script for that event, and the icon's appearance tells you more about the script:

If there is a script	The script icon displays
For the current object or control	With text
In an ancestor object or control only	In color
In an ancestor as well as in the object or control you are working with	Half in color

The same script icons appear in the Event List view.

The third list is available in descendant objects. It lists the current object and all its ancestors, so you can view scripts in the ancestor objects.

Script Navigation Option

Set scripts to open separately or in the same view.

Select **Tools > Options > Text Editor > PowerScript > Miscellaneous** to determine the behavior of the Script editor when you select an item from one of the drop-down lists.

Option	Description
Open scripts in the current Script view	Items open in the same Script view, similar to PowerBuilder Classic.
Open scripts in a new Script view	Items open in a new Script view, one for each item.

Opening the Script View

You can open the Script view from the Solution Explorer context menu or by double-clicking a control, function, or event.

If there is no open Script view, selecting a menu or PainterBar item that requires a Script view automatically opens one.

Do one of:

From the Solution Explorer	Right-click a scriptable control and select **Open Script**.
For a control	In the Layout view, double-click a scriptable control.
For a function or event	Double-click an event or function in the Solution Explorer or PB Object Outline.

If you double-click a control, function, or event, the Script view shows the script for the selected item. If the Script view is in a tabbed pane and is hidden, it pops to the front. If there is no open Script view, PowerBuilder creates a new one.

Modifying Script View Properties

The Script view automatically color-codes scripts to identify datatypes, system-level functions, flow-of-control statements, comments, and literals. It also indents the script based on flow-of-control statements. You can modify these and other properties.

1. Select **Tools** > **Options** to display the Options dialog box for the painter.

   The Options dialog box includes pages that affect the Script view.
2. Choose the page appropriate to the property you want to specify:

To specify	Choose this page
Font family, size, and color	**Environment** > **Fonts and Colors**
Tab size and automatic indenting	**Text Editor** > **PowerScript** > **Tabs**
Preferred coding style	**Text Editor** > **PowerScript** > **Formatting**
Script list contents	**Script Lists** > **General**

Editing Scripts

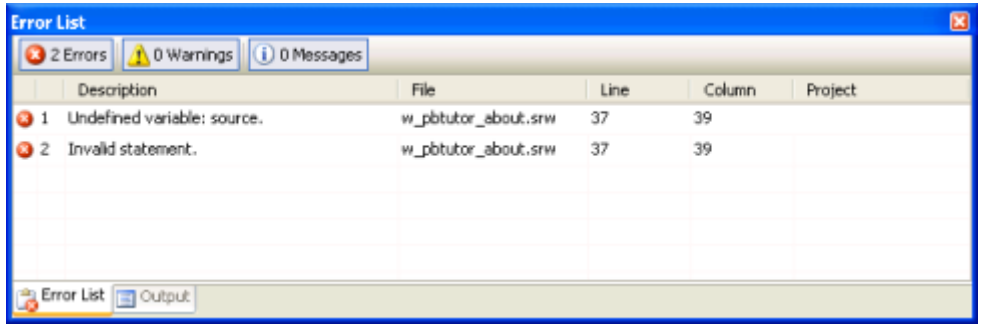
Perform standard editing tasks in the Script view using the Edit menu, the pop-up menu in the Script view, or the PainterBars. There are shortcuts for many editing actions.

Note: You can set up your own shortcuts. In the Options dialog box, select **Tools** > **Options** > **Environment** > **Keyboard**.

See the *PowerBuilder Users Guide*.

Handling Problems with Script Compilation

If problems occur, PowerBuilder displays messages in the Error List tab of the Output view.



There are three kinds of messages:

- Errors
- Warnings
- Information messages

See the *PowerBuilder Users Guide*.

Code Snippets

You can store short code snippets and reuse them in scripts.

Code snippets enable you to store and reuse pieces of code in multiple scripts or files. Snippets are similar to clips in the PowerBuilder Classic environment.

PowerBuilder .NET provides a library of PowerBuilder-specific snippets. These are available with the Insert Snippet and Surround Snippet functions when you are editing code in PowerScript.

For information about adding, using, and managing IntelliSense code snippets, refer to the Visual Studio help.

IntelliSense

IntelliSense helps you write PowerScript code more quickly by providing a lookup and paste service inside the Script view.

IntelliSense is a feature of Visual Studio that is similar to Autoscript in PowerBuilder Classic.

You can use IntelliSense when you:

- Can remember part of the name and you want IntelliSense to finish typing it for you or show you a list of alternatives.
- Cannot remember the name or you just want a list. IntelliSense options can help you narrow the list if you do not know the name but you do know the type you are looking for.
- Want a list of the properties, functions, or events that apply to an identifier, followed by a dot.

Identifier Names

The rules for naming variables, labels, functions, windows, controls, menus, and anything else you would reference in a script are different in PowerBuilder .NET than in PowerBuilder Classic.

Rules for identifiers:

- Must start with a letter or an _ (underscore)
- Cannot use reserved words (see "Reserved Words" in the *PowerScript Reference*)
- Can have up to 40 characters but no spaces
- Are not case sensitive (PART, Part, and part are identical)
- Can include any combination of letters, numbers, or _ (underscores)
- Cannot include any of these special characters:
  - - Dash
  - $ Dollar sign
  - # Number sign
  - % Percent sign

Inner Control Properties and Methods

The properties and methods of controls in the System.Windows.Controls namespace are accessible in the PowerBuilder .NET Script Editor through the PowerScript InnerControl property. These controls are the ancestors to all PowerBuilder .NET controls.

The IntelliSense feature in the PowerBuilder .NET Script Editor displays the list of properties, events, and functions of the ancestor Microsoft control when you enter script for the descendant PowerScript control's InnerControl property, followed by a dot.

The InnerControl property returns the actual datatype of the ancestor control. It is read-only, although you can use it to change properties or call methods on the ancestor control.

This example changes the value of the Content property of the ancestor to a PowerScript CommandButton control:

```
cb_1.InnerControl.Content = "My Content"
```

In PowerScript code, you cannot access the Content property on the CommandButton control itself, although you can enter a value for it in the Properties view. When you enter a value for Content in the Properties view, the value is available to the Text property for the CommandButton control, but only when the Text property is not already assigned.

Declaring Variables and External Functions

The default layout in the Application, Window, and User Object painters includes a Script view where you can declare variables. Keeping a separate Script view open makes it easy to declare any variables or external functions you need to use in your code without closing and compiling the script.

1. Select **(Declare)** from the first list in the Script view.
2. Select the variable type (instance, shared, or global) or the function type (local or global) from the second list.

   You can also declare namespaces, using namespace directives, and interfaces in the Script view.
3. Type the declaration in the Script view.

Next

For more information about:

- Declaring and using variables, see the *PowerScript Reference*.

- Declaring and using external functions, see the *PowerScript Reference* and *Application Techniques*.
- Namespaces and using namespace directives, see *Declaring a Namespace*.
- Interfaces, see *Defining an Interface*.

.

Memory Allocation for External Functions

When calling a string-related external function in .NET targets, you must modify the function's source code in the DLL to allocate memory for the string.

If the original source code looks like this:

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

Sybase recommends that you change it to:

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

Go To Definition

Go To Definition allows you to find the definition of a type, variable, method, event, or object that you select in the Script view.

From the Script view, you can easily navigate to a definition by right-clicking an item and selecting **Go To > Go To Definition**. The item opens in the editor as if you had selected it from the Solution Explorer.

Skin Selection for Applications and Controls

At design time, you can select system-defined and custom skins for PowerBuilder .NET applications and individual visual controls. You can also allow customers to change skins on applications and controls at runtime.

The ability to apply different skins to PowerBuilder .NET controls allows you to create visually compelling applications that can also be customized by application users. PowerBuilder provides system-defined skins, but also allows you to select custom skin files that use the XAML extension.

System-defined skins

The default skin defines the behavior and simulates the appearance of visual controls in PowerBuilder Classic. It also supports display themes that depend on system selections for the design time or runtime computers. These include the standard Microsoft Windows themes, such as Aero on Windows Vista, Luna on Windows XP, and Windows Classic on all Windows operating systems. Customized themes are also supported.

PowerBuilder also provides the following system-defined skins: `ExpressionDark`, `Metal`, `RainierOrange`, `RoughGreen`, and `ShinyDarkTeal`. These skins inherit from the default skin, enabling them to retain the behavior of all PowerBuilder controls. However, their styles are not affected by the theme setting of the host computer, and they display control surfaces with different colors.

The Application object in all WPF Windows applications always has a skin, whether the application has been migrated from a PowerBuilder Classic target or created from scratch. By default, visual controls are not assigned a skin. However, if you do not assign skins to visual controls, they inherit skin settings from their immediate parent. For example, if there is no skin defined for a CommandButton, it takes its parent control's skin. The parent can be a GroupBox, TabPage, Window, or other container control. If the container controls do not have assigned skins, the CommandButton uses the skin set in the Application object.

Note: Skin inheritance works only at runtime. If the Skin property is not set on a control, the skin of its container or the Application object is not applied at design time.

Custom skins

You can create and select custom skins to affect the styles (colors, fonts, borders, and so on) of the visual controls in PowerBuilder .NET applications. However, if you use a custom skin that is not based on the system-defined default skin, you are likely to lose some of the controls' more complex behaviors that are embedded in the default skin's XAML definitions.

Add custom skin files to the directory containing the application executable, or to a subdirectory of that directory. You can select the custom skin for a control by assigning it to the Skin property for the control.

To use the custom skin at runtime, add the XAML file to a folder that you create under the target containing the control. If the compile-time Build Action property in the Properties view for the XAML file is set to "Content" (the default), the XAML file is deployed as a standalone file. If you set the Build Action property to "Embedded Resource," the XAML file is compiled into the target assembly. Custom skins are not applied at runtime if the Build Action property for the XAML file is set to "None" when the project is deployed.

Skin property

The Skin property is available for all visual controls and for the Application object in a WPF Window target. You can set the value of the Skin property in PowerScript code, or by entering

it in the Properties view for a visual control or the Application object. The Skin property is under the General category in the Properties view when you display properties by category.

The datatype of the Skin property is string. You can set this property to a system-defined skin, or to a custom skin file with a XAML extension. The default skin is designated by an empty string (" ").

This example sets an application-level skin to the system-defined Metal skin:

```
applicationName.Skin = "Metal"
```

This example sets a window-level skin to an external file:

```
w_main.Skin = "skin1.xaml"
```

Right-To-Left Formatting

In PowerBuilder .NET, the RightToLeft PowerScript property has been deprecated, and is replaced by the FlowDirection property.

The RightToLeft properties that you set in the PowerBuilder Classic IDE automatically convert to FlowDirection values when you migrate your applications to PowerBuilder .NET.

Levels of right-to-left support

Right-to-left features in WPF applications can affect text order, text and control alignment, control title bar layout, control position mirroring, and can be inherited from container controls.

In PowerBuilder Classic client server applications, setting the RightToLeft property supports only right-to-left text order, and text and control alignment. In .NET Windows Forms applications, the RightToLeft property also determines the layout of the title bar (for controls that have title bars), causing the Close, Maximize, and Minimize buttons to appear on the left side of the title bar, and the control icon on the right.

With earlier versions of PowerBuilder, you could design two separate user interfaces (UIs) to mimic control position mirroring. You can still use this type of design for PowerBuilder Classic, and you can migrate these applications to PowerBuilder .NET. However, for newer PowerBuilder .NET applications, it is much easier to use a single UI design and take advantage of the FlowDirection property for automatic mirroring support.

Property inheritance for right-to-left features in PowerBuilder .NET applications means that if you set the flow direction on a container control, such as a GroupBox or Grid, all the controls that are placed in the container automatically use the same flow direction value—unless you override that value in the individual controls. In earlier versions of PowerBuilder, and in PowerBuilder Classic, this type of inheritance is not supported.

Although you cannot currently set a FlowDirection property on DataWindow objects and DataWindow columns, when you display a DataWindow object in a DataWindow control, the

flow direction of the DataWindow object and its columns is determined by the value of the FlowDirection property of the DataWindow control.

Effects of flow direction on system functions

When calling a system function to open an object (**Open**, **OpenSheet**, **OpenSheetWithParm**, **OpenWithParm**, **OpenUserObject**, **OpenUserObjectWithParm**, **OpenTab**, or **OpenTabWithParm**), flow direction follows the FlowDirection property set for the object's container or for the Application object, unless the object you are opening has its own FlowDirection setting.

When calling a system print function with coordinate arguments (**Print**, **PrintScreen**, **PrintText**, **PrintBitmap**, **PrintRect**, **PrintRoundRect**, **PrintLine**, or **PrintOval**), and the Application object FlowDirection property is set to RightToLeft!, the print coordinates are converted to new values based on the change of the 0,0 start point from the top, left-most part of the page to the top, right-most part of the page.

The **MessageBox** and **PopMenu** system functions inherit the Application object FlowDirection setting.

Migrating right-to-left applications

Migration converts RightToLeft property values that you set in the PowerBuilder IDE, but it does not convert values that you set in script. Although the RightToLeft property is still supported by PowerBuilder .NET at runtime, it is often more useful to modify these scripts to use the FlowDirection property.

The PowerBuilder .NET Migration wizard has a "Specify RTL Conversion Option" page that enables you to designate whether the conversion process includes automatic mirroring. Do not select this option if you are migrating right-to-left applications with separate UIs that mimic control position mirroring, and you want to preserve the dual UI design. When the option is not selected, the Migration wizard strategically places FlowDirection values to prevent automatic mirroring, yet continues to maintain the right-to-left orientation where required.

If you later want to switch to automatic mirroring, edit the XAML to remove any of the LeftToRight! FlowDirection values inserted by the Migration wizard. Without an explicit FlowDirection value, the controls take on the FlowDirection value of their containers, which, in these types of migrated applications, is RightToLeft!.

FlowDirection Property

The FlowDirection property affects text order, text and control alignment, control title bar layout, and control position mirroring in WPF applications. It replaces the deprecated PowerScript RightToLeft property.

Applies to

Application object and all visual controls

Usage

The FlowDirection property takes one of these enumerated values:

- LeftToRight! (default) – characters display in left-to-right order.
- RightToLeft! – characters display in right-to-left order, control position is automatically mirrored, control title bar button positions are mirrored, and text and controls are right-aligned.

The flow direction on a container control is automatically inherited by other controls in the container. However, you can override the flow direction of the container by including a different flow direction value for the controls you place inside the container.

To use Arabic or Hebrew text for messages, set the FlowDirection property of the Application object to RightToLeft!. This causes the characters in messages to read from right to left. However, text continues to show in English unless you are running a localized version of PowerBuilder.

You can set the FlowDirection property:

- In a painter – in the Properties view, select one of the enumerated values in the drop-down list for the FlowDirection property.
- In scripts – set the FlowDirection value to LeftToRight! (default) or RightToLeft!.

Coding Restrictions

Several PowerScript coding practices are not permitted in the .NET environment.

Dashes in identifiers

In the .NET platform, you cannot use dashes in identifiers. If you include a dash or hyphen in the name of an object that you create in the IDE, you see an error message when you try to save the object. The migration wizard in PowerBuilder .NET includes a check box on the Convert Dashes in Identifiers page that you can select to convert dashes in identifiers to default "dash" strings in applications that you migrate from PowerBuilder Classic. The conversion is automatic unless you clear the check box before completing the wizard, or change the DashesInIdentifiers setting in the `PB.INI` file to 0. The wizard also allows you to convert dashes to a string of your own choosing.

Microsecond separators

The .NET platform requires you to use decimal points as microsecond separators in time functions, rather than colons. Although you can continue to use colons as microsecond separators in PowerScript, the PowerBuilder to .NET compiler converts colon separators to decimal points prior to deployment of .NET applications and components.

PowerScript GoTo statement

In PowerBuilder Classic, you can jump directly to a branch of a compound statement because the concept of scope inside a function does not exist in PowerScript. For example, this code works well in PowerBuilder Classic:

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

This PowerScript translates conceptually into this C# code:

```
if (b == 0)
   {    // opening a new scope
   label: ...
   }
else
   {
    ...
   }
goto label;
```

Since a GoTo statement is not allowed to jump to a label within a different scope in .NET, the C# code would not compile. For this reason, avoid using GoTo statements in PowerBuilder .NET.

Casting objects 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.

Accelerator Characters in Control Labels

XAML uses the first single underscore in a control label to define the character that follows the underscore as an accelerator character for the control. This is different from PowerScript, which uses an ampersand to signal an accelerator character.

When you migrate PowerBuilder Classic targets to PowerBuilder .NET, the ampersand that signals an accelerator character in PowerScript is converted to an underscore. If the control label has an underscore in it, the migration adds an extra underscore as an escape character, and XAML displays the control label with a single underscore that is a regular part of the label text. You cannot use the underscore character itself as an accelerator in XAML.

However, as in PowerBuilder Classic, accelerator characters in PowerBuilder .NET control labels appear with an underscore in the label.

Although migration converts the accelerator characters for PowerBuilder Classic targets, any new accelerator characters you want to use in Script views or in the XAML editor must follow the XAML accelerator character rules. The rules for accelerator characters apply to CommandButton, PictureButton, CheckBox, RadioButton, StaticText, StaticHyperlink, and GroupBox controls.

In XAML, the control label *myCommandButton__1_x* designates the character *x* as the accelerator for the control: *myCommandButton_1* x. In this example, *x* is the first character after a single underscore, which marks it as an accelerator character.

Keywords as Identifiers

When you use a PowerScript keyword as an identifier, PowerBuilder .NET automatically prepends the commercial at symbol (@) to the identifier in the source code.

The @ symbol allows the compiler to distinguish between the keyword and the identifier. Although the identifier still appears without the @ symbol in PowerBuilder .NET painters and in IntelliSense, when you add the identifier to a script by selecting it in the IntelliSense list, the Script Editor includes the @ symbol prefix in the identifier.

If you add the same identifier by typing it in a script, make sure to enter the @ symbol prefix. The compiler removes the symbol when it builds the application or component containing the identifier. However, if you type two @ symbols as a prefix to an identifier, the symbols are not removed by the compiler and become part of the identifier name after compilation.

Note: Use of the @ symbol is optional for these keywords: open, close, create, destroy, post, describe, and update.

Also, "system" is not a keyword in the PowerBuilder .NET IDE, although it remains a keyword (reserved word) in PowerBuilder Classic. The following external function declaration is a correct usage of the "system" keyword in PowerBuilder Classic, but reports an error when used in PowerBuilder .NET:

```
function int foo() system library "aaa"
```

You can find a list of PowerScript reserved words in the *PowerScript Reference*.

If you name an event in an object painter with the keyword "event," a System.Type return type, and a System.Type argument to which you assign the "for" identifier, the source code for the event is saved as:

```
event type System.@Type @event (System.@Type @for)
end type
```

Supported Custom Events

PowerBuilder .NET cannot generically support user-defined custom events because there is no matching concept for these types of events in WPF applications. However,

PowerBuilder .NET does provide support for some of the more commonly used events of this type.

This table lists PowerBuilder event IDs for custom events supported in PowerBuilder .NET. Custom events supported in an ancestor object are also supported in its descendants.

PowerScript object type	Supported custom event
DragObject	pbm_char, pbm_keydown, pbm_keyup, pbm_lbuttondown, pbm_lbuttonup, pbm_mousemove, pbm_rbuttondown, pbm_rbuttonup, pbm_size
DataWindow	pbm_dwnkey, pbm_dwndropdown, pbm_dwngraphcreate, pbm_dwnmessagetext, pbm_dwnprocessenter, pbm_dwntabout, pbm_dwnbacktabout, pbm_dwntabupout, pbm_dwntabdownout
SingleLineEdit	pbm_enchange
MultiLineEdit	pbm_enchange
Window	pbm_move

Using Multithreading

When you deploy a PowerBuilder application that contains shared objects 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:

```
//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
```

Using .NET Synchronization Functions

When using threading in PB .NET, consider using .NET synchronization functions to protect user data from corruption. Synchronization functions require passing a System.Object instance to the functions, so your script must define a System.Object variable To use .NET synchronization functions in PowerScript:

1. Declare a global variable.
2. Initialize the global variable.
3. Use the global variable in your .NET synchronization functions.

For example:

```
//Declare a System.Object class
System.Object obj

//PowerScript function definition
//using .NET thread synchronization
//functions

global subroutine f_compute ();
  System.Threading.Monitor.Enter(obj);
  gCount = gCount + 1
  System.Threading.Monitor.Exit(obj);
end subroutine
```

Printing DataWindow and DataStore Objects

For printing DataWindow and DataStore objects, these limitations apply:

- Only deployed objects like .NET Assembly and WCF support multithreading. Client applications likeWPF and WinForms applications do not support multithreading.
- You cannot use interactive operations or UI-related dialogs to print in multithread mode. Instead, print silently by providing the print file name for the DataWindow. For example:

  ```
  //ids is a datastore instance
  ids.object.DataWindow.Print.FileName =ls_filename
  ids.Print( False, False)
  ```

- If you call any of a Data Window's Print, SaveAs, or Graph functions in a worker thread, start the thread in STA mode. For example:

  ```
  dwThread = create Thread (dwMethodFunc)
      dwThread.SetApartmentState(ApartmentState.STA!)
      dwThread.Start()
  ```

Unsupported Properties, Events, and Functions

Several PowerScript properties, events, and functions are not supported in PowerBuilder .NET.

Unsupported PowerScript properties

This table lists PowerBuilder .NET objects and controls with unsupported PowerScript properties:

Object or control	Unsupported properties
All controls inheriting from DragObject	AccesibleDescription and AccessibleRole AccessibleName is supported in DataWindow objects, but not in other objects that inherit from DragObject.
Application object	ToolbarUserControl
CheckBox	Automatic
DataWindow control (for a list of unsupported DataWindow object properties, see *DataWindow Differences Between PowerBuilder Classic and PowerBuilder .NET*)	ControlMenu, HSplitScroll, Icon, MaxBox, MinBox, Resizable, Title, and TitleBar
DatePicker	AllowEdit
DropDownListBox	IMEMode
DropDownPictureListBox	IMEMode
EditMask	AutoHScroll, AutoVScroll. FontCharSet, FontFamily, FontPitch, HideSelection, HScrollbar, IgnoreDefaultButton, IMEMode, TabStop, and VScrollBar
HTrackBar	SliderSize
ListBox	TabStop
ListView	IMEMode
Menu	MergeOption
MenuCascade	Columns, CurrentItem, DropDown, and MergeOption

Object or control	Unsupported properties
MultiLineEdit	AutoVScroll, HideSelection. IgnoreDefaultButton. IMEMode, and TabStop
Picture	Map3DColors (Use PictureMaskColor instead)
PictureButton	Map3DColors (Use PictureMaskColor instead)
PictureHyperLink	Map3DColors (Use PictureMaskColor instead)
PictureListBox	TabStop
RadioButton	Automatic
RichTextEdit	Accelerator, ControlCharsVisible, DisplayOnly, FontCharSet, FontFamily, FontPitch, HScrollBar, ImeMode, PicturesAsFrame, Resizable, ReturnsVisible, RulerBar, SpacesVisible, TabBar, TabsVisible, Underline, UndoDepth, and WordWrap
SingleLineEdit	HideSelection, IMEMode
Tab	Alignment, FixedWidth, FocusOnButtonDown, ImeMode, and RaggedRight
UserObject	ColumnsPerPage, LinesPerPage, Style, UnitsPerColumn, and UnitsPerLine
VTrackBar	SliderSize
Window	AccessibleDescription, AccessibleRole, AnimationTime, ClientEdge, CloseAnimation, ColumnsPerPage, ContextHelp, LinesPerPage, OpenAnimation, ToolbarHeight, ToolbarWidth, ToolbarX, ToolbarY, Transparency, UnitsPerColumn, and UnitsPerLine

Note: Several properties have been renamed with a "PB" prefix. For example, Height and Width properties are set as PBHeight and PBWidth; the Tag and FontFamily properties are set as PBTag and PBFontFamily. TabOrder is included as a property in the Property view for an object or control.

Unsupported PowerScript events

This table lists PowerBuilder .NET controls with unsupported PowerScript events:

Control	Unsupported events
All controls	Other
DataWindow	HtmlContextApplied, PrintMarginChange, and RButtonUp
DatePicker	UserString
Menu	Help

Note: The Help event is supported, and can be triggered by pressing the F1 key in windows, menus, and for all controls that inherit from DragObject. Although the Help event can also be triggered in PowerBuilder Classic by clicking a question mark icon in a window title bar, then clicking a control in the same window, in the .NET Framework, the question mark icon is not available, and users cannot open context-specific help using this method.

Unsupported PowerScript functions

This table lists PowerBuilder .NET objects and controls with unsupported PowerScript functions:

Object or control	Unsupported functions
All objects and controls	SetRedraw (Supported for DataWindow controls with this behavior change: setting this method to false prevents the DataWindow inner control from being redrawn, but does not prevent the DataWindow control itself from being redrawn.)
Application objects	SetTransPool Obsolete method: SetLibraryList
DataStore	CopyRTF, CreateFrom, GenerateResultSet, GetStateStatus, InsertDocument, PasteRTF, ResetInk, SaveInk, SaveInkPicture, SetCultureFormat, and SetHTMLAction Obsolete method: GenerateHTMLForm
DataWindow	GenerateResultSet, GetDataLabelling, GetSeriesLabelling, GetStateStatus, OleActivate, SetCultureFormat, SetDataLabelling, SetHTMLAction, and SetSeriesLabelling Obsolete methods: DBErrorCode, DBErrorMessage, GenerateHTMLForm, GetMessageText, GetSQLPreview, and GetUpdateStatus

Object or control	Unsupported functions
DataWindowChild	OleActivate and SetCultureFormat Obsolete methods: DBErrorCode, DBErrorMessage, GetSQLPreview, and GetUpdateStatus
Graph (Graph controls are available in the DataWindow painter only.)	GetDataLabelling, GetSeriesLabelling, SetDataLabelling, and SetSeriesLabelling
RichTextEdit	GetSpacing, PrintEx, and Scroll InsertDocument and SaveDocument work with RTF and TXT formats only.
UserObject	AddItem, DeleteItem, EventParmDouble, EventParmString, and InsertItem
Window	ArrangeSheets, CloseChannel, ExecRemote, GetCommandDDE, GetDataDDE, GetDataDDEOrigin, GetRemote, OpenChannel, RespondRemote, SetDataDDE, SetRemote, StartHotLink, StartServerDDE, StopHotLink, and StopServerDDE

XAML

eXtensible Application Markup Language (XAML) allows you to code an applications' presentation aspects separately from the business logic that you code in the PowerBuilder .NET painter Script views.

Changes that you make in a painter Layout view (Design pane) are reflected in the XAML editor code, and changes you make in the XAML editor can be seen in the Design pane. You can view the XAML editor for a painter only when the Design pane is also open, although you can collapse either of these panes and alternately display the XAML editor or the Design pane by selecting the corresponding tab.

There are many Web resources and books available for learning XAML.

AutoWidth for User Object

Use XAML to set the Width parameter of a user object to "auto."

On the innerControl, set:

```
OpenUserObject(uo, 10, 10)
uo.InnerControl.Width = System.Double.NaN
```

CLS Compliance in PowerBuilder

PowerScript is compliant with Common Language Specification (CLS) rules and restrictions.

The CLS ensures that components developed in one language can be fully accessible to any other .NET language. It describes a set of commonly used features that serve as the minimum language requirement for any .NET language.

With PowerBuilder .NET, you can create extensions to other CLS-compliant languages and deploy components that can be consumed by any CLS-compliant application.

CLS Roles

The Common Language Specification (CLS) defines three roles for CLS-compliant applications: framework, consumer, and extender.

Enhancements to the PowerScript language enable PowerBuilder .NET to support a CLS-extender role.

The WPF Window Application targets in PowerBuilder .NET are CLS extenders because they allow you to consume and extend the CLS framework. The .NET Assembly and WCF Services targets in the PowerBuilder .NET IDE support a framework role for CLS compliance, since the components generated from these targets can be consumed by any other .NET language.

CLS role	Requirements
Framework	Guarantees interoperability across different .NET languages. A CLS-compliant .NET library is called a framework.
Consumer	A CLS consumer must be able to use CLS-compliant libraries, but cannot extend the CLS Framework. The CLS-compliant consumer must be able to: • Call any CLS-compliant method • Have a mechanism for calling methods for which names are keywords in the language • Create an instance of any CLS-compliant type • Read and modify any CLS-compliant field • Access any CLS-compliant property and event • Have a mechanism to use generic types and methods, and to access nested types

CLS role	Requirements
Extender	A CLS extender must allow users to consume and extend the CLS Framework. Everything that applies to CLS consumers also applies to CLS extenders, but extenders must also be able to: • Define new nongeneric CLS-compliant types that extend CLS-compliant base types • Implement any CLS-compliant interface • Place CLS-compliant custom attributes on appropriate elements

PowerBuilder Array Enhancements

PowerBuilder 12 introduces several array enhancements to the PowerScript language in support of its evolution as a CLS-compliant language.

These enhancements are available in all PowerBuilder .NET targets and in the .NET targets of PowerBuilder Classic.

Runtime Array Bounds

For PowerBuilder .NET targets, the PowerScript language allows you to declare an unbounded array at design time, and to dynamically create the array bounds at runtime.

Note: Although you could create and use an unbounded one-dimensional array in earlier versions of PowerBuilder, you could not set bounds for these arrays at runtime.

PowerScript lets you specify array bounds for one-dimensional or multidimensional unbounded arrays. The array bounds are created at runtime.

This example sets the indexes for an unbounded one-dimensional array to 2, 3, 4, and 5:

```
int arr[] //one-dimensional unbounded integer array
arr = create int[2 to 5]
```

This example sets the indexes of an unbounded two-dimensional array to 4 for one dimension, and 2, 3, 4, and 5 for the other dimension:

```
int arr[,] //two-dimensional unbounded integer array
arr = create int[4, 2 to 5]
```

Although you can set runtime bounds for unbounded arrays, you cannot reset design-time array bounds at runtime. For example, this code produces an error:

```
int a[3]
a = create int[5] //ERROR
```

Other useful array rules

- You can pass any kind of array to a .NET function as long as the array dimensions match the dimensions in the function's array parameter.
- Blob and byte arrays, and string and char arrays can be assigned to each other.

Returning an Array for a Function or Event

PowerBuilder .NET allows you to return an array datatype from a function or event.

In versions older than 12, PowerBuilder required you to map an Any datatype whenever a function or event returned an array datatype. You must still do this in PowerBuilder Classic. Although you can also map an Any datatype for an array datatype in PowerBuilder .NET, doing so is neither type-safe nor efficient.

To set an array datatype for a return value:

1. Open a painter for a PowerBuilder object. To return an array on a new:
   - Event – select the object name from the drop-down list at the upper left of the painter, then select **New Event** from the second drop-down list.
   - Function – select **Functions** from the upper left drop-down list.

   Note: You can also add an array datatype on a new function object that you define from the New dialog box.
2. For **Return Type**, enter the datatype followed by brackets. For multidimensional arrays, you can add commas inside the brackets.
3. Complete the remaining prototype fields available for the event or function, and add script for the new method.

   For a function script, include a return value that matches the array datatype you entered in the previous step.

This example shows PowerScript syntax for a function that returns an array of type integer:

```
public function integer[] ut_test_array ()
   integer ret[]
   ret = {1, 2}
   return ret
 end function
```

Jagged Array Support

For PowerBuilder .NET targets, the PowerScript array syntax allows you to declare jagged arrays, which have elements of differing dimensions and sizes, and are sometimes called "arrays of arrays."

This example creates an array with three elements, each of type int[ ], and initializes the three elements of the first array dimension with references to individual array instances of varying lengths in the second array dimension:

```
int arr[][]
arr = create int[3][]
arr[1] = create int[5]
arr[2] = create int[20]
arr[3] = create int[10]
```

.NET System.Array Support

PowerScript arrays include all the functionality of the .NET System.Array type.

All the public properties, and public static and instance methods, of the .NET System.Array type can be accessed by any PowerBuilder .NET application or component. This .NET functionality enables you to get the number of dimensions in an array.

You can continue to use PowerScript methods to determine the lower bounds and upper bounds of a particular dimension, or to get and set values of array items, but you can also use .NET array object methods. The System.Array methods and properties are visible in the Solution Explorer, and in the Script Editor with IntelliSense.

BitRight and BitLeft Operator Support

PowerBuilder 12 includes support for bitwise right-shift and left-shift operators.

BitRight operator

The right-shift operator shifts the first operand to the right by the number of bits specified in the second operand. This works the same as the C# “>>” operator.

The shift to the right can be arithmetic or logical, depending on the datatype of the first operand:

- Arithmetic – if the first operand is an signed integral datatype, high-order empty bits are set to the sign bit.
- Logical – if the first operand is an unsigned integral datatype, high-order bits are filled with zeros.

BitLeft operator

The left-shift operator shifts the first operand to the left by the number of bits specified in the second operand. This works the same as the C# "<<" operator.

The high-order bits of the first operand are discarded and the low-order empty bits are filled with zeros. Shift operations never cause overflows. The shift amount to the left depends on the bit quantity of the first operand datatype. For example:

- 32-bit quantity – if the first operand is a long or ulong datatype, the shift amount is given by the low-order five bits of the second operand. The maximum left shift is 0x1f, or 31 bits.
- 64-bit quantity – if the first operand is a longlong or ulonglong datatype, the shift amount is given by the low-order six bits of the second operand. The maximum left shift is 0x3f, or 63 bits.

Operator precedence

Bitwise shift operations are performed before relational (=, >, <, <=, >=, <>), negation (NOT), and logical (AND, OR) operations. They are performed after grouping, unary, exponentiation, arithmetic (multiplication, division, addition, subtraction), and string concatenation operations.

Inheritance from .NET System.Object

In PowerBuilder 12.5, the .NET System.Object type is the ancestor type of the PowerObject datatype.

Because the PowerObject datatype is the base class of all PowerScript object types, you can invoke the public members of the .NET System.Object type from any PowerBuilder object.

This example calls the **ToString** method that a PowerObject inherits from System.Object:

```
PowerObject po
String str
Po = create PowerObject
Str = Po.ToString()
```

You can assign any PowerBuilder type to the System.Object type.

This example assigns an integer type to a System.Object instance:

```
System.Object o
   Int i = 1
   o = i
```

Declaring a Namespace

You can specify a namespace for these PowerBuilder objects: custom and standard class objects, custom, standard, and external visual objects, windows, menus, structures, functions, interfaces, and enumerations.

Declare a namespace in the Declare Namespace\Usings window of the painter for any PowerBuilder object that supports namespace declarations.

You can also add using namespace directives in the Declare Namespace\Usings window. Using directives allow you to refer to a named type by its simple name rather than its compound name, which includes a namespace prefix.

1. Open a painter for a PowerBuilder object.
2. Select **Declare** from the drop-down list at the upper left of the painter, then select **Namespace\Usings** from the second drop-down list.

   Note: If it is available, you can click the Namespace\Usings tab at the bottom of the object painter instead of selecting from the drop-down lists.
3. In the **Namespace** field, enter the namespace declaration.

   The namespace declaration applies only to the selected object and any objects that inherit from the selected object.

   Skip to step 5 if you do not want to include a Using directive in the source code for the current object.
4. Click **Add** to open the Select Namespace dialog box and:
   a) Select an assembly or other item from the Assemblies list at the bottom of the dialog box.
   b) Select a namespace from the list of namespaces at the top of the dialog box, and click **OK**.
      The namespace you selected appears in the Using field of the object painter.
   c) Click **Add** again to include as many using namespace directives as you need.
5. Save the object with its declared namespaces.

This example shows a namespace declaration as it would appear in source code after entering "sybase.pb.ns" for the namespace name and "dir1" and "dir2" for using namespace directives:

```
namespace
namespace sybase.pb.ns
using dir1
using dir2
end namespace
```

Access Order with Unqualified Names

By declaring and using namespaces, you can qualify objects that would otherwise have the same names. Once declared, you can call these objects without using their qualified names.

PowerBuilder .NET uses the following algorithm to determine which object to call when multiple objects have the same names in a window that belongs to a declared namespace, and that includes a Using directive for a different namespace:

1. The object with the same namespace as its window or container object.
2. The object without a namespace.
3. The object with a different namespace that is included in the Using directive of the window or container object.

Syntax for Returning Namespace Names

PowerBuilder .NET provides an overload of the PowerScript **ClassName** system function that allows you to return the namespace of an object along with its class name.

The syntax for the **ClassName** function that can return a namespace name is:

```
string ClassName{any variable, boolean b_namespace )
```

Argument	Description
variable	The name of the object for which to return a class name.
b_namespace	Boolean argument indicating whether or not to include the namespace in the return value. Values are: • True – returns the namespace with the object name. • False – does not return the namespace with the object name.

The format of the returned value when *b_namespace* is true is:

```
nameSpace.className
```

Defining an Interface

An interface specifies the members of a class that must be supplied by any class that implements the interface. In PowerBuilder .NET, you can create interfaces from the New dialog box.

Because PowerScript has no concept of abstract classes, all methods, properties, events, and indexers defined in PowerBuilder .NET interfaces must be implemented by a PowerBuilder object class.

1. Select **File > New > PBObject > Interface** and:
   - Click **Next** to open and scroll through the wizard.
     Provide a name for the interface and select the library where you want to save the interface. On the next wizard page, you can select a namespace and using namespace directives for the new interface, and you can declare additional interfaces that must be implemented by objects implementing the new interface.
   - Click **Finish** to open the Interface painter.
     If you click Finish without providing a name for the interface, the Interface painter displays `Untitled` as the temporary name for the interface.
2. Make sure the interface name (or `Untitled`) appears in the drop-down list at the upper left of the Interface painter.
3. From the second drop-down list at the top of the Interface painter, select **New Event** to add events to the interface, and enter values and parameters for each new event.
4. From the first drop-down list, select the appropriate option to add functions, indexers, or properties to the interface, or to declare a namespace or another interface.

   A new painter window opens for the item you select. You may need to select an appropriate item (New Function, New Indexer, New Property, Namespace/Usings, or Interfaces) from the second drop-down list in the painter.

   Add as many functions, indexers, properties, or declarations as required for the interface.
5. Save the interface.
   If you did not provide a name for the interface in the wizard, the Save Interface dialog box prompts you for an interface name and the library in which to save it.

Implementing an Interface

To use the functionality of an interface, create a class that implements the interface, or derive a class from one of the .NET Framework classes that implements the interface.

These types of PowerBuilder .NET objects can implement interfaces: custom class, standard class, custom visual, external visual, standard visual, window, and menu objects.

If an object class implements two interfaces that contain a member with the same signature, implementing that member in the class implements that member for both interfaces. Naming

or identity conflicts are resolved by including the name of the interface with the name of the class member, separated by a dot.

Note: .NET rules do not allow you to directly call an identical member of two interfaces on a class that implements those interfaces. You must use a variable of one of the implemented interface types to call the member. If you try to call the member on the implementing class, PowerBuilder .NET displays a compilation error.

1. Open the painter for an object that can implement interfaces.
2. Select **Declare** from the drop-down list at the top left of the painter, then select **Interfaces** from the second drop-down list.
3. Right-click in the Interface list box and select **Add Interface**.
4. From the Select Interface dialog box, select the interface to implement and click **OK**.
5. Write scripts for all interface events, functions, .NET properties, and indexers.
6. Save the object with its interface implementations.

   You can view the object and the interfaces it implements in the the PB Object Outline view.

Deleting a Declared Interface

Delete a declared interface from the object painter for the object implementing the interface.

When you delete an interface, you can keep or delete the interface functions, indexers, and properties.

1. Open the painter for the object with a declared interface that you want to delete.
2. Select **Declare** from the drop-down list at the top left of the painter, then select **Interfaces** from the second drop-down list.
3. Right-click the interface to delete and select **Delete**.
4. In the Delete Interface dialog box, unselect any checked functions, indexers, or properties you want to keep.
5. Click **OK**.
   If you open the PB Object Outline view, you still see the functions, indexers, or properties you chose to keep.

System Interface Syntax

PowerBuilder .NET provides system interfaces that include implementations for all the methods of DataWindow controls and DataStores. You can use these system interfaces in your WPF applications or components.

You can work with system interfaces instead of individual DataWindow, DataStore, or Picture controls. Because these controls automatically implement system interfaces, you can pass the interface rather than the control, and directly call methods on the interface.

You need not declare system interfaces, or add implementations for their methods.

IDataWindowBase and its children

The IDataWindowBase system interface includes all the methods common to DataWindows and DataStores. It also is the base interface of three other system interfaces: IDataWindowControl, IDataWindowChild, and iDataStore.

You can assign any variable with a DataWindow, DataStore, or DataWindowChild datatype to the IDataWindowBase system interface. This allows you to take advantage of the interface's built-in polymorphism, so you need not duplicate code or check whether the assigned variable is a DataWindow or DataStore. The inherited system interfaces are slightly more restrictive, but can be assigned variables with the datatype indicated in this table:

System interface	Variable datatype to which it can be assigned
IDataWindowBase and IDataWindowControl	DataWindow
IDataWindowBase and IDataWindowChild	DataWindowChild
IDataWindowBase and iDataStore	DataStore

This syntax defines a function that determines the number of rows in a DataWindow or DataStore after a filter is applied:

```
                public function long f_filter (IDataWindowBase
idw_base, string as_filter);
                long ll_rows

                idw_base.SetFilter (as_filter)
                idw_base.Filter ()

                return idw_base.RowCount ()
                end function
```

You can then call this function in a script, passing in a valid DataWindow or DataStore and the value you want to use as a filter. This script filters a DataStore on a department ID of 100:

```
                DataStore ldw_emps
                   ldw_emps = Create DataStore
                   ldw_emps.DataObject = "d_myDWO"
                   ldw_emps.SetTransObject (SQLCA)    // Assumes a
connected transaction
                   ldw_emps.Retrieve ()

                   this.f_Filter (ldw_emps, "dept_id = 100")
```

IPicture

PowerBuilder .NET also provides an IPicture interface that exposes all the common methods of the PowerBuilder picture control. This example calls the **SetPicture** and **Draw** methods on an object that inherits from the IPicture interface:

```
IPicture ip = p_1
ip.SetPicture (myPictureBlob)
ip.Draw (x, y)
```

The IDataWindowControl and IDataWindowChild interfaces also use the IPicture system interface to set the *picturename* parameter in the **SetRowFocusIndicator** method of a DataWindow or DataWindowChild control.

Inheriting from a .NET Class

You can create a nonvisual user object that inherits from a .NET class in the same way that you create an object, visual or nonvisual, that inherits from a PowerBuilder object.

Prerequisites

Before you can create an object that inherits from a .NET class in an assembly, you must add the assembly as a target reference.

Task

To inherit from a .NET class defined in an assembly listed in the target References folder:

1. Select **File > Inherit** from the PowerBuilder .NET menu, or click **Inherit** in the IDE toolbar.

   Note: Alternatively, you can expand the node for an assembly in the References folder of your target to show the class you want to inherit from, right-click the class, select **Inherit From**, and skip to step 6.
2. In the Inherit from Object dialog box, select the target where you want to add the new object.
3. Select **Classes** or **All Objects** as the object type.
4. From the Libraries list, select an assembly containing the class you want to inherit from.
5. From the Objects list, select the class you want to inherit from and click **OK**.
6. In the Object painter, implement all abstract and virtual functions, interfaces, events, indexers, and .NET properties defined in the base class.
7. In the new object, add interfaces, functions, events, indexers, .NET properties, and parameterized constructors as for any other object.

   Note: By default, the new nonvisual user object extends the default (parameterless) constructor of its base .NET class. However, if the base .NET class has only parameterized constructors, you must manually call one of these constructors from a constructor of the new object:

```
super::Constructor (parameters)
```

Syntax to Support Inheritance from a .NET Class

PowerScript allows you to create nonvisual objects that override all virtual and abstract methods, properties, indexers, and events defined in .NET base classes and interfaces.

This example defines a PowerScript nonvisual object that inherits from a .NET class:

```
global type NVO from DotNetClass1, ISub1
end type
```

You can use the same syntax for calling PowerScript functions and events to call the functions and events of a .NET class:

```
{ objectname.} { type } { calltype } { when } name
( { argumentlist } )
```

This table describes the arguments used in function or event calls. All arguments except the function or event name are optional.

Argument	Description
objectname	The name of the object where the function or event is defined, followed by a period or the descendant of that object, or the name of the ancestor class of the object followed by two colons.
type	A keyword specifying the method type you are calling. Values are: • FUNCTION (default) • EVENT
calltype	A keyword specifying when PowerBuilder looks for the function or event. Values are: • STATIC (default) • DYNAMIC
when	A keyword specifying when the function or event should execute. Values are: • TRIGGER (default) – execute immediately. • POST – put in the object's queue and execute after other pending messages have been handled.

Argument	Description
name	The name of the function or event to call.
argumentlist	The values to pass to the function or event. Each value must have a datatype that corresponds to the declared datatype in the function or event definition or declaration.

Members in an Inherited .NET Object

You cannot change sealed members (functions, indexers, properties, or events) of an inherited .NET object.

Adding a Parameterized Constructor

In PowerBuilder .NET, you can define parameterized constructor events that overload the default constructor event, and instantiate the object with the arguments that you assign in the overriding event prototypes.

Parameterized constructor events are supported on all custom and standard class nonvisual user objects. If no arguments are passed in the call to instantiate a user object, the object's default constructor event is triggered instead of the parameterized constructor.

To create and invoke a parameterized constructor:

1. Open the object painter for an object that supports parameterized constructor events.

   To create a new object with a parameterized constructor, double-click the object type in the New dialog box and follow the same steps as for an existing object.
2. Make sure the object name or `Untitled` appears in the first drop-down list in the object prototype area, and select **New Constructor** from the second drop-down list.
3. Define an overloaded constructor event with the arguments you want to include.

   Add at least one argument to each new constructor event.
4. Create an object by passing in values for the arguments in the overloaded constructor event.

   This example creates an object with two arguments:

```
Obj obj = CREATE obj(1, 2)
```

Defining .NET Properties

.NET properties get or set a specific data value of an object, and allow you to process that value while enabling the actual member instance to remain private.

You can define .NET properties in application objects, windows, menus, and standard and custom user objects. PowerBuilder .NET object painters provide an easy way to script "getters" and "setters" for .NET properties.

1. Open the painter for the object in which you want to define .NET properties.
2. From the drop-down list at the upper left of the painter, select **Properties**.
   A property signature or prototype area appears between the main item selection area and the scripting pane.
3. In the main item selection area, make sure New Property appears in the second drop-down list.

   Before a .NET property is defined for the current object, New Property is the default selection. If .NET properties are already defined, the default selection is the most recent .NET property viewed for the current object.
4. In the drop-down list at the right of the prototype area, select:

   - **get** – to add a getter method.
   - **set** – to add a setter method.
5. In the drop-down list at the left of the prototype area, select the access scope for the .NET property.

   Available selections are public (default), protected, and private.
6. In the second drop-down list of the prototype area, select the return type for the .NET property.
7. In the third drop-down list of the prototype area, enter a name for the .NET property.
8. In the scripting pane, enter the processing code for the .NET property getter or setter method.

   If you do no other processing with the values that you get or set, you typically:
   - For getters – enter `return` followed by a variable of the selected return type.
   - For setters – set the property or a variable to the context keyword `value`.
9. To script:

   - The other accessor type (getter or setter) for the current .NET property, repeat this procedure from step 4, making the other accessor type selection from the one you previously made.
   - A new .NET property for the current object, repeat this procedure from step 3.
10. Save the object.

Defining Indexers

Indexers get or set values stored in an object. They are similar to .NET properties, except that they take parameters, and can be overloaded.

You can define indexers in Application objects, windows, menus, and standard and custom user objects. PowerBuilder .NET object painters provide an easy way to script "getters" and "setters" for indexers.

1. Open the painter for the object in which you want to define indexers.
2. From the drop-down list at the upper left of the painter, select **Indexers**.
   An indexer signature or prototype area appears between the main item selection area and the scripting pane.
3. In the second drop-down list of the main item selection area, make sure New Indexer appears.

   Before an indexer is defined for the current object, New Indexer is the default selection. If indexers are already defined, the default selection is the most recent indexer viewed for the current object.
4. In the drop-down list at the right of the prototype area, select:

   - **get** – to add a getter method.
   - **set** – to add a setter method.
5. In the drop-down list at the left of the prototype area, select the access scope for the indexer.

   Available selections are public (default), protected, and private.
6. In the second drop-down list of the prototype area, select the return type for the indexer.
7. Enter required information for indexer arguments:
   a) From the Argument Type drop-down list, select a datatype.
   b) In the Argument Name text box, enter an argument name.
   c) To add another argument, right-click in the prototype area, select **Add Argument**, and repeat this step as many times as required.

   Note: The Script Editor shows dimmed fields for the indexer name and the argument "pass by" fields. This is because the indexer name is always "this" (meaning the current object), and you can pass indexer arguments only by value.
8. Enter the processing code for the indexer getter or setter in the scripting pane.
   The following code sample is from a getter script for an indexer that has a single integer argument, nIndex, and returns a string array or a simple string:

```
if nIndex >0 or nIndex <= names.Length then
        return names[nIndex]
else
```

```
                return "null"
        end if
```

The code sample for a setter script for the same indexer looks like this:

```
        if nIndex >0 or nIndex <= names.Length then
                names[nIndex] = value
        else
                names[nIndex] = "no name"
        end if
```

9. To script:
   - The other accessor type (getter or setter) for the current indexer, repeat this procedure from step 4, making the other accessor type selection from the one you previously made.
   - A new indexer for the current object, repeat this procedure from step 3.
10. Save the object.

Creating a Global User-Defined Enumeration

In PowerBuilder .NET, you can create global enumerations in the Enumeration painter or in a script.

Global enumerations can be used by all objects in your application or component target. Use the Enumeration painter to define global enumerations for your applications:

1. In the New dialog box, select **Enumeration** under the PB Object node and click **Finish**.
2. If you want the values for each item in the enumeration to be bit fields, select the **Flags** option.
3. In the Name column, enter a name for each item you want to add to the enumeration.
4. (Optional) Add values and comments for each item.
5. Select **File > Save**, provide a name for your global enumeration, and click **OK**.

Syntax for User-Defined Enumerations

Use PowerScript to create custom enumerations for PowerBuilder .NET targets.

This example creates a user-defined enumeration named MyEnum:

```
global type MyEnum enumerated
    item1,
    item2 = 3
end type
```

The syntax to access an enumeration constant is:

```
[[namespacename.]enumerationType.]enumerationEntryName!
```

Use the enumeration type to access the items of user-defined enumerations. For example:

```
Enum me
me = MyEnum.item2!
```

This syntax helps make the code readable and avoids naming conflicts.

If you do not provide an enumeration type, the enumeration is assumed to be a system-defined type and PowerBuilder .NET tries to find it in PowerBuilder system-defined enumerations. If the enumeration type exists, PowerBuilder .NET tries to find the enumeration type first and then find the entry for that enumeration type.

Creating a Local User-Defined Enumeration

In PowerBuilder .NET, you can create local enumerations for an object in an Enumeration painter that you open from the object painter.

To define a local enumeration using the PowerBuilder .NET user interface, first open an object painter that has a local Enumeration painter attached. You can define local enumerations for these kinds of objects: custom and standard class objects, custom, standard and external visual objects, windows, and menus.

In the Enumeration painter, you can specify entry name, value, and comments for each item that you enter for an enumeration type.

1. In the Solution Explorer, right-click the object for which you want to add a local enumeration and select **Open**.
2. From the drop-down list at the upper left of the object painter, select **Enumerations**.
3. If it is not already selected, select **New Enumeration** from the second drop-down list.
4. In the Enumeration text box, enter a name for the new enumeration.
5. In the Name column, enter a name for each item you want to add to the enumeration.
6. (Optional) Add values and comments for each item in the painter.
7. Save the object with its new enumeration.

Consuming a .NET Delegate

In PowerBuilder .NET, you can invoke .NET delegates from PowerScript code. Delegates allow you to treat functions as entities that can be assigned to variables and passed as parameters.

A delegate type represents references to methods that have a particular parameter list and return datatype.

The PowerBuilder .NET IDE exposes all the visible delegates in the .NET assemblies referenced by the current target in the Solution Explorer or object browser.

1. Import an assembly containing a .NET delegate to a PowerBuilder .NET target.
2. Determine whether to invoke the .NET delegate synchronously or asynchronously.
3. Write PowerScript code to assign a function or functions to a variable that has the .NET delegate type.

Next

If you invoke the delegate asynchronously, determine whether to join the results immediately after the invoked thread completes its execution, after a callback function is triggered, or as the result of polling that allows other processing to complete even after the child thread has finished running.

You can also decide whether to use multicasting, which uses a function chain that passes arguments by value or reference from one function to another in a single invocation of the .NET delegate.

Syntax for Consuming .NET Delegates

In PowerBuilder .NET, you can use PowerScript to invoke delegates in imported assemblies either synchronously or asynchronously.

Delegate declaration

This C# syntax declares delegate types named Func and FuncAsync in a .NET assembly that you import to your PowerBuilder .NET target.

.NET class declaration in C# (in an external assembly):

```
using System;
delegate double Func(double x);
delegate double FuncAsync(double[] a);
```

The example for the synchronous use case that follows consumes the Func delegate, and the example for the asynchronous use case consumes the FuncAsync delegate.

Synchronous use case

This PowerScript syntax uses the delegate type Func from the external DLL that you import to your target. The delegate signature requires a function that takes a double datatype and returns a value with a double datatype.

```
public function double[]  of_apply  (double a_v[], Func
a_f)
        double result[]
        for  i = 0 to i < a_v.Length step 1
             result[i] = a_f (a_v[i])
         next
             return result
   end function
```

```
                public function double of_multiply(double
a_x)
                        return a_x * factor;
                end function
```

```
                 public subroutine of_test()
                      double a[] = {1.0,2.0,3.0,4.0,5.0}
                      double doubles[] =  of_apply(a, of_multiply)
                end subroutine
```

In the above example, the **of_test** method has a vector of double values. The vector is passed to the function **of_apply** which calls the function **of_multiply** for each value in the vector using the delegate variable *a_f*. This is possible since **of_apply** takes a delegate as an argument (*a_f*).

You can use the above method to apply different algorithms to the same data. Instead of directly calling **of_multiply**, you can include additional functions with **of_apply** and the delegate variable, as long as the additional functions process a single value of the double datatype and return a value of the double datatype.

Asynchronous use case

This PowerScript syntax uses the delegate type FuncAsyc from the external DLL that you import to your target:

```
                public function double of_sum( double a_v[] )
                     double sum
                     for  i = 0 to i < a_v.Length step 1
                          sum = sum + a_v[i]
                      next
                          return sum
                end function
```

```
                 public subroutine test()
                      double a[] = {1.0,2.0,3.0,4.0,5.0}
                      FuncAsync D = of_sum
                      System.AsyncCallback nullValue
                      integer stateValue
                      System.IAsyncResult ar = D.BeginInvoke( a,
nullValue, stateValue )
                      //background thread starts
                      ...
                      double result = D.EndInvoke( ar ) // wait for
thread to return
                      result (join)
                end subroutine
```

The above example corresponds to a "wait-until-done pattern." You can also use asynchronous processing in a "polling pattern," where the invoked thread is polled to

determine whether it is finished, or in a "callback pattern," where a callback function is called when the thread has finished executing.

The code between BeginInvoke and EndInvoke runs in the main thread, but the computation of the sum has its own thread from the .NET thread pool. When EndInvoke is called, it acts as a Join operation and completes when the sum is returned by the child thread.

Note: To maintain thread safety, .NET prevents child threads from directly getting or setting property values in a form or its child controls. You can read from and write to the user interface only from the main UI thread.

Support .NET Events

You can invoke .NET events from PowerScript code.

Since .NET events are a kind of delegate variable, you can use delegates to consume .NET events in PowerScript, as long as the delegate has the same signature as the .NET event. You can connect single or multiple methods to a .NET event in Powerscript. The method can be a PowerBuilder global function, an instance function of a PowerBuilder object, or an instance or static function of a .NET object.

When you consume a .NET event, you do not declare it in PowerBuilder code, because it is not a type. Instead, declare a variable or parameter and assign it a .NET event, using the Powerbuilder += operator.

Dynamically Connecting to a .NET Event in PowerScript

```
// Define a PowerBuilder event that has the same signature as the.NET
Clicked event
// (defined in the .NET system.windows.controls.button control).
Event  OnClick1(system.object sender, RouteEventArgs e)

// Connect the PowerBuilder event to the .NET Clicked event.
System.Windows.Controls.Button cb1
Cb1  = create System.Windows.Controls.Button()
Cb1.clicked += onClick1
```

Consume .NET Events

You can dynamically connect PowerBuilder events or functions to a .NET event of a third-party control using PowerScript.

There are two ways to make the connection: using Delegate += or .NET AddHandler function.

Declare Event or Function

First, declare a PowerBuilder event or function with parameters that are the same as the .NET event.

```
event MyDockWindowClosing(object sender, RoutedEventArgs e)
{
CancelSourceRoutedEventArgs cancelArg = e;
cancelArg.Cancel = true;
```

```
mle_1.text += e.OriginalSource;
}
```

or

```
function wf_DockWindowClosing(object sender, RoutedEventArgs e)
{
CancelSourceRoutedEventArgs cancelArg = e;
cancelArg.Cancel = true;
mle_1.text += e.OriginalSource;
}
```

Delegate +=

```
dockWindow.Closing += MyDockWindowClosing
```

or

```
dockWindow.Closing += wf_DockWindowClosing
```

.NET AddHandler function

```
System.Windows.RoutedEventHandler reh
reh = this.wf_DockWindowClosing //or reh = this.MyDockWindowClosing
System.Windows.RoutedEvent re
re = DevComponents.WpfDock.DockWindow.ClosingEvent
w_frame.InnerControl.AddHandler(re, reh)
```

Syntax for Consuming Generic Classes

Generics are classes, structures, interfaces, and methods with placeholders for the datatypes that they store or use.

A generic collection class can use a datatype parameter as a placeholder for the types of objects that it stores or returns. A generic method can also use its datatype parameter as a type for any of its formal parameters (arguments).

In PowerBuilder .NET, you can consume .NET generic classes and methods that conform to CLS generic rules. However, you cannot define a generic type or override generic methods.

This example calls the generic **SortedList** method:

```
System.Collections.Generic.SortedList<string, integer> teams
teams = create System.Collections.Generic.SortedList<string, &
   +integer>
teams["DEV"] = 30
teams["QA"] = 25
```

The generic parameter type can be any type, including a primitive type, .NET class type, PowerBuilder user-defined type, or a nested generic type. Inner generic types are also supported. For example:

```
MyDll.GenericClass2<MyDll.Class1>.GenericClass2_1<System.String> o1
```

```
o1 = CREATE MyDll.GenericClass2<MyDll.Class1>.GenericClass2_1 &
   +<System.String>
```

The syntax for calling functions of .NET classes is also available for the .NET generic type. For example:

```
      string p1 = "p1", p2 = "p2"
      MyDll.GenericClass1<MyDll.Class1> o2
      o2 = CREATE MyDll.GenericClass1<MyDll.Class1>

      //Calling a normal function of the .NET generic type
      //with the Dynamic keyword
      o2.Dynamic GenericTest11(p1, p2)

      //Calling a generic function of the .NET generic type
      //with the Dynamic keyword
      o2.Dynamic GenericTest12<MyDll.MyClass2>(p1, p2)

      //Calling a generic function of the .NET generic type
      //with Dynamic and Post keywords
      o2.Dynamic POST GenericTest11<MyDll.MyClass2>(p1, p2)

      //Calling a generic function of the .NET generic type
      //with Post and Dynamic keywords
      o2.POST Dynamic GenericTest12<MyDll.MyClass2>(p1, p2)
```

Using an Enumerator to Traverse a .NET Collection

Use an enumerator to traverse the contents of a .NET collection.

For example:

```
IEnumerator myEnum
CommandBar myCommandBar

myApplication =
Microsoft.VisualBasic.Interaction.CreateObject("VisualStudio.DTE.
9.0","");
myCommandBars = myApplication.CommandBars ;
myEnum = myCommandBars.GetEnumerator();
do while myEnum.MoveNext()
    myCommandBar = myEnum.Current
    MessageBox ( "CommandBar", myCommandBar.Name )
loop
```

Enhancements to .NET Component Projects

When you generate .NET assembly components from PowerBuilder .NET projects, you can take advantage of CLS-compliant features of PowerBuilder .NET that are not available in PowerBuilder Classic.

For example, in PowerBuilder .NET, you can include jagged arrays as parameter types of public methods in .NET Assembly targets.

If you set the CLSCompliant value of a .NET component project to true, the PowerBuilder to .NET compiler (**pb2cs**) generates the CLSCompliantAttribute (true) attribute for the component, and the C# compiler applies the CLS rules to the public methods of the component. If you set this value for a component that is not CLS compliant, the compiler issues the appropriate warnings.

DataWindows

Build DataWindow objects to retrieve, present, and manipulate data in your applications.

DataWindows in PowerBuilder .NET

Use DataWindow objects to retrieve, present, and manipulate data from a relational database or other data source. In PowerBuilder .NET, WPF DataWindows use fully managed code.

DataWindow objects have knowledge about the data they are retrieving. You can specify display formats, presentation styles, and other data properties so that users can make the most meaningful use of the data.

Using DataWindow Objects in PowerBuilder .NET

You can use DataWindow objects in WPF Window applications.

Before you can use a DataWindow object, you must build it using the DataWindow painter.

This procedure describes the overall process for creating and using DataWindow objects. For complete information about using DataWindow objects in different kinds of applications and writing code that interacts with DataWindow objects, see the *DataWindow Programmers Guide*.

1. Create the DataWindow object by using one of the DataWindow wizards in the DataWindow group of the New dialog box.

   The wizard helps you define the data source, presentation style, and other basic properties of the object. Define additional properties for it, such as display formats, validation rules, and sorting and filtering criteria.
2. Place a DataWindow control in a window or user object.

   The control enables your application to communicate with the DataWindow object you created in the DataWindow painter.
3. Associate the DataWindow control with the DataWindow object.
4. In the Window painter, write scripts to manipulate the DataWindow control and its contents.
   For example, use the PowerScript **Retrieve** method to retrieve data into the DataWindow control.

   You can write scripts for the DataWindow control to deal with error handling, sharing data between DataWindow controls, and so on.

DataWindow Differences Between PowerBuilder Classic and PowerBuilder .NET

In most ways, the DataWindow in PowerBuilder .NET behaves like the DataWindow in PowerBuilder Classic; however there are some differences.

Changes to the DataWindow object

The DataWindow was rewritten for PowerBuilder .NET using managed code.

To open the Preview, Column Specifications, and Data views, click **View > DataWindow Painter Windows**. The DataWindow object and the data from the database appear in the Preview view only when you click Retrieve.

The Control List has been replaced with the Document Outline, which is a hierarchical list of controls, organized within the bands of the DataWindow object (header, detail, summary, and footer).

In PowerBuilder .NET, you can add custom controls to DataWindow objects.

Closing the Layout view is the same as closing the DataWindow painter in PowerBuilder Classic.

Cascading menu item in View menu	Menu item
Other Windows	Command Window, Document Outline, Task List, Error List, Start Page, Web Browser
DataWindow Painter Windows	Preview, Column Specifications, Data **Note:** Import/Export Template is not available in PowerBuilder .NET version 12.5.

Unsupported features

These features are not supported for DataWindow objects in PowerBuilder .NET.

- EMF and WMF image formats are not supported by WPF
- OLE DataWindow presentation style and OLE controls are not supported by WPF

Feature	Unsupported
Expressions	AscA, CharA, FillA, LeftA, LenA, MidA, PosA, ProfileInt, ProfileString, ReplaceA, RightA
Import types	Dbase 2 & 3
Export types	Dbase 2 & 3, DIF, EMF, Excel, Excel 5, HTML Table, Powersoft Report, Sylk, WKS, WK1, WMF, XSL-FO

Feature	Unsupported
DataWindow object properties	CSSGen.*property*, Data.HTML, Data.HTMLTable, Data.XHTML, Data.XMLWeb, Data.XSLFO, Export.PDF.Distill.CustomPostScript, Export.PDF.Method, Export.PDF.XSLFOP.Print, Export.XHTML.TemplateCount, Export.XHTML.Template[ ].Name, Export.XHTML.UseTemplate, Font.Bias, HideGrayLine, HorizontalScrollMaximum2, HorizontalScrollPosition2, HorizontalScrollSplit, HTMLDW, HTMLGen.*property*, JSGen.PublishPath, JSGen.ResourceBase, Label.Sheet, OLE, Client.*property*, Picture.Blur, Print.ClipText, Print.Color, Print.Copies, Print.Duplex, Print.OverridePrintJob, Print.Page.Range, Print.Page.RangeInclude, Print.Paper.Source, Print.Preview.Rulers, Print.Quality, Print.Scale, ReplaceTabWithSpace, RichText.ControlsCharVisible, RichText.DisplayOnly, RichText.PictureFrame, RichText.ReadOnly, RichText.ReturnsVisible, RichText.RulerBar, RichText.SpacesVisible, RichText.TabBar, RichText.TabsVisible, RichText.WordWrap, ShowBackColorOnXP, ShowFocusRectangle, XHTMLGen.Browser, XMLGen.*property*, XSLTGen.*property*, Zoom Background.Brushmode: ScaledAngle brushmode is not supported in WPF Obsolete properties: Storage, StoragePageSize, Table.CrosstabData, Table.Data.Storage OLE is not supported: OLE.Client.Class, OLE.Client.Name

This table lists DataWindow objects and controls with unsupported properties.

DataWindow control	Unsupported properties
Bitmap	AccessibleDescription, AccessibleRole, HTML.*property*, TransparentColor
Button	AccessibleDescription, AccessibleRole, Font.CharSet, Font.Escapement, Font.Family, Font.Pitch, Font.Strikethrough, Font.Underline, Font.Width
Column	AccessibleDescription, AccessibleRole, CheckBox.3D, CheckBox.Scale, Compute, dddw.AutoHScroll, dddw.HSplitScroll, dddw.ShowList, dddw.UseAsBorder, ddlb.AutoHScroll, ddlb.ShowList, ddlb.UseAsBorder, ddlb.VScrollBar, Edit.AutoHScroll, Edit.AutoVScroll, Font.CharSet, Font.Escapement, Font.Family, Font.Pitch, Font.Width, HTML.*property*, Ink.AntiAliased, InkEdit.UseMouseForInput, LineRemove, MultiLine, RadioButtons.3D, RadioButtons.Scale, UseInternalLeading, Width.AutoSize

DataWindow control	Unsupported properties
Compute	AccessibleDescription, AccessibleRole, Font.CharSet, Font.Escapement, Font.Family, Font.Pitch, Font.Width, HTML.*property*, MultiLine, Width.AutoSize
Custom	AccessibleDescription, AccessibleName, AccessibleRole
Graph	AccessibleDescription, AccessibleRole, *property.DispAttr*.Font.CharSet, *property.DispAttr*.Font.Escapement, *property.DispAttr*.Font.Family, *property.DispAttr*.Font.Pitch, *property.DispAttr*.Font.Width
GroupBox	AccessibleDescription, AccessibleRole, Border, Font.CharSet, Font.Escapement, Font.Family, Font.Pitch, Font.Strikethrough, Font.Underline,
InkPicture	Ink.AntiAliased, InkPic.AutoErase, InkPic.HighContrastInk, InkPic.InkEnabled
Report	AccessibleDescription, AccessibleName, AccessibleRole, Criteria, Pointer
TableBlob	ClientName, OLEClass, Template
Text	AccessibleDescription, AccessibleRole, Font.CharSet, Font.Escapement, Font.Family, Font.Pitch, Font.Width, HTML.*property*
Tooltip	HasCloseButton

Other properties, functions, and methods not supported in PowerBuilder .NET are listed in *Unsupported Properties, Events, and Functions*.

Preview view

When Retrieve On Preview is enabled, the data appears in the Preview view when it is first opened. If the view is already open, however, you must click Retrieve to see the data.

Data View

If you make changes in the Data view, you must use Save twice to save the changes. First, save the changes in the Data view by selecting Save Changes, and then use **File > Save** to save the changes to the DataWindow object.

The context menu for the Data view in PowerBuilder .NET includes a Save Changes option.

SQL Data Sources

When you are choosing the data source for a DataWindow object, you can select SQL in the wizard and in the next step choose the type: Quick Select, SQL Select, Query, or paste SQL from clipboard.

New features

The property RightToLeft has been deprecated and is kept for backward compatibility. FlowDirection replaces RightToLeft. See *Right-To-Left Formatting*.

When you save content to an external file, the formatting is preserved in XPS documents but not in PDF documents.

Behavior Changes for DataWindow Objects

The .NET and WPF environments affect the behavior of many DataWindow features at runtme.

The following table highlights some of the important changes to the runtime behavior of DataWindow features in WPF applications:

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
Alignment	If a column is right aligned and is not wide enough to display all the text, the text at the left is not visible.	If a column is right aligned and is not wide enough to display all the text, the text at the right is not visible.
Background color for columns	• If the background color on a column is transparent, the color changes to black when the column gets focus. • If the background is a gradient, when it gets focus, the color appears solid.	• If the background color on a column is transparent, the color remains transparent when column gets focus. • If the background is a gradient, it does not change when the column gets focus.
Band rendering	• Detail band renders before the header and footer bands. • The detail band is truncated if there is not enough space to display the entire band. • The summary band may appear separate from the last detail row.	• Header and footer bands render first. If their heights are large, they may obscure the detail band. • The summary band is truncated if there is not enough space to display the entire band. • The summary band and last detail row appear together.
Brushmode, Angle	When the angle is within the ranges of 91-179 or 271-359, the height midpoint is the axis.	When the angle is within the ranges of 91-179 or 271-359, the top left corner is the axis.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
CanUndo and Undo behavior	If data is modified in a column and does not lose focus, call CanUndo and it returns true, then call Undo and the data is restored. Call CanUndo again, it returns true, then call Undo function, the data will be restored again.	If data is modified in a column and does not lose focus, call CanUndo and it returns true, then call Undo and the data is restored. Call CanUndo function again, it returns false, then call Undo function, the data is not restored.
CodeTable property after editing data	If you edit column data and then set the CodeTable property to no, text appears in the column.	If you edit column data and then set the CodeTable property to no, the data is restored.
Data view changes update Preview view	After editing content in the Data view and saving the changes, the Preview view does not display the edited content unless you close and reopen the DataWindow object.	The Preview view displays the content you edited in the Data view after you save the changes.
EditMask Date fields	If the user does not enter the full date, the system supplies the other values.	If the user does not enter the full date, the date is not accepted and the field does not populate.
Disabling DataWindow control	If the DataWindow control is disabled, there is no change in the appearance of the child controls.	If the DataWindow control is disabled, the child controls are disabled and dimmed.
DropDown event behavior	The first time a drop-down DataWindow is clicked, the DropDown event triggers and it opens. The second time it is clicked, the drop-down DataWindow closes without triggering the event. If there is no data, the DropDown event triggers but the drop-down DataWindow does not appear.	The first time a drop-down DataWindow is clicked, it opens. The second time it is clicked, the drop-down DataWindow closes. This happens whether the drop-down DataWindow contains data.
Drop-down DataWindows	If a drop-down DataWindow is opened, the MouseMove event may be triggered when you move the mouse.	If a drop-down DataWindow is opened, the MouseMove event is not triggered when you move the mouse.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
EditMask columns using code tables	• When a Date, Time, DateTime, or String column has a value that is not in the code table, the column displays 00/00/00 or 00:00:00. • If the user enters a value that does not exist in the CodeTable, when the column loses focus, the data is not accepted. • If the user enters a value that exists in the CodeTable but in a format that does not match the code table, when the column gets focus, the value appears. • For numeric and decimal datatypes, if the value does not exist in the CodeTable, the value still appears. The mask is effective when inputing data.	• If the value for a Date, Time, DateTime, or String column has a value that is not in the code table, the column displays nothing. • If the user enters a value that does not exist in the CodeTable, when the column loses focus, nothing appears. • If the user enters a value that exists in the CodeTable, when the column gets focus, the display value appears even if the input data format is not consistent with the data value format in the CodeTable. • For numeric and decimal datatypes, if the value does not exist in the CodeTable, the value still appears. The mask does not work when the user inputs data; when it loses focus, the mask works.
EditMask.Spin property	If a column uses a CodeTable and initially has a spin control that is then disabled, the CodeTable value is not shown in the column.	If a column uses a CodeTable and initially has a spin control that is then disabled, the CodeTable value appears in the column.
EditMask field in Properties view	If you double-click the EditMask field where it is divided into segments (such as date or time fields), only the segment is highlighted after the double-click.	Double-clicking in the EditMask field selects all of the content in this field.
Event sequence, Click and DoubleClick	For a column control, the event sequence is click, click, double-click. If the control has focus or the double-click occurs on a non-column control, the event sequence is click, double-click.	The events trigger sequence is always click, double-click, click.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
Event sequence, Click and GetFocus	When the DataWindow does not have focus and it is clicked, the event sequence is: GetFocus, Clicked.	When the DataWindow does not have focus and it is clicked, the event sequence is: Clicked, GetFocus.
Event sequence, change of focus	The event sequence for RowFocusChanging, RowFocusChanged, and ItemFocusChanged varies, depending on the operation. • When the DataWindow is reset before inserting a row, the event sequence is: RowFocusChanging, ItemFocusChanged, RowFocusChanged. • When inserting a row without resetting, the event sequence is: RowFocusChanging, RowFocusChanged, ItemFocusChanged.	The trigger sequence is always: RowFocusChanging, RowFocusChanged, ItemFocusChanged.
Event sequence, print events	If Print is called first, the event sequence is: PrintStart, PrintPage, PrintEnd, and other evens. If you call PrintCancel after Print, the DataWindow object prints. The Print function returns the value when the printing is complete.	PrintPage and PrintEnd can be triggered after other events that are triggered by the Print function. If you call PrintCancel after Print, the DataWindow object does not print. The Print function is only used to prepare a print document and add a print job ot the print manager. You can cancel the print job if the printing is not complete.
Event sequence, retrieve and resize events	The event sequence is: RetrieveStart, Resize, RetrieveEnd.	The event sequence is: RetrieveStart, RetrieveEnd, Resize
Event sequence, scrolling with PageUp and PageDown	The event sequence is: RowFocusChanging, ScrollVertical, RowFocusChanged.	The event sequence is: RowFocusChanging, RowFocusChanged, ScrollVertical.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
Export format	The format of date and time columns uses a fixed format.	The format of date and time columns uses the culture set for the runtime operating system.
Export Graph object	• When the Graph object is exported with the header, the header value is (NONE) • XLS format is supported	• When the Graph object is exported with the header, the header is the label text • XLS format is not supported
Format keywords for Date,Time, and Currency(7) datatypes	Uses static masks: dd/mm/yyyy, hh:mm:ss:fff, $#,###.## and ($#,###.##).	Uses masks based on the culture set for the runtime operating system.
GetText function	If the EditMask edit style uses a code table: • If the EditMask is the same as the CodeTable value's format, GetText function return value uses a fixed format for each datatype. • If the EditMask is not the same as the CodeTable value's format and it uses a date, date/time, or time datatype, GetText returns null. For numeric and decimal datatypes, GetText returns data with no format. If the EditMask edit style does not use a CodeTable: • For date, date/time, and time datatypes, the GetText function return value uses a fixed format for each datatype. • For numeric and decimal datatypes, GetText returns data with no formatting.	If the EditMask edit style uses a code table, the GetText return value uses the format that the code table value used. If the EditMask edit style does not use a CodeTable: • For date, date/time, and time datatypes, the GetText function return value uses the same format as the EditMask. • For numeric and decimal datatypes, GetText returns the data with no formatting.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
GraphCreate event	During Retrieve, GraphCreate is triggered twice. During InsertRow, DeleteRow, and Update, GraphCreate is triggered once.	Retrieve triggers GraphCreate once. InsertRow, DeleteRow, and Update do not trigger GraphCreate.
ItemError event	If the user enters a value in an invalid format and ItemError returns 2, PowerBuilder Classic sets it to a random valid value. The ItemChanged and ItemFocusChanged events are triggered.	When the user input is in an invalid format, PowerBuilder .NET changes it back to the original value. ItemChanged is not triggered and ItemFocusChanged is triggered.
KeyDown event	Pressing the Alt key does not trigger the DataWindow's KeyDown event.	Pressing the Alt key triggers the DataWindow's KeyDown event.
Mode expression function	Mode calculates the most frequently occuring value for a column.	Because of a new sorting algorithm, the value returned by Mode may be different.
Pen.Width property behavior	The Pen.Width is always measured in pixel units.	The Pen.Width uses the unit of measurement specified in the DataWindow.

Print function	If the Print.*FileName* of a DataWindow or DataStore is not empty, then:
	`dw_1.Print(false,false) //silent print`
	prints to file or a physical printer, depending on the default printer
	or
	`dw_1.Print(false,true) //show print dialog`
	• if the default printer is a physical printer, it prints a hard copy of the file
	• if the default printer is a virtual printer, is saves a file (of that printer's type) with the name entered in the file dialog
		When Print.*FileName* for a DataWindow or DataStore is not empty, either a silent print or show print dialog prints to an XPS file.
		PrintToFile was removed from the printer setting for security reasons. The Print dialog displays so the user can confirm the printing operation. When there is no user interaction, the DataWindow or DataStore uses SaveAs XPS file.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
PrintPage behavior	When the return code is set to 1, the page is not printed.	When the return code is set to 1, the page prints.
Prompt Before Printing property	If Prompt Before Printing is selected, the Print Preferences dialog appears. Closing the Print Preferences dialog continues the print job.	If Prompt Before Printing is selected, the Print dialog appears (and you can access the Print Preferences from there). Closing the Print dialog cancels the print job.
Protected columns	• In a Grid, if a protected column is selected, it gains focus and then loses focus. • In other presentation styles, the selected column gets focus and does not lose focus.	Select a protected column and it does not get focus. The current row and column get focus.
Read only behavior	When a read-only column gets focus, the cursor may appear.	When a read-only column gets focus, the cursor does not appear.
Retrieve argument	If the retrieve argument type is not correct, RetrieveStart and RetrieveEnd are not triggered.	If the retrieve argument is not correct, retrieving the DataWindow triggers RetrieveStart and RetrieveEnd.
RichText column, bullets or numbering	Setting the bullet or numbering does not change the column's indent.	Setting the bullet or numbering changes the column's indent.
RichText column, edit data and set AutoHeight	If a user enters data and does not accept the text, then sets the column's AutoHeight, the data is not restored.	If a user enters data and does not accept the text, then sets the column's AutoHeight, the data is restored.
RichText column, setting font properties	If text is not selected, and the cursor is placed in a word, any changes to the font properties affect only the cursor location.	If the cursor is placed in the middle of a word and changes are made to the font properties, the changes will be made to the entire word.
RichText column, TextStyle status	If the cursor is placed before text, the text's style may not display in the toolbar.	The text's style will display when the cursor is placed before it.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
RichText column, return character	The return character is ignored in the EditChanged and ItemChanged events' data arguments.	The return character is returned in the EditChanged and ItemChanges events' data arguments.
RichText column, select and delete content	Use the mouse to select all the content in a RichText column from the end to the start and delete it, the data becomes an empty string. No return character is appended at the end.	Use the mouse to select all the content in a RichText column from the end to the start (without selecting the final return character) and delete it, the data is not an empty string. If you do select and delete the final return character, it does return an empty string.
RichText column, set font property	When the cursor is at the start or end of the input field, and the user sets the font property then clicks in the field, the font property applies.	When the cursor is at the start or end of the input field, and the user sets the font property then clicks in the field, the font property does not apply. When the control loses and regains focus, the style at the cursor position resets to the adjacent character's style.
RichText columns, where data is stored by TX control and displayed using MS RichText control	The text is centered.	The text is left-aligned.
RichText input fields	• Copy and paste an input field, the pasted field looks like an input field but it does not actually access the data. • Copy and paste an input field into a text edit, just the text is copied. • Copy some of the text and paste it, only the selected text is pasted. • The input field can be partially selected with other text.	• Copy and paste an input field, the pasted field is an actual input field that can retrieve and modify data. • Copy and paste an input field into a text editor, the input field changes to the format {name, value}. • Copy some of the text and paste it, the entire input field is pasted. • The input field cannot be partially selected with other text.

Feature	PowerBuilder Classic behavior	PowerBuilder .NET behavior
RichText DataWindow final character	The final character is treated as an empty line.	The final character is not treated as an empty line.
RichText DataWindow font properties	When the user double-clicks the toolbar, the Font dialog appears and the user can set the font properties, including color.	The Font dialog is not available. Use the toolbar to set all font properties, including color.
RichText DataWindow input fields	• The color of the input field does not change when the content is selected. • Right-clicking on a field does not select it.	• The color of the input field changes to gray when the content is selected. • Right-clicking on a field selects it.
RichText DataWindow, RichTextCurrentStyleChanged event	• The event triggers at the same time whether the user selects left to right or right to left. • Setting the font in the toolbar does not trigger the event.	• The event triggers at different times if the user selects left to right or right to left. If the user selects it quickly, the event may not trigger at all. • Setting the font in the toolbar triggers the event.
RichText DataWindow, SelectTextAll function	The final paragraph marker character is not at the end of the TX control.	The final paragraph marker is appended automatically when the MS RichTextBox loads content.
Selected text color	When the user selects text, the highlight color is dark and the text turns white.	When the user selects text, the highlight color is light and the text does not change color.

Presentation Styles for DataWindow Objects

The presentation style you select for a DataWindow object determines the format PowerBuilder uses to display the DataWindow object in the Design view. You can use the format as it appears or modify it to meet your needs.

When you create a DataWindow object, you can choose from a variety of presentation styles.

Table 16. DataWindow presentation styles

Use this DataWindow wizard	To create a new DataWindow object
Composite	That includes other DataWindow objects (can also be created as a control)
Crosstab	With summary data in a spreadsheet-like grid
Freeform	With the data columns going down the page and labels next to each column
Graph	With data displayed in a graph
Grid	With data in row and column format with grid lines separating rows and columns
Group	With data in rows that are divided into groups
Label	That presents data as labels
N-Up	With two or more rows of data next to each other
RichText	That combines input fields that repesent database columns with formatted text
Tabular	With data columns going across the page and headers above each column
TreeView	With data grouped in rows in a TreeView that displays data hierarchically in a way that allows you to expand and collapse it

Selecting a SQL Data Source

When creating a DataWindow object using SQL, you can choose one of four SQL data source types: Quick Select, SQL Select, Query Object, or paste SQL from the clipboard.

See the *PowerBuilder Users Guide* for information about the data source types.

1. Create a new DataWindow object.
2. Enter the DataWindow name and click **Next.**
3. Select **SQL** and click **Next.**
4. Select the SQL data source type to use:
   - Quick Select SQL
   - SQL Select
   - Select Query Object
   - Paste SQL from Clipboard

You can also enter SQL directly into the SQL Statement area.

5. Follow the instructions for the type you selected.
   When you finish, you return to the DataWindow wizard.
6. You can continue with the remaining steps of the wizard or click **Finish** to proceed to the DataWindow painter.

Using SQL Select

When you specify data for a DataWindow object, SQL Select provides more options for specifying complex SQL statements when you use SQL Select rather than Quick Select.

When you choose SQL Select, you go to the SQL Select painter, where you can paint a SELECT statement that includes:

- More than one table
- Selection criteria (**WHERE** clause)
- Sorting criteria (**ORDER BY** clause)
- Grouping criteria (**GROUP BY** and **HAVING** clauses)
- Computed columns
- One or more arguments to be supplied at runtime

Note: In the SQL Select painter, you can save the current SELECT statement as a query by selecting **File > Save As** from the menu bar. This lets you easily use this data specification again in other DataWindows. See *Queries in PowerBuilder .NET* on page 168 and the *PowerBuilder Users Guide*.

Defining the Data Using SQL Select

Use the SQL Select painter to define a SQL statement to retrieve data for the DataWindow object.

1. In the New DataWindow wizard, click **SQL Select** in the Choose Data Source page and click **Next**.
   The Select Tables dialog box opens.
2. Select the tables and views to use in the DataWindow object.
   See *Selecting Tables and Views Using SQL Select* on page 166.
3. Select the columns to be retrieved from the database.
   See *Selecting Columns Using SQL Select* on page 167.
4. Join the tables if you have selected more than one.
   See *Joining tables* in the *PowerBuilder Users Guide*.
5. Select retrieval arguments if appropriate.

See *Using retrieval arguments* in the *PowerBuilder Users Guide*.

6. Limit the retrieved rows with **where**, **order by**, **group by**, and **having** criteria, if appropriate.

   See *Specifying selection, sorting, and grouping criteria* in the *PowerBuilder Users Guide*.

7. To eliminate duplicate rows, select **Design > Distinct**.
   The **distinct** keyword is added to the select statement.
8. Click **OK**.
   You return to the wizard to complete the definition of the DataWindow object.

Selecting Tables and Views Using SQL Select

Select the tables and views to include in a query. The tables and views that are available depend on the DBMS.

After you have chosen SQL Select as the data source in the New DataWindow wizard, the Select Tables dialog box appears in front of the Table Layout view of the SQL Select painter. For some DBMSs (SQL Anywhere®, for example) you see all tables and views, whether or not you have authorization. If you select a table or view you are not authorized to access, the DBMS issues a message.

For ODBC databases, the tables and views you see depend on the driver for the data source.

To select the tables and views, do one of the following:

Option	Description
Click the name of each table or view to open	Each table you select is highlighted. (To unselect a table, click it again.) Click **Open** to close the Select Tables dialog box.
Double-click each table or view to open	Each object opens immediately behind the Select Tables dialog box. Click **Cancel** to close the Select Tables dialog box.

Representations of the selected tables and views display. You can move or size each table to fit the space as needed.

Table Layout View in SQL Select

The Table Layout View in the SQL Select dialog shows representations of any tables and views you select.

By default, other tabbed views appear below the Table Layout view. Use these tabs to specify the select statement in more detail.

Specifying What Appears in Table Layout

In the Table Layout view, you can hide or display comments, datatypes, and labels (label information comes from the extended attribute system tables).

1. Right-click on any unused area of the Table Layout view and select **Show**.
   A cascading menu appears.
2. Select or clear Datatypes, Labels, or Comments as needed.

Selecting Columns Using SQL Select

In the Table Layout view, click each column to include from the table representations.

The columns you select are highlighted and placed in the Selection List at the top of the SQL Select painter.

1. In the Table Layout view, click a column once to select it.

Option	Description
Select all columns from a table	Right-click on the table name and select **Select All**.
Reorder the selected columns	Click and drag a column in the Selection List. Release the mouse button when the column is in the proper position in the list.

2. To see a preview of the query results, click **Execute** in the Painter bar.
   A table displaying the query results appears in the Results tab.
3. Click **OK**.

Including Computed Columns Using SQL Select

Computed columns that you define in the SQL Select painter are added to the SQL statement and used by the DBMS to retrieve the data. The expressions you define here follow your DBMS's rules.

Note: You can also define computed fields, which are created and processed dynamically by PowerBuilder after the data has been retrieved from the DBMS. There are advantages to doing this. For example, work is offloaded from the database server, and the computed fields update dynamically as data changes in the DataWindow object. (If you have many rows, however, dynamic updating may result in slower performance.)

To include computed columns:

1. Click the Compute tab (or select **View > Compute** if the Compute view does not currently appear).

Each row in the Compute view is a place for entering an expression that defines a computed column.

2. Enter one of the following:

Option	Description
An expression for the computed column	For example: `salary/12`
A function supported by your DBMS	For example (SQL Anywhere function): `substr("employee"."emp_fname",1,2)`

3. Right-click any row in the Compute view to display the context menu, which has options that allow you to select and paste the following into the expression:
   - Names of columns in the tables used in the DataWindow or pipeline
   - Any retrieval arguments you have specified
   - Functions supported by the DBMS (The functions are provided by your DBMS. They are not PowerBuilder functions. You are now defining a SELECT statement to be sent to your DBMS for processing.)
4. Press the **Tab** key to move to the next row to define another computed column, or click another tab to make additional specifications.

   PowerBuilder adds the computed columns to the list of selected columns.

Queries in PowerBuilder .NET

When you choose a query as the data source, you select a predefined SQL SELECT statement (a query) as specifying the data for your DataWindow object.

Queries save time, because you specify all the data requirements just once. For example, you can specify the columns, which rows to retrieve, and the sorting order in a query. Whenever you want to create a DataWindow object using that data, simply specify the query as the data source.

In PowerBuilder .NET version 12.5:

- If you import a target with existing queries, you can use those queries.
- You can create new queries by saving from the SQL Select dialog. See *Using SQL Select* on page 165.

You can edit existing queries using the text editor.

Previewing the Query

Preview a query to make sure it is retrieving the correct rows and columns.

1. Select **Design > Execute**.
   PowerBuilder retrieves the rows that satisfy the currently defined query in the Results view.
2. In the results view, manipulate the retrieved data as you do in the Database painter.

   You can sort and filter the data, but you cannot insert or delete rows or apply changes to the database. For more about manipulating data, see *Managing the Database* in the *PowerBuilder Users Guide*.

Saving the Query

Save a query at any time you are working on it.

1. Select **File > Save As Query**.

   PowerBuilder displays the Save Query dialog box.
2. In the Save Query dialog box, enter a name for the query.

   The query name can be any valid PowerBuilder identifier up to 255 characters. Use a unique name for each query. A common convention is to use a two-part name: a standard prefix that identifies the object as a query (such as **q_**) and a unique suffix. For example, you might name a query that displays employee data **q_emp_data**. For information about PowerBuilder identifiers, see the *PowerScript Reference*.
3. (Optional) Enter comments to describe the query.

   These comments appear in the Library painter. Sybase recommends that you use comments to remind yourself and others of the purpose of the query.
4. Specify the library in which to save the query, and click **OK**.

Modifying the Query

Use the text editor to modify existing queries. Before doing so, be familiar with PowerBuilder SELECT syntax.

1. Select **File > Open**.
2. Select the Queries object type and then the query you want to modify. Click **OK**.
3. In the text editor, modify the query.

About Composite Controls

A composite control is a container for Child DataWindows.

When you use the wizard to create a DataWindow in the composite style, you can create either a control or a DataWindow object.

- The Composite DataWindow object is a container for nested reports.

- The Composite control is a container for Child DataWindows.

Creating a Composite Control

You can create a composite as a control or DataWindow object.

1. Select **File > New** from the menu, or click the **New** button in the PowerBar.
2. Select **Composite** and click **Next**.
3. Enter a unique DataWindow name, select the library in which to save the new composite, then click **Next**.
4. Choose the DataWindows to include in the composite.
5. Specify the type of composite to create:
   - Select the Create as composite control check box to create a composite control, which is a container for child DataWindows.
   - Unselect the option to create a composite DataWindow object, which is a container for nested reports.
6. (Optional) Click **Next** to review the characteristics.
7. Click **Finish**.

Next

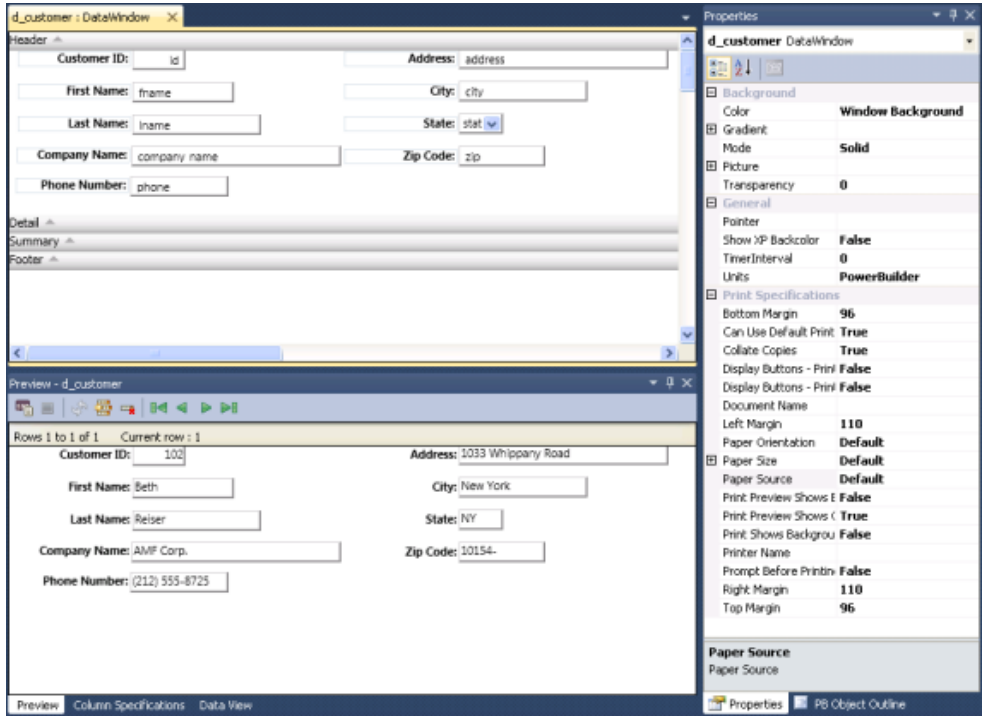
Use the Child DataWindow control to add the composite control to a DataWindow object.

DataWindow Object Enhancements

Use the DataWindow painter to enhance DataWindow objects, making them easier to use and interpret data.

DataWindow Painter

The DataWindow painter provides views related to the DataWindow object you are working on.



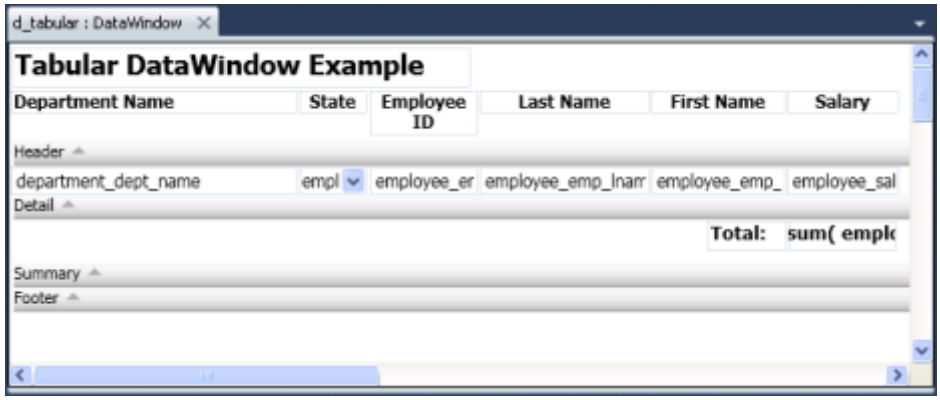
DataWindow Painter Design View

For most presentation styles, the DataWindow painter Design view is divided into areas called bands. Each band corresponds to a section of the DataWindow object.

DataWindow objects with these presentation styles are divided into four bands: header, detail, summary, and footer. Each band is identified by a bar containing the name of the band above the bar.

These bands can contain any information you want, including text, drawing controls, graphs, and computed fields containing aggregate totals.

Figure 1: Design view for a tabular DataWindow object



Band	Includes
Header	Information at the top of every screen, such as the name of the report or current date
Detail	Data from the database or other data source
Summary	Summary information that follows all data, such as totals and counts
Footer	Information that appears at the bottom of every page or screen, such as page number and page count

Header Band in DataWindows

The header band contains heading information that appears at the top of every screen or page.

The presentation style determines the contents of the header band:

- If the presentation style is Tabular, Grid, or N-Up, the headings defined for the columns in the Database painter are in the header band and the columns are on a single line across the detail band
- If the presentation style is Freeform, the header band is empty and labels are in the detail band next to each column

You can specify additional heading information (such as a date) in the header band and you can include pictures, graphic controls, and color to enhance the appearance of the band.

Note: To include the current date in the header, place a computed field that uses the Today DataWindow expression function in the header band.

Detail Band in DataWindows

The detail band shows retrieved data. It is also where the user enters and updates data.

The number of rows of data that appear simultaneously in PowerBuilder at one time is determined by this expression:

(Height of the DataWindow object - Height of headers and footers) / Height of the detail band

The presentation style determines the contents of the detail band:

- If the presentation style is Tabular, Grid, N-Up, or Label, the detail band displays column names, representing the columns.
- If the presentation style is Freeform, the labels defined for the columns in the Database painter appear in the detail band, with boxes for the data to the right.

You can define the display and validation information for each column of the DataWindow object, and add other controls, such as text, pictures, graphs, and drawing controls.

How PowerBuilder names the columns in the Design view

If the DataWindow object uses only one table, the names of the columns in the Design view are the same as the names in the table.

If the DataWindow object uses more than one table, the names of the columns in the Design view are *tablename_columnname*. This prevents ambiguity, since different tables can have columns with the same name.

Summary and Footer Bands

Use the summary and footer bands of the DataWindow object the same way you use summary pages and page footers in a printed report.

- The contents of the summary band appear at the end, after all the detail rows; this band often summarizes information in the DataWindow object.
- The contents of the footer band appear at the bottom of each screen or page of the DataWindow object; this band often includes the page number and name of the report.

DataWindow Painter Toolbars

The DataWindow painter contains two customizable PainterBars.

The PainterBars include buttons for standard operations (such as Save, Print, and Undo on PainterBar1) and for common formatting operations (such as Currency, Percent, and Layout in PainterBar2).

Properties View in the DataWindow Painter

Each part of the DataWindow object (such as text, columns, computed fields, bands, graphs, even the DataWindow object itself) has a set of properties appropriate to the part. These properties are in the Properties view.

Use the Properties view to modify the parts of the DataWindow object.

For example, the Properties view for a column has categories of information that you access by selecting the appropriate category. To choose an edit style for the column, select the Edit Style category.

Using the Preview View of a DataWindow Object

Use the Preview view to see a DataWindow object as it will appear with data and test the processing that takes place in it.

Note: When Retrieve On Preview is enabled, the data appears in the Preview view when it is first opened. If the view is already open, however, you must click Retrieve to see the data.

1. To open the Preview view, select **View > DataWindow Painter Views > Preview**. (If the menu option is not available, make sure the DataWindow painter is active.)

   In the Preview view, the bars that indicate the bands do not appear. If you defined retrieval arguments, you are prompted to supply arguments.
2. Test your DataWindow object.
   For example, modify some data, update the database, re-retrieve rows, and so on.

Saving Data to an External File

While previewing a DataWindow, you can save the data retrieved in an external file.

When you save data to a file:

- Data and headers (if specified) are saved.
- Information in the footer or summary bands is not saved unless you are saving as a PDF file or as an XPS file.
- If you are using right-to-left formatting in the DataWindow, the formatting is preserved if you save it as an XPS file, but not as a PDF file.

Controls in DataWindow Objects

Enhance DataWindow objects by adding and editing controls such as columns, drawing objects, buttons, and computed fields.

Adding Controls to a DataWindow Object

Enhance a DataWindow object by adding controls.

Using the Toolbox, you can add controls such as columns, drawing objects, buttons, and computed fields. You can also change the layout of the DataWindow object by reorganizing, positioning, and rotating controls.

Adding Columns to a DataWindow Object

You can add columns that are included in the data source to a DataWindow object.

When you create a DataWindow object, each column in the data source is automatically placed in it. Typically, you would add a column to restore one that you had deleted from the DataWindow object, or to show the column more than once in the DataWindow object.

Note: To add a column from the DataWindow definition to a DataWindow, insert it from the Toolbox. You might see unexpected results if you copy a column from one DataWindow object to another if they both reference the same column but the column order is different. (Copying a column copies a reference to the column's ID in the DataWindow definition.)

Suppose **d_first** and **d_second** both have **first_name** and **last_name** columns, but **first_name** is in column 1 in **d_first** and column 2 in **d_second**. If you delete the **first_name** column in **d_second** and paste column 1 from **d_first** in its place, both columns in **d_second** display the **last_name** column in the Preview view, because both columns now have a column ID of 1.

1. In the Toolbox, click the **Column** control.
2. Click where you want to place the column.
   You see the Select Column dialog box, which lists all columns included in the data source of the DataWindow object.
3. Select the column.

Adding Text to a DataWindow Object

You can add text anywhere in a DataWindow object.

When PowerBuilder generates a basic DataWindow object from a presentation style and data source, it places columns and their headings in the DataWindow painter. You can add text anywhere you want to make the DataWindow object easier to understand.

1. In the Toolbox, click the **Text** control.
2. Click where you want the text.
   PowerBuilder places the text control in the Design view and displays the word `text`.
3. Type the text you want.
4. (Optional) Use the Properties view to change the font size, style, and alignment.

Adding Drawing Controls to a DataWindow Object

You can add drawing controls to a DataWindow object.

You can add these drawing controls to a DataWindow object to enhance its appearance:

- Rectangle
- RoundRectangle
- Line
- Oval

1. In the Toolbox, select the drawing control.
2. Click where you want the control to appear.
3. Resize or move the drawing control as needed.
4. Use the drawing control's Properties view to change its properties as needed.

For example, you might want to specify a fill color for a rectangle or thickness for a line.

Adding a GroupBox to a DataWindow Object

To visually enhance the layout of a DataWindow object you can add a GroupBox. A GroupBox is a static frame used to group and label a set of controls in a DataWindow object.

1. In the Toolbox, select **GroupBox**.
2. Click where you want the control to appear.
3. With the GroupBox selected, edit the Text property in the General category in the Properties view. This changes the text that appears in the frame.
4. Move and resize the GroupBox as appropriate.

Adding Pictures to a DataWindow Object

You can place pictures, such as your company logo, in a DataWindow object.

If you place a picture in the header, summary, or footer band of the DataWindow object, it appears wherever the band appears. If you place a picture in the detail band of the DataWindow object, it appears on each row.

Note: WMF and EMF image formats are not supported in PowerBuilder .NET.

1. In the Toolbox, select the **Bitmap** control.
2. In the Design view, click where you want the picture to appear.
   The Open dialog box opens.
3. Browse to the image file and select it. Click **Open**.

   The picture must be a bitmap (BMP), runlength-encoded (RLE), Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPG) file, or Portable Network Graphics (PNG) file.
4. You can use the mouse to change the size of the image in the DataWindow painter, or set its Height and Width properties in the Properties view.

Adding Computed Fields to a DataWindow Object

You can use computed fields in any band of the DataWindow object.

You can enter any valid DataWindow expression when defining a computed field. You can paste operators, columns, and DataWindow expression functions into the expression in the Modify Expression dialog box. Use the + operator to concatenate strings.

You can use any built-in or user-defined global function in an expression. You cannot use object-level functions.

Note: You are entering a DataWindow expression, not a SQL expression processed by the DBMS, so the expression must follow the rules for DataWindow expressions.

1. In the Toolbox, select the **Compute** control.

2. In the Design view, click where you want to place the computed field.

   For calculations based on column data that changes for each row, place the computed field in the detail band.

   The Modify Expression dialog box lists:
   - DataWindow expression functions you can use in the computed field
   - The columns in the DataWindow object
   - Operators and parentheses
3. Enter the expression that defines the computed field.
4. (Optional) Click Verify to test the expression.
5. Click OK.

To display	Enter	In this band
Current date at top of each page	`Today()`	Header
Current time at top of each page	`Now()`	Header
Current page at bottom of each page	`Page()`	Footer
Total page count at bottom of each page	`PageCount()`	Footer
Concatenation of `Fname` and `Lname` columns for each row	`Fname + " " + Lname`	Detail
Monthly salary if **Salary** column contains annual salary	`Salary / 12`	Detail
Four asterisks if the value of the **Salary** column is greater than $50,000	`IF(Salary > 50000, "****", "")`	Detail
Average salary of all retrieved rows	`Avg(Salary)`	Summary
Number of retrieved rows, assuming each row contains a value for **EmpID**	`Count(EmpID)`	Summary

You can refer to other rows in a computed field. This is particularly useful in N-Up DataWindow objects when you want to refer to another row in the detail band. Use this syntax:

```
ColumnName[x]
```

where *x* is an integer. 0 refers to the current row (or first row in the detail band), 1 refers to the next row, -1 refers to the previous row, and so on.

For complete information about the functions you can use in computed fields in the DataWindow painter, see the *DataWindow Reference*.

Computed Columns Versus Computed Fields

When you define computed columns in the SQL Select painter, the value is calculated by the DBMS when the data is retrieved. When you define computed fields in the DataWindow painter, the value is calculated after the data has been retrieved.

When you create a DataWindow object, you can define computed columns and computed fields as follows:

- In the SQL Select painter, define computed columns when you define the SELECT statement that will retrieve data into the DataWindow object.
- In the DataWindow painter, define computed fields after you define the SELECT statement (or other data source).

When you define the computed column in the SQL Select painter, the value is calculated by the DBMS when the data is retrieved. The computed column's value does not change until data has been updated and retrieved again.

When you define the computed field in the DataWindow painter, the value of the column is calculated in the DataWindow object after the data has been retrieved. The value changes dynamically as the DataWindow object changes.

Note: If you want your DBMS to do the calculations on the server and you do not need the computed values to be updated dynamically, define the computed column as part of the SELECT statement.

If you need computed values to change dynamically, define computed fields in the DataWindow painter Design view.

Consider a DataWindow object with four columns: **Part number**, **Quantity**, **Price**, and **Cost**. **Cost** is computed as **Quantity** * **Price**.

Part#	Quantity	Price	Cost
101	100	1.25	125.00

If **Cost** is defined as a computed column in the SQL Select painter, the SELECT statement is:

```
SELECT part.part_num,
part.part_qty
part.part_price
part.part_qty * part.part_price
FROM part;
```

If the user changes the price of a part in the DataWindow object in this scenario, the cost does not change in the DataWindow object until the database is updated and the data is retrieved again. The user sees the changed price but the unchanged, incorrect cost.

Part#	Quantity	Price	Cost
101	100	2.50	**125.00**

If **Cost** is defined as a computed field in the DataWindow object, this is the SELECT statement (no computed column):

```
SELECT part.part_num
part.part_qty
part.part_price
FROM part;
```

The computed field is defined in the DataWindow object as `Quantity * Price`.

In this scenario, if the user changes the price of a part in the DataWindow object, the cost changes immediately.

Part#	Quantity	Price	Cost
101	100	2.50	**250.00**

Insert a Computed Field for the Current Date or Page Number

PowerBuilder provides a quick way to create computed fields that summarize values in the detail band, display the current date, or show the current page number.

The functions for the current date and page number are available in the Modify Expression dialog box.

1. Click the Compute control in the Toolbox.
2. Select today() or page() from the function list in the Modify Expression dialog box.
3. Click in the DataWindow object where you want the field to appear.

If you want to display Page *n* of *n*, create the expression: `'Page ' + page() + ' of ' + pageCount()`

Adding Buttons to a DataWindow Object

Button controls make it easy to provide command button actions in a DataWindow object. No coding is required.

The use of Button controls in the DataWindow object, instead of CommandButton controls in a window, ensures that actions appropriate to the DataWindow object are included in the object itself.

The Button control is a command or picture button that you can place in a DataWindow object. When clicked at runtime, the button activates either a built-in or user-supplied action.

For example, you can place a button in a report and specify that clicking it opens the Filter dialog box, where users can specify a filter to be applied to the currently retrieved data.

Note: Do not use a message box in the Clicked event. If you call the MessageBox function in the Clicked event, the action assigned to the button is executed, but the ButtonClicking and ButtonClicked events are not executed.

1. In the Toolbox, click the **Button** control.
2. Click where you want the button to appear.

   You may find it useful to put a Delete button or an Insert button in the detail band. Clicking a Delete button in the detail band deletes the row next to the button. Clicking an Insert button in the detail band inserts a row following the current row.

   Note: Buttons in the detail band repeat for every row of data, which is not always desirable. Buttons in the detail band are not visible during retrieval, so a Cancel button in the detail band would be unavailable when needed.
3. In the General category of the Properties view, enter the text for the button.
4. Select the action you want to assign to the button.

Actions Assignable to Buttons in DataWindow Objects

You can assign predefined actions to a Button control, or you can define an event script.

The table shows the actions you can assign to a button in a DataWindow object. Each action is associated with a numeric value (the Action DataWindow object property) and a return code (the actionreturncode event argument).

Use this code in the ButtonClicked event to display the value returned by the action:

```
MessageBox("Action return code", actionreturncode)
```

Action	Effect	Value	Return
NoAction (default)	Allows the developer to program the ButtonClicked event with no intervening action occurring.	0	The return code from the user's coded event script.
RetrieveYield	Retrieves rows from the database. Before retrieval occurs, the option to yield turns on; this allows the Cancel action to take effect during a long retrieve.	1	Number of rows retrieved. -1 if retrieve fails.
Retrieve	Retrieves rows from the database. The option to yield is not automatically turned on.	2	Number of rows retrieved. -1 if retrieve fails.
Cancel	Cancels a retrieval that has been started with the option to yield.	3	0

Action	Effect	Value	Return
PageNext	Scrolls to the next page.	4	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
PagePrior	Scrolls to the prior page.	5	The row that appears at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
PageFirst	Scrolls to the first page.	6	1 if successful. -1 if an error occurs.
PageLast	Scrolls to the last page.	7	The row displayed at the top of the DataWindow control when the scrolling is complete or attempts to go past the first row. -1 if an error occurs.
Sort	Displays Sort dialog box and sorts as specified.	8	1 if successful. -1 if an error occurs.
Filter	Displays Filter dialog box and filters as specified.	9	Number of rows filtered. Number < 0 if an error occurs.
DeleteRow	If button is in detail band, deletes row associated with button; otherwise, deletes the current row	10	1 if successful. -1 if an error occurs
AppendRow	Inserts row at the end.	11	Row number of newly inserted row.
InsertRow	If button is in detail band, inserts row using row number associated with the button; otherwise, inserts row using the current row.	12	Row number of newly inserted row.

Action	Effect	Value	Return
Update	Saves changes to the database. If the update is successful, a **Commit** will be issued; if the update fails, a **Rollback** will be issued.	13	1 if successful. -1 if an error occurs
SaveRowAs	Displays Save As dialog box and saves rows in the format specified.	14	Number of rows filtered. Number < 0 if an error occurs.
Print	Prints one copy of the DataWindow object.	15	0
Preview	Toggles between preview and print preview	16	0
PreviewWithRulers	Toggles between rulers on and off.	17	0
QueryMode	Toggles between query mode on and off.	18	0
QuerySort	Allows user to specify sorting criteria (forces query mode on).	19	0
QueryClear	Removes the WHERE clause from a query (if one was defined).	20	0

Add Graphs to DataWindow Objects

PowerBuilder offers many types of graphs and lets you control how they look.

Graphs are one of the best ways to present information. For example, if your application reports on sales information over the course of a year, you can easily build a graph in a DataWindow object to display the information visually.

See *Graphs in PowerBuilder .NET*.

Adding InkPicture Controls to a DataWindow Object

Add InkPicture controls to a DataWindow object.

The InkPicture control is designed for use on Tablet PCs and provides the ability to capture ink input from users of Tablet PCs: signatures, drawings, and other annotations that do not need to be recognized as text.

1. In the Toolbox, click the **InkPicture** control.
2. Click where you want to place the InkPicture control.

3. In the Ink Picture Database Setup dialog box, specify a blob column to store the ink data and another to use as a background image.
4. Click **OK**.

Adding Third-Party and Custom Controls to DataWindow Objects

Add third-party and custom controls to DataWindow objects using the Custom Control.

1. In the Toolbox, in the DataWindow Controls group, select **Custom Control**.
2. Click the location in the DataWindow where you want to place the custom control. The Select Custom Control dialog box opens.
3. Either:
   - Click the ellipsis button next to the From File field and navigate to the XAML file that defines the custom control.
   - Click the ellipsis button next to the From Assembly field and select the control.

   The XAML definition populates the Preview/Edit Xaml field.
4. Click **OK**.

Adding a TableBlob to a DataWindow Object

Add Table Binary/Large Text Objects to DataWindow objects.

Blob data is a binary large-object such as a Microsoft Word document or an Excel spreadsheet.

1. In the Toolbox, select the **TableBlob** control.
2. Click where you want to place the TableBlob control.
3. In the Database Binary/Large Text Object dialog box, specify the blob you want to appear in the DataWindow object.
4. Click **OK**.

Adding Reports and Child DataWindows to a DataWindow Object

You can nest reports (nonupdatable DataWindow objects) or Child DataWindows (updatable DataWindow objects) in a DataWindow object.

Nesting reports works the same in PowerBuilder .NET as in PowerBuilder Classic, except that you select the **Report** control from the **Toolbox**. See the *PowerBuilder Users Guide*.

The Child DataWindow control is similar to the nested report, but you do not need to set the AutoSizeHeight property to true because the control supports the HScrollbar, VScrollbar, and LiveScroll properties. Also, if you insert it into a Composite presentation style DataWindow object, the Child DataWindow is updatable. See *About Child DataWindows*.

Tooltips in DataWindow Objects

Tooltips display text when a mouse pauses over a DataWindow column or control.

You can use tooltip text to explain the purpose of a column or control. Select the column or control for which you want to create a tooltip and use the Tooltips properties in the Properties view.

Specify:

- Text for the tooltip
- Title for the tooltip
- Color of the background and text
- Icon for the tooltip
- Delay before the tooltip appears and disappears
- Whether the tooltip appears as a rectangle or callout bubble

About Child DataWindows

A Child DataWindow nests a DataWindow object inside another DataWindow object.

You can place one or more Child DataWindow controls within a DataWindow object (except the RichText or Crosstab presentation styles). If you add a Child DataWindow to a Composite DataWindow object, it is editable.

Use the `GetChild` method in Composite style DataWindow objects to get the reference of a Child DataWindow control; see the *DataWindow Reference*.

The Child DataWindow also supports these properties; see the *DataWindow Reference*.

- AccessibleName
- Attributes
- Band
- Border
- DataObject
- FlowDirection
- Height
- HideSnaked
- HScrollBar
- LiveScroll
- Moveable
- Name
- Nest_Arguments
- Pointer

- Resizeable
- SlideLeft
- SlideUp
- Tag
- Type
- Visible
- VScrollBar
- Width
- X
- Y

Adding Child DataWindows to a DataWindow

The Child DataWindow is similar to a Nested Report, except that it does not force print preview mode. Also it is editable in Composite Style DataWindow objects.

Note: You cannot place Child DataWindow controls in RichText or Crosstab style DataWindow objects.

1. In the DataWindow painter, select the **Child DataWindow** control from the **Toolbox**.
2. Click where you want the control to appear.
3. Select the DataWindow object.
4. Resize or move the Child DataWindow control as needed.
5. Use the control's Properties view to changes its properties as needed.

Child DataWindow Object

Use syntax to define a Child DataWindow.

Syntax

PowerBuilder dot notation:

```
dataobject = dw_control.object.cdw_control.dataobject
```

Describe argument:

```
dw_control.Describe("cdw_control.dataobject")
```

Modify argument:

```
dw_control.Modify("cdw_control.dataobject = 'd_dwobject'")
```

Usage

In the painter: Select the control and set the values in the Properties view.

Graphs in PowerBuilder .NET

You can present information as a graph in a DataWindow object or window.

PowerBuilder .NET provides various graphs and allows you to customize them in many ways. Graphs must be placed in a DataWindow object, and the source of the data is the database.

Parts of a Graph

Learn about the parts of a graph.

Here is a column graph created in PowerBuilder that contains most major parts of a graph. It shows quarterly sales of three products: Stellar, Cosmic, and Galactic printers.

How data is represented

Graphs display data points. To define graphs, you need to know how the data is represented. PowerBuilder organizes data into three components.

Table 17. Components of a graph

Component	Description	Meaning
Series	Set of data points	Each set of related data points makes up one series. Each series in a graph is distinguished by color, pattern, or symbol.
Categories	Major divisions of the data	Series data is divided into categories, which are often non-numeric. Categories represent values of the independent variables.

Compo-nent	Description	Meaning
Values	Data point values	The values for the data points (dependent variables).

Organization of a graph

Table 18. Parts of a typical graph

Part	Represents
Title	An optional title, appearing at the top of the graph.
Value axis	The axis of the graph along which the values of the dependent variable(s) are plotted. In a column graph, for example, the Value axis corresponds to the y axis in an XY presentation. In other types of graphs, such as a bar graph, the Value axis may be along the x dimension.
Category axis	The axis along which are plotted the major divisions of the data, representing the independent variable(s). In column graphs, the Category axis corresponds to the x axis in an XY presentation. These form the major divisions of data in the graph.
Series	A set of data points. In bar and column charts, each series is represented by bars or columns of one color or pattern.
Series axis	The axis along which the series are plotted in three-dimensional (3-D) graphs.
Legend	An optional listing of the series. The preceding graph contains a legend that shows how each series is represented in the graph.

Types of Graphs in PowerBuilder .NET

PowerBuilder provides many types of graphs.

Choose the graph type on the Define Graph Style page in the DataWindow wizard or in the General category in the Properties view for the graph.

Area, Bar, Column, and Line Graphs

Area, bar, column, and line graphs are conceptually very similar. They differ only in how they physically represent the data values—whether they use areas, bars, columns, or lines to represent the values. All other properties are the same.

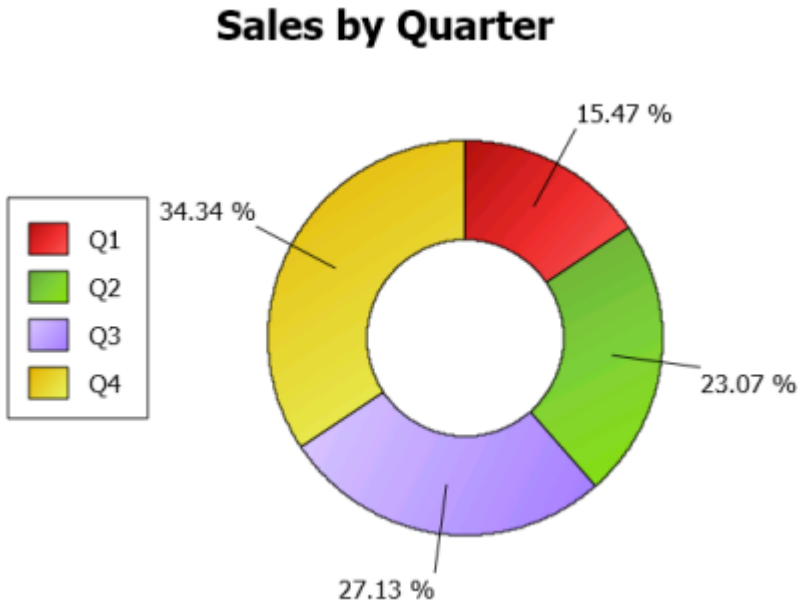
For more information on graphs, see the *PowerBuilder Users Guide*.

Pie and Donut Graphs

Pie and donut graphs typically show one series of data points with each data point shown as a percentage of a whole.

You can see the relative values in a series of data.

Figure 2: Donut graph



You can create pie and donut graphs that show more than one series; the series are shown in concentric circles. Multiseries pie and donut graphs can be useful in comparing series of data.

Scatter and Bubble Graphs

Scatter graphs use xy data points to show the relationship between two sets of numeric values.

You can only use numeric values in scatter graphs.

Scatter graphs do not use categories. Instead, numeric values are plotted along both axes, whereas other types of graphs have values along one axis and categories along the other axis.

Bubble graphs let you chart three data values in two dimensions, using one x data point and two y data points for each bubble.

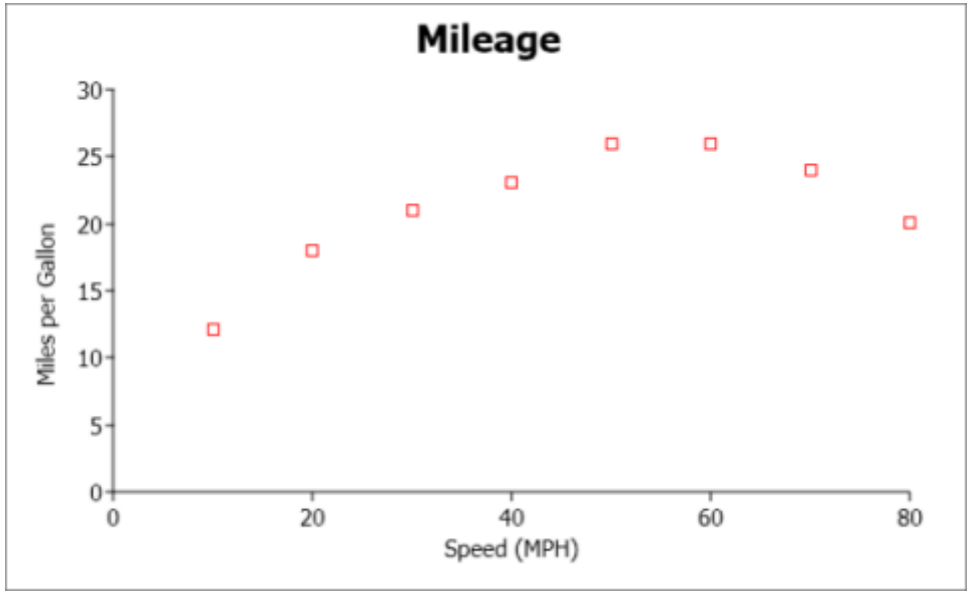
For example, the following data shows the effect of speed on the mileage of a sedan, and the number of minutes required to travel a mile at that speed:

Speed	Mileage	Minutes per mile
10	12	6.00
20	18	3.00
30	21	2.00
40	23	1.50
50	26	1.20

Speed	Mileage	Minutes per mile
60	26	1.00
70	24	0.86
80	20	0.75

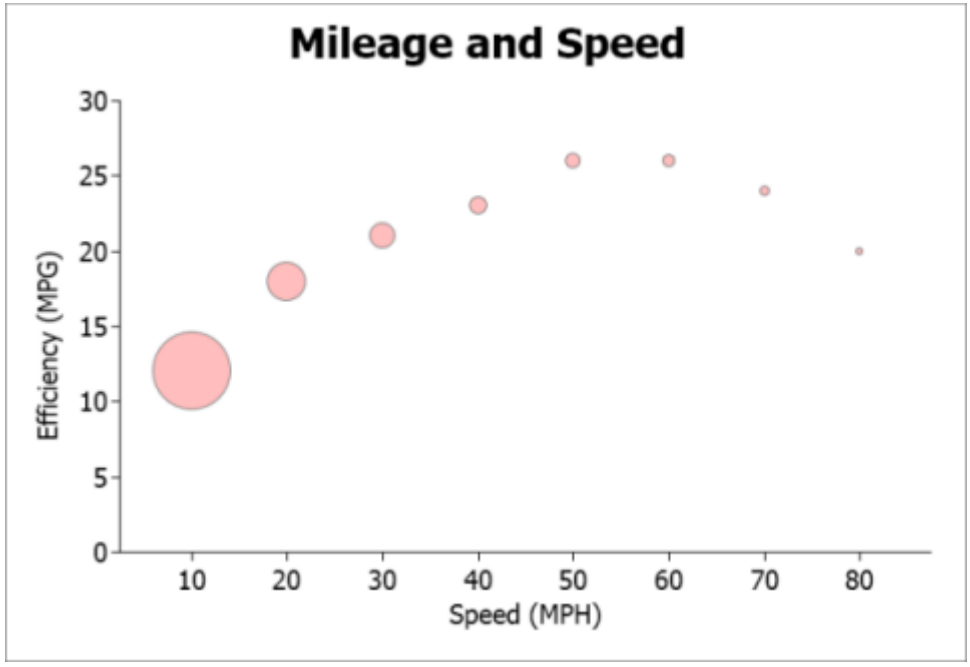
This scatter graph shows the data from the first two columns:

Figure 3: Scatter graph of the effect of speed on mileage:



This bubble graph shows the data from all three columns:

Figure 4: Bubble graph of the effect of speed on mileage and distance traveled:



SetBubbleSize Method

Use the **SetBubbleSize** function to define the size of the bubbles in a bubble graph.

Syntax

```
integer controlName.SetBubbleSize({string graphControl,} integer
seriesNumber, integer itemNumber, double size)
```

Argument	Description
controlName	The name of the graph in which to set the bubble size, or the name of the DataWindow control containing the graph.
graphControl (DataWindow control only)	(Optional) A string with a value that is the name of the graph in the DataWindow control.
seriesNumber	The number of the series in which to set the bubble size.
itemNumber	The number of the data point for which to set the bubble size.
size	(exp) A double that defines the size of the bubble. *Size* can be a DataWindow expression.

Return value

Returns 0 if successful, 1 for no action. If an error occurs, **SetBubbleSize** returns a negative integer. Values are:

- -1 – the row number provided is invalid.
- -2 – the column number provided is invalid.
- -3 – request requires currency but no current.
- -4 – datatype of request does not match column.
- -5 – argument passed is invalid.
- -6 – result is null.
- -7 – expression is bad.
- -8 – error occured during file I/O processing
- -9 – user cancelled action.
- -13 – the DataWindow is invalid.
- -14 – the graph object specified is invalid.
- -15 – code table index is invalid.
- -16 – invalid tree.
- -17 – runtime error.
- -18 – skip this request and execute the next one.

Usage

In bubble graphs, you can plot three data values on one x and two y axes. The size of the bubble is proportional to the value it represents. Use **SetBubbleSize** to define the size of the bubbles.

Example

This statement changes the size of the first bubble in the first series of graph **gr_1** in the DataWindow control **dw_1** to 25:

```
dw_1.SetBubbleSize("gr_1", 1, 1, 25)
```

BubbleSize Property

The size of a bubble in a bubble graph.

Syntax

PowerBuilder dot notation:

```
controlName.Object.graphname.BubbleSize
```

Argument	Description
controlName	The name of the graph in which to set the bubble size, or the name of the DataWindow control containing the graph.
graphName	A string value that is the name of the graph in the DataWindow control.

Argument	Description
size	A double that defines the size of the bubble.

Usage

In the painter, select the graph control and set the value in the Graph category in the Properties view.

Three-Dimensional Graphs

You can create three-dimensional (3-D) graphs of area, bar, column, line, pie, and donut graphs.

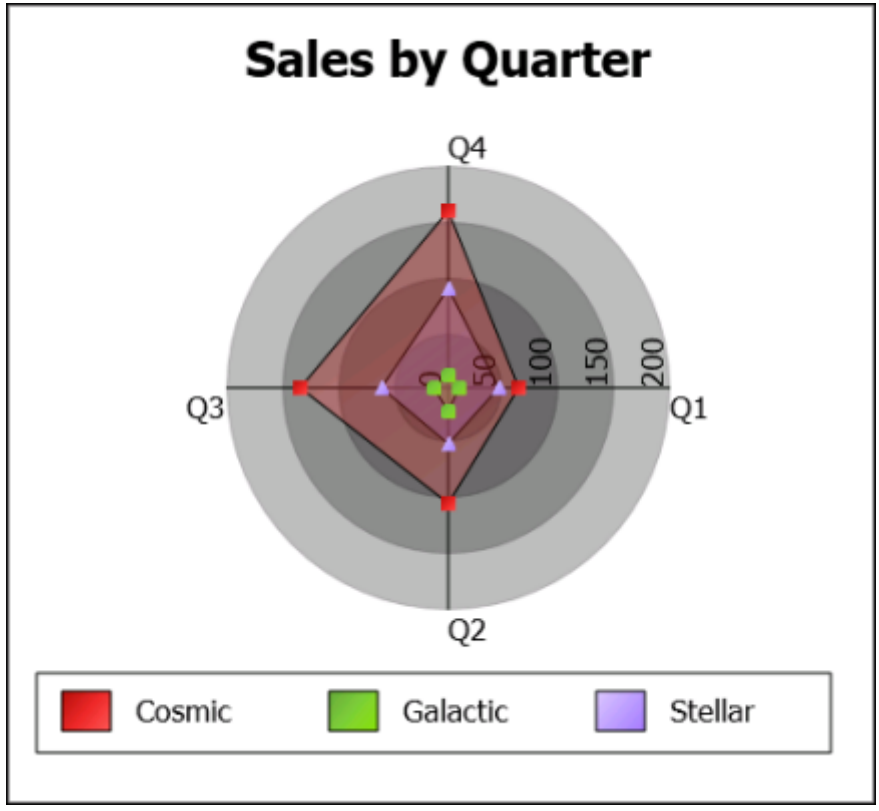
PowerBuilder .NET uses WPF features to render 3D graphs, color gradients, and graph object transparency, which makes Render3D an obsolete property. The Render3D property is supported in PowerBuilder .NET for backward compatibility, but it works only for Column3D and Bar3D graphs.

See the *PowerBuilder Users Guide*.

Radar Graphs

Radar graphs graphically display multivariate data (with three or more quantitative variables) in a two-dimensional chart represented on axes starting from the same point.

Radar graphs display data points on radii, displaying the data in a way that allows you to see patterns such as dominant variables, outliers, and other data patterns.



Stacked Graphs

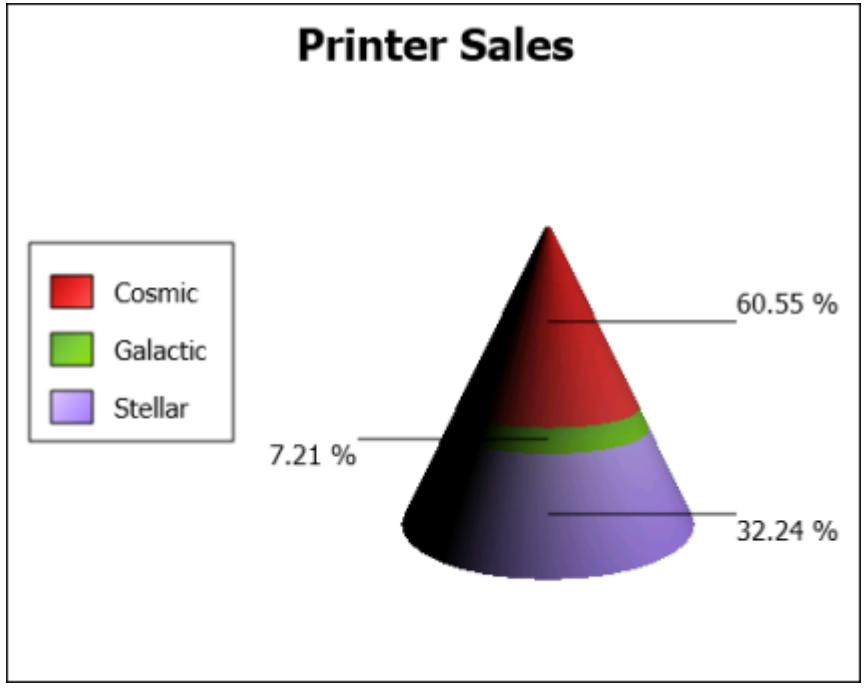
In bar and column graphs, you can stack the bars and columns.

In stacked graphs, each category is represented as one bar or column instead of as separate bars or columns for each series.

Cone Graphs

The cone graph style shows the percentage of every value in a series of data.

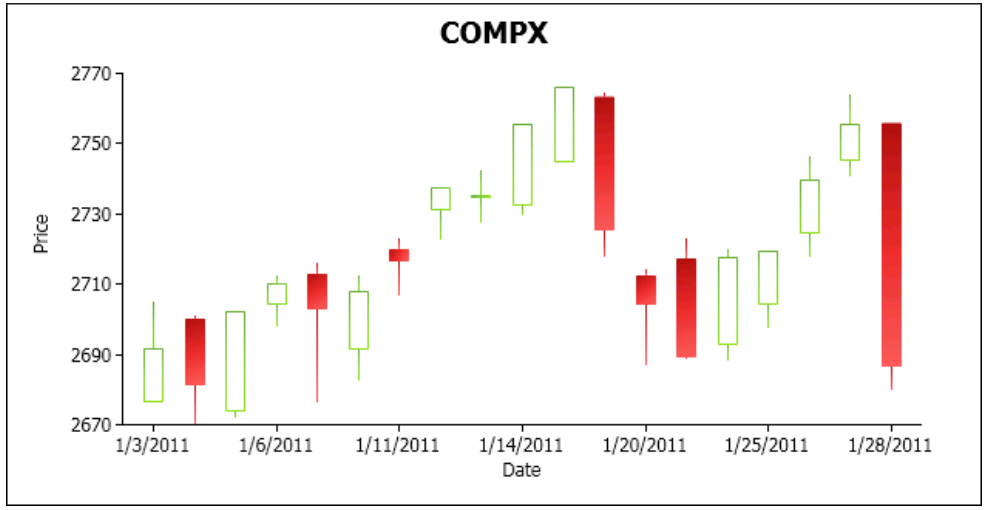
A cone graph represents one series of data. Different colors represent different data items in the series, and the heights represent the percentage it takes from the sum of the series.



Candlestick Graphs

Use the candlestick graph to show the prices of stocks, bonds, commodities, and so on.

- The end points of the line show the highest price and the lowest price at the date or time.
- The top and bottom of the rectangle show the opening price and closing price, which depends on whether the opening price is higher or lower than the closing price.
- An empty rectangle indicates that there was appreciation. A filled rectangle indicates depreciation.



Using the Palette

You can define palettes for any graph, which is useful for setting colors; for example, in some countries red has a negative connotation and is used for depreciation, but in other countries it has a positive connotation and is used for appreciation.

- The brush color at index 0 is for depreciation.
- The brush color at index 1 is for appreciation.

Integer for PowerBuilder .NET GraphType Property

The type of graph, such as donut, bubble, radar, and so on.

In addition to the graph types available in PowerBuilder Classic (see the *DataWindow Reference*), you can use integers to set the graph type value to one of the graphs only available in PowerBuilder .NET:

18 - Cylinder
19 - CylinderBar
20 - Bubble
21 - Radar
22 - Cone
23 - Donut
24 - Donut3D
25 - Candlestick

Graph Differences Between PowerBuilder Classic and PowerBuilder .NET

There are several differences between the graphs and Graph DataWindow objects in PowerBuilder .NET and PowerBuilder Classic.

Note: The graph control is no longer supported for Windows.

New graph styles and appearance

The appearance of existing graph styles has been improved in PowerBuilder .NET, and these styles are available.

- Bubble
- Cone
- Donut and 3-D donut
- Radar

You can customize the configuration of graph colors or the brushes arrangement. In 3-D graphs with axis frames, you can choose the brush for each frame. See *Axis Frames for Three-Dimensional Graphs* on page 198.

There is a new value for the Legend property. This value, Default, is the initial value for the graph's legend. If you change the value from this to another one, you cannot reset the legend to Default.

Tooltips

The Tooltip.*property* works a little differently in PowerBuilder .NET graphs.

Also, you can use tooltips to display the value of data points in graphs.

Unsupported properties

These properties are not supported for graphs in PowerBuilder .NET.

Property	Notes
axis.backedgepicture	You can change the appearance of pictures for the axis frames in some 3D graphs. See Picture.*property* in the *DataWindow Reference* for descriptions of all the properties available for pictures. When you set an axis to display a picture, you can use most of the properties included for pictures, except: clip.bottom, clip.left, clip.right, clip.top, mode, scale.x, and scale.y.
Render3D	Instead of using Render3D to make bar and column graphs three-dimensional, select the Solid Bar (Bar3DObj) or Solid Column (Col3DObj) graph. Render3D is maintained only for backward compatibility.

Palettes for Graphs

You can specify and configure the palette and brushes that will be used for the data or series style. The palette and brushes are defined in an XML file and then loaded using PowerScript.

LoadPalette Method

You can configure the color palette to use for the data or series style in a graph. Use the **LoadPalette** method to load the palette from an XML file.

Syntax

```
int LoadPalette(string graphcontrol, string xmlfilename)
```

Parameters

- **graphcontrol –** a string value that is the name of the graph in the DataWindow control.
- **xmlfilename –** a string value that is the name of the XML file containing the palette definition that you want to use in the graph.

Examples

- **–** To load the color palette for a graph, use:

```
dw_1.LoadPalette("gr_1", "palette.xml")
```

SavePalette Method

You can save a palette as an XML file to use later or in other graphs.

Syntax

```
int SavePalette(string graphcontrol, string xmlfilename)
```

Parameters

- **graphcontrol –** a string value that is the name of the graph in the DataWindow control.
- **xmlfilename –** a string that names the XML file you create.

Examples

- **–** To save a palette is saved as an XML file, use:

```
dw_1.LoadPalette("gr_1", "palette.xml")
```

SetDefaultBrushTable Method

Use the **SetDefaultBrushTable** method to change a single item of a palette.

Syntax

```
int SetDefaultBrushTable(string graphcontrol, int index, string
brushDefinitionFile)
```

Parameters

- **graphcontrol –** a string value that is the name of the graph in the DataWindow control.
- **index –** an integer 0–15 that defines the index of the palette.
- **brushDefinitionFile –** a string indicating the path name for the brush definition XAML file.

Examples

- **–** To set a brush in the palette, use:

```
SetDefaultBrushTable("gr_1", 2, "palette,xaml")
```

Axis Frames for Three-Dimensional Graphs

In PowerBuilder .NET, you can set the appearance of axis frames individually using solid colors, gradient colors, or images.

Properties

For each 3-D graph, there are three axis groups: Axis [Category], Axis [Series], and Axis [Values]. When the Shaded Back Edge property is set to True for an axis, you can set the color and gradient or picture for that axis.

Property for axis	Value
axis.backedgecolor	An integer specifying the color of the axis frame. Painter: Axis group, Backedge Color
axis.backedgepicture.file	A string indicating the path name for the picture file to be used for the axis frame. Supported formats are BMP, GIF, JPEG, RLE, and PNG. Painter: Axis group, Backedge Picture
axis.backedgepicture.transparency	An integer in the range 0 – 100, where 0 means that the background bitmap is opaque, and 100 that it is completely transparent. Painter: Axis group, Backedge Transparency

Property for axis	Value
axis.brushmode	An integer that defines the type of brush used for the axis frame. Values are: • 0 – Solid • 1 – HorizontalGradient • 2 – VerticalGradient • 3 – AngleGradient • 5 – RadialGradient • 6 – Picture Painter: Axis group, Backedge Mode
axis.gradient.angle	An integer indicating the angle in degrees (values are 0 – 360) used to offset the color and transparency gradient. This property is used only when the brushmode property takes a value of 3. Painter: Axis group, Backedge Gradient, Angle
axis.gradient.color	A long specifying the color (the red, green, and blue values) to be used as the axis frame's secondary color. The gradient defines transitions between the primary and secondary background colors. Painter: Axis group, Backedge Gradient, Color
axis.gradient.focus	An integer in the range 0 – 100, specifying the distance (as a percentage) from the center where the background color is at its maximum. (For example, if the radial gradient is used and the value is set to 0, the color is at the center of the background; if the value is 100, the color is at the edges of the background.) Painter: Axis group, Backedge Gradient, Focus
axis.gradient.repetition.count	An integer specifying the number of gradient transitions for background color and transparency. A value of 0 indicates 1 transition. A value of 3 indicates 4 transitions. This property is used only when the *axis*.gradient.repetition.mode property for the column or control takes the value of 0 (by count). Painter: Axis group, Backedge Gradient, RepetitionCount

Property for axis	Value
axis.gradient.repetition.length	A long specifying the number of gradient transitions. This property is used only when the gradient.repetition.mode property for the column or control takes the value of 1 (by length). The units for the length that you assign for gradient transitions are set by the DataWindow object's Units property. Painter: Axis group, Backedge Gradient, Repetition-Length
axis.gradient.repetition.mode	Specifies the mode for determining the number of gradient transitions for the axis frame's background color and transparency. Permitted values are: • ByRepetitionCount[0] – *axis*.gradient.repetition.count determines the number of gradient transitions. • ByLength[1] – *axis*.gradient.repetition.length determines the number of gradient transitions. Painter: Axis group, Backedge Gradient, Reptition Mode
axis.gradient.scale	An integer in the range 0 – 100 specifying the rate of transition to the gradient color (as a percentage). Painter: Axis group, Backedge Gradient, Scale
axis.gradient.spread	An integer in the range 0 – 100 indicating the contribution of the second color to the blend (as a percentage). Painter: Axis group, Backedge Gradient, Spread
axis.gradient.transparency	An integer in the range 0 – 100, where 0 means that the axis frame's secondary (gradient) background is opaque and 100 that it is transparent. The gradient defines transitions between the primary and secondary transparency settings. Painter: Axis group, Backedge Gradient, Transparency

Tooltip Functions for Graphs

You can create a tooltip to display the category, series, and value of a point in a graph. Use these methods to get the information for the tooltip: GetCurrentCategory, GetCurrentSeries, and GetCurrentValue.

GetCurrentCategory

Gets the category for a data point in a graph so that you can include in a tooltip.

Syntax

```
dwcontrol.modify("graphcontrol.tooltip.tip='~tGetCurrentCategory(graphcontrol)'")
```

Parameters

- **dwcontrol –** A reference to the DataWindow control containing the graph.
- **graphcontrol –** A string value that is the name of the graph in the DataWindow for which you want to get the category.

Examples

- **–** In this example, the tooltip for the graph is enabled and displays the name of the category.

```
dw_1.modify("gr_1.tooltip.tip='~tGetCurrentCategory(gr_1)'")
dw_1.Modify("gr_1.tooltip.enabled=1")
```

GetCurrentSeries

Gets the series for a data point in a graph so that you can include it in a tooltip.

Syntax

```
dwcontrol.modify("graphcontrol.tooltip.tip='~tGetCurrentSeries(graphcontrol)'")
```

Parameters

- **dwcontrol –** A reference to the DataWindow control containing the graph.
- **graphcontrol –** A string value that is the name of the graph in the DataWindow for which you want to get the series.

Examples

- **–** In this example, the tooltip for the graph is enabled and displays the name of the series.

```
dw_1.modify("gr_1.tooltip.tip='~tGetCurrentSeries(gr_1)'")
dw_1.Modify("gr_1.tooltip.enabled=1")
```

GetCurrentValue

Gets the value for a data point in a graph so that you can include it in a tooltip.

Syntax

```
dwcontrol.modify("graphcontrol.tooltip.tip='~tGetCurrentValue(graphcontrol)'")
```

Parameters

- **dwcontrol –** A reference to the DataWindow control containing the graph.
- **graphcontrol –** A string value that is the name of the graph in the DataWindow for which you want to get the category.

Examples

- – In this example, the tooltip for the graph is enabled and displays the value.

```
dw_1.modify("gr_1.tooltip.tip='~tGetCurrentValue(gr_1)'")
dw_1.Modify("gr_1.tooltip.enabled=1")
```

Database Management in PowerBuilder .NET

You can manage a database from within PowerBuilder.

For the most part, the Database painter works the same in PowerBuilder .NET as in PowerBuilder Classic.

PowerBuilder 12.5 provides enhanced support for connecting to any .NET provider through a managed ADO.NET interface. You can use the built-in ADO.NET drivers for Oracle, Adaptive Server Enterprise™, and SQL Server from PowerBuilder Classic or PowerBuilder .NET applications. These drivers enable you to access most of the current database features of those providers.

See *New Features in PowerBuilder 12* in the online help.

Defining Database Profiles

You can edit an existing database profile or create a new one.

Database profiles defined in either PowerBuilder Classic or PowerBuilder .NET are available to both.

Select **Tools > Database Profiles**, then click the **Edit** or **New** button.

See *Connecting to Your Database* in the *PowerBuilder Users Guide*.

The Database Painter in PowerBuilder .NET

Like the other PowerBuilder tool windows, the Database painter contains a menu bar, customizable PainterBars, and several views.

All database-related tasks that you can perform in PowerBuilder can also be performed in the Database painter. Because it is a tool window, you can dock it, auto-hide it, float it, or view it as a tabbed document.

To open views in the Database painter, click **View > Database Painter** .

Manipulating Data in the Database Painter

As you work on the database, you might want to look at existing data or create some data for testing purposes. You might also want to test display formats, validation rules, and edit styles on real data.

PowerBuilder provides data manipulation for such purposes. With data manipulation, you can:

- Retrieve and manipulate database information
- Save the contents of the database in a variety of formats (such as Excel, PDF, or XML)

These functions work the same as in PowerBuilder Classic; see the *PowerBuilder Users Guide*.

SQL Statements in the Database Painter

The Database painter's Interactive SQL view is a SQL editor in which you can enter and execute SQL statements. The view provides all editing capabilities needed for writing and modifying SQL statements.

You can cut, copy, and paste text; search for and replace text; and create SQL statements. You can also set editing properties to make reading your SQL files easier.

For the most part, the Database painter works the same in PowerBuilder .NET as in PowerBuilder Classic; see the *PowerBuilder Users Guide*.

DSI Database Trace Tool

The DSI Database Trace tool is a managed database interface that records the internal commands PowerBuilder executes while accessing a database.

Use the DSI Database Trace tool in PowerBuilder .NET WPF applications to trace DSI database connections at runtime. Unless you use the PBADO driver, the DSI data source interface calls the DBI database interface to connect to a database. This, in turn, activates the DBI Database Trace tool to trace the DBI database connections, although you can disable DBI tracing through a DSI parameter setting.

Enabling the DSI Database Trace tool

Enable DSI tracing in WPF applications by adding "tra" followed by a blank space before the assignment of the transaction object DBMS. For example:

```
SQLCA.DBMS = "TRA ODBC"
```

The DSI Database Trace tool writes the output of the DSI trace to a log file that you specify in a DSI parameter setting. When you initially enable DSI or DBI database tracing, PowerBuilder

creates the log file on your computer. Tracing continues until you disconnect from the database.

At design time, you can use only DBI tracing. See the *PowerBuilder Classic Users Guide*.

Parameters of the DSI database trace

Set the DSI Database Trace tool parameters in a DSITrace section of the configuration file for your application. The DSITrace section must also include a type parameter set to "Sybase.PowerBuilder.DataSource.DSITraceConfig, Sybase.PowerBuilder.DataSource,Version=12.0.0.0, Culture=neutral, PublicKeyToken=598c7456a83d557a".

Note: The DSITrace section with the required type parameter is automatically generated in the application configuration file at deployment time, but is commented out. You can uncomment this section when you set parameters for DSI tracing in the configuration file.

This table describes parameters you can set for DSI tracing in a DSITrace element after the configSections tag in the application configuration file.

Parameter	Description
fileName	Name and location of the trace log file. If you do not specify a full path name, the log file is saved in the current directory. If you do not use this parameter, the DSI trace file is not generated.
disableDBI-Trace (option-al)	Set or cancel DBI tracing when DSI tracing is enabled. This parameter is not valid for PBADO drivers. Values are: • 0 – (default) DBI tracing is enabled. • 1 – DBI tracing is disabled.
showParame-ters (optional)	Include or exclude SQL command parameters in the DSI trace log file. Values are: • 0 – (default) SQL command parameters are not included in the log file. • 1 – SQL command parameters are included in the log file.
showFetchData (optional)	Include or exclude data from fetch buffers in the DSI trace log file. Values are: • 0 – (default) Fetch data is not included in the log file. • 1 – Fetch data is included in the log file.

Configuration file example

The following settings in an application configuration file disable DBI tracing, cause the DSI trace to include SQL command parameters and fetch data in its log file, and save the log file to the root directory on the `c:` drive with the default `DSITRACE.LOG` file name:

```
<?xml version="1.0" encoding="utf-8" ?>
 <configuration>
```

```
           <configSections>
            <section name="DSITrace"
                  type="Sybase.PowerBuilder.
                           DataSource.DSITraceConfig, Sybase.
                          PowerBuilder.DataSource,Version=12.0.0.0,
                           Culture=neutral,
PublicKeyToken=598c7456a83d557a"  />
             </configSections>
            ...
          <DSITrace fileName="c:\dsitrace.log"  disableDBITrace="1"
                            showParameters="1"   showFetchData="1" /
>
             ...
              </configuration>
```

Trace file contents

The log file that you generate with the DSI Database Trace tool lists information from commands of three object types: DSConnection, DSCommand, and DSReader. The log file also lists SQL commands and data from fetch buffers if you enabled those options in the configuration file.

The format for the DSI command information is as follows, where *tID* is the thread ID, *objID* is a random 8-digit number for the DSObject, *DSObject* is DSConnection, DSCommand, or DSReader, and *commandIssued* is the name of the command. The starting time for the command is given to the second, and the duration of the command is given in milliseconds (MS):

```
   (tID.objID) DSObject.commandIssued() MM/DD/YYYY HR:MIN:SEC / ####
MS
```

Sharing ADO.NET Database Connections

PowerBuilder WPF applications can share database ADO.NET connections with third-party .NET assemblies.

Sharing database ADO.NET connections between an application and third-party .NET assemblies can minimize the number of connections to a database, or reduce the connection time.

About ADO.NET Database Connections

Database ADO.NET connections between WPF applications and third-party .NET assemblies are shared through a PowerBuilder connection proxy.

The connection proxy is defined in the `Sybase.PowerBuilder.DataSource.Sharing.dll` assembly, which implements the .NET Framework **System.Data.IDbConnection** interface. Both the

PowerBuilder application and the third-party assembly manage connections and transactions by referencing the proxy.

In the WPF application, the PowerBuilder Transaction object is the standard PowerBuilder nonvisual object used in database connections. To manage the shared connection, the Transaction object references the **AdoConnectionProxy** object using these methods:

- **bool SetAdoConnection (object connectionProxy) –** accepts an imported ADO.NET connection.
- **object GetAdoConnection() –** accepts an ADO.NET connection exported from the Transaction object.

Exporting an ADO.NET Database Connection

You can export an ADO.NET connection from a PowerBuilder WPF application, enabling the same database connection to be shared by an external .NET assembly.

Follow these guidelines to export a shared connection:

- In your PowerBuilder Transaction object, include a `DBMS` property that starts with `ADO`, and a `DBParm` property that specifies the driver's namespace.
- Start the connection before invoking GetAdoConnection.
- Use the **GetAdoConnection** method to export the connection proxy, using this syntax: object GetAdoConnection().

  The object is the **IAdoConnectionProxy** instance to be returned.

  The method returns `null` if it fails to make the connection.
- To use the shared connection, your third-party assembly must reference the exported connection proxy and manage the transaction. You can subscribe the **IAdoConnection.TransactionChanged** event to be notified when the active transaction is changed.
- Remember to close the connection.

The proxy's Connection property passes the shareable connection as a **System.Data.IDbConnection** object, which is available to the third-party .NET assembly at runtime. The active transaction object is assigned to the proxy's `IAdoConnectionProxy.Transaction` property.

- **Sample PowerScript code: –**

```
SQLCA.DBMS = "ADO.NET"
SQLCA.AutoCommit = false
SQLCA.DBParm = "Namespace='System.Data.Odbc', DataSource='SQL
Anywhere 11 Demo'"
Connect Using SQLCA;
emp.ConnectionProxy = SQLCA.GetAdoConnection()

// db operations
disconnect using SQLCA;
```

- **Sample C# assembly code –**

```
// Manage the transaction
public class Emp {
    ...

    IAdoConnectionProxy proxy;
    IDbTransaction trans;

    ...

    public object ConnectionProxy {
        get { return proxy; }
        set {
            proxy = value as IAdoConnectionProxy;
            ...
            proxy.TransactionChanged += new
EventHandler(proxy_TransactionChanged);
        }
    }

    void proxy_TransactionChanged(object sender, EventArgs e) {
        ...
        trans = sender as IDbTransaction;
        ...
    }
    ...
}
```

Importing an ADO.NET Database Connection

You can import an ADO.NET connection from an external .NET assembly into a PowerBuilder WPF application, enabling the application and the assembly to share the connection.

Follow these guidelines:

- In your PowerBuilder Transaction object, include a `DBMS` property that starts with `ADO`, and a `DBParm` property.
- Use the **SetAdoConnection** method to import the connection.

  bool SetAdoConnection(*ConnectionProxy*)

  ConnectionProxy is a proxy, of type **IAdoConnectionProxy**, that is instantiated by the third-party .NET assembly. The proxy's `Connection` property passes the shareable connection as a **System.Data.IDbConnection** object.

  The method returns `true` if the parameter is an available instance or null.

  The method returns `false` if it fails to make the connection or if the operation fails.
- Start the connection after invoking **SetAdoConnection**.

Note: If you start a transaction using C# code, update the transaction property.

- **Sample PowerScript code –**

```
SQLCA.DBMS = "ADO.NET"
SQLCA.AutoCommit = true
SQLCA.DBParm = "Namespace='System.Data.Odbc', DataSource='SQL
Anywhere 11 Demo'"
bool retVal = SQLCA.SetAdoConnection(emp.AdoConnectionProxy) //
emp is an instance of a type in the 3rd-party .NET assembly
if (retVal = true) then
    connect using SQLCA;
    // db operations
end if
```

- **Sample C# assembly code –**

```
public class Emp {

 private IDbConnection conn;
 private IDbTransaction trans;
 ...
  private IAdoConnectionProxy proxy;
  ...
  public object AdoConnectionProxy {
   get {

    //disposing/clean-up actions.

    if (null == proxy) {
     proxy = new AdoConnectionProxy();
    }
    proxy.Connection = conn;
    proxy.Transaction = trans;
    return proxy;
   }
   set {
    //disposing/clean-up actions.

    proxy = value as IAdoConnectionProxy;

    if (null != proxy) {
     if (conn != proxy.Connection as IDbConnection)
      this.Disconnect();

      conn = proxy.Connection as IDbConnection;
      trans = proxy.Transaction as IDbTransaction;
      proxy.TransactionChanged += new
EventHandler(proxy_TransactionChanged);
     } else {
       //disposing/clean-up actions.
     }
    }
   }
  ...
}
```

Debugging an Application

Use the PowerScript Debugger to identify and resolve logic and semantic errors that occur when you run your application.

Debugging is usually an iterative process: you run the debugger to evaluate your application's execution and diagnose problems in your code. After editing your code to fix the problems, run the debugger again to test your fixes. You might repeat this cycle many times.

This is a single iteration in a typical debugging cycle:

1. *Insert breakpoints* on page 213 where you suspect problems might occur.
2. *Run the application* on page 218 in debug mode, either stopping at breakpoints, or stepping through one line at a time.
3. Each time execution stops, use debugger functions to *examine the context of the application* on page 220, such as coding logic, the values of variables, or the call stack. You can add or modify breakpoints while your application runs, and test the effects of different variable values on execution.
4. *To fix problems that you discover during debugging,* on page 225 stop the debugger and edit your application in design mode.

PowerBuilder .NET Debugger Changes

The debugger is an implementation of the Visual Studio debugger. If you are familiar with either the PowerBuilder Classic or the Visual Studio debugger, you will notice both similarities and differences in the debugger.

This list summarizes important differences between the PowerBuilder .NET, PowerBuilder Classic, and Visual Studio debuggers:

PowerBuilder .NET debugger for WPF applications and .NET targets only – the PowerScript debugger for PowerBuilder .NET is used only to debug PowerScript code in WPF applications within the IDE. It is not used to debug other types of applications, such as WPF browser, Windows Forms, or WPF and .NET applications that you create outside of PowerBuilder .NET.

You cannot attach the debugger to the process of an application running outside of the IDE, or perform remote debugging.

Only debug version can be debugged – the release version cannot be debugged. If a debug version does not exist, the debugger fails. This is not related to the build type set in the Project painter.

IDE is read-only during debugging – in PowerBuilder .NET, all windows in the IDE become read-only when you start the debugger. To edit a script, stop the debugger to return to design mode.

Breakpoints enhanced – PowerBuilder .NET includes several breakpoint features from the Visual Studio debugger: in addition to breaking when a condition expression is true, you can break when it is false or when it changes. You can associate a custom action with a breakpoint using a Visual Studio tracepoint.

Edit Breakpoints dialog box replaced by Breakpoints window – the Breakpoints window is where you view and manage breakpoints.

Exceptions handled as in the Visual Studio debugger – the PowerBuilder .NET debugger implements exception handling from Visual Studio, which is different in some details from exception handling in PowerBuilder Classic.

Run to a specified function on the call stack – the application executes up to the first line of the specified function. This feature is implemented from the Visual Studio debugger.

PowerBuilder Classic debugger features not implemented – PowerBuilder Classic debugger features not impemented in PowerBuilder .NET include:

Just-in-time debugging
Objects in memory view
Source Browser and Source History view

Visual Studio debugger features not implemented – Visual Studio debugger features not implemented in PowerBuilder .NET include:

Watchpoints
Edit and Continue
Just-in-Time debugging
Debugging multithreaded applications

Variable views replaced by Visual Studio debugger windows – variable views in PowerBuilder Classic are replaced by these Visual Studio variable windows:

- **All Variables –** the All Variables window displays local variables in the procedure or function you are currently executing, as well as instance, global, and shared variables. The debugger automatically populates this window.
- **Autos –** the Autos window displays variables used in the current line of code and the preceding line of code. Like the All Variables window, the Autos window is populated automatically by the debugger.
- **Watch –** the Watch window is where you can add variables and expressions whose values you want to watch.
- **QuickWatch –** the QuickWatch dialog box is similar to the Watch window, but QuickWatch can display only one variable or expression at a time. QuickWatch can be

useful when you want to take a quick look at a variable or expression without using the Watch window.

TipWatch implemented as Visual Studio DataTip – in the PowerBuilder Classic Source view, you can place the cursor over a variable or simple expression to open a pop-up window that shows the current value of the variable or expression. In a PowerBuilder .NET Script window, the same action provides a tree display of the variable's elements.

Debugging and the Development Cycle

The PowerBuilder debugger lets you detect and diagnose problems while you run your application

After you build all or part of an application, you can run it directly in the PowerBuilder development environment. To simulate how your users will see and run the completed application, you can run it in regular mode.

However, an application often does not behave as you expect. Perhaps a variable is not being assigned the value you expect, or a script does not perform as desired. To detect problems as they occur during execution and to diagnose their causes, you can run your application in debug mode.

The PowerBuilder debugger lets you stop execution at places or times that you determine, such as a particular line of code or a function. The debugger also provides special tools for examining the application's state at each of those points.

Note: The PowerBuilder .NET debugger is based on the Visual Studio shell, so it is different from the PowerBuilder Classic debugger in important ways. See *PowerBuilder .NET Debugger Changes* on page 211 for a summary of these differences.

Setting a Breakpoint

Set breakpoints where you want to test or examine your application's state.

If you suspect a problem is occurring in a particular statement or function, insert a breakpoint at that point in the code, using any of several PowerBuilder windows. The breakpoint signals the debugger to interrupt program execution so you can examine it. In windows that display breakpoints, each one is marked in with a circular icon. The icon color indicates whether the breakpoint is currently enabled (red) or disabled (white).

The PowerScript debugger provides a range of ways to set and use breakpoints:

- Select a line of code in the Script Editor and add a breakpoint
- Specify the location of a breakpoint in a file
- Break only when a breakpoint is hit a specified number of times

- Break only for specified conditions
- Filter breakpoints for specified environments
- Define custom actions for breakpoints
- Set a breakpoint on a function
- Disable, enable, or delete a breakpoint

Setting a Breakpoint in a Script

You can set a breakpoint in a script at any time, while working in either debug or design mode.

1. Open the script in the Script Editor.
2. Place the cursor in the line of executable code where you want to interrupt execution.
3. Right-click the line and select **Insert Breakpoint**.
   PowerBuilder sets a breakpoint at the line and displays a red breakpoint icon in the margin. The red color indicates that the breakpoint is enabled. The breakpoint is also added to the Breakpoint window.

   If you select a line without executable code, PowerBuilder sets the breakpoint at the beginning of the next executable statement.

Changing a Breakpoint Location

Move a breakpoint to a different file or line.

1. Open the File Breakpoint dialog box:
   - In the Breakpoints window, right-click a breakpoint and choose **Location**.
   - In the Script window or Call Stack window, right-click a breakpoint and select **Breakpoint > Location**.
2. To set the new breakpoint location, specify the breakpoint file, line number, and character number within the line.
3. To be able to change a breakpoint to any file that is under version control, select **Allow the source code to be different from the original version**.

   By default, if you set a breakpoint to a file that is under source control, the breakpoint is hit only if the working copy of the file exactly matches the version that the executable was compiled from. The breakpoint is not hit if you have modified the working copy of the file.

   Selecting this option enables you to change breakpoints to files that are under version control, even if they have been modified.
4. Click **OK**.

Conditional Breakpoints and Hit Counts

You can use conditional breakpoints and hit counts to control when breakpoints are executed.

A condition limits when a breakpoint is executed. The condition can be any valid expression recognized by the debugger.

Each breakpoint also has a hit count, which specifies the number of times a breakpoint must be hit before it is executed. By default, the hit count is set to 0, meaning the breakpoint executes each time it is hit. You change how frequently a breakpoint is executed by specifying a positive hit count value.

Each time execution passes through the breakpoint, the debugger evaluates any condition expression. If the expression is true, the debugger increments a counter for the breakpoint, and compares its new value to the hit count setting:

- If the breakpoint counter is lower than the hit count, the breakpoint is not triggered.
- If the breakpoint counter is equal to or higher than the hit count, the debugger triggers the breakpoint and resets the breakpoint counter to 0.

If you specify a hit count of 3 and the condition `notisNull(val)`, PowerBuilder checks whether val is NULL each time the statement is reached. The breakpoint is triggered on the third occurrence of a non-NULL val, then again on the sixth occurrence, and so on.

Setting a Breakpoint Condition

Define a condition under which a breakpoint should be executed.

You can set both a conditional and a hit count on the same breakpoint.

1. Open the Breakpoint Condition dialog box:
   - In the Breakpoints window, right-click a breakpoint and select **Condition**.
   - In the Script window or Call Stack window, right-click a breakpoint and select **Breakpoint > Condition**.
2. Specify a valid boolean PowerScript expression.
3. Set the **Is true** option:
   - Select the option to break when the expression is true.
   - Unselect it to break when the expression is false.

   By default, this option is selected.

   If the expression is invalid, PowerBuilder ignores it, so the breakpoint is always triggered.
4. Set the **Has changed** option:
   - Select this option to break execution when the value of the expression changes.

Note: The debugger does not break on the first iteration of the breakpoint, because it does not yet detect a change in the condition; it detects changes only on subsequent iterations of the breakpoint.

- Unselect (default) to disregard changes in the value of the expression.

5. Click **OK**.

Setting a Breakpoint Hit Count

Control the number of times a breakpoint is hit before it stops execution, or track the hit count.

By default, execution breaks every time it hits a breakpoint. However, you might not always want to interrupt execution. For example, you might want to simply check the progress of a loop at a breakpoint. To change how frequently a particular breakpoint interrupts execution, you can change its hit count property.

1. Open the Hit Count dialog box for a breakpoint:
   - In the Breakpoints window, right-click a breakpoint and select **Hit Count**.
   - In a Script Editor or Call Stack window, right-click a breakpoint and select **Breakpoints > Hit Count** .
2. Specify when the breakpoint responds to the hit count:
   - **break always** (the default)
   - **break when the hit count is equal to**
   - **break when the hit count is a multiple of**
   - **break when the hit count is greater or equal to**
3. If you choose an action other than **break always**, specify the hit count value to trigger the break.

   The implicit value for **break always** is 0. For all other options, enter a number greater than 0.

   To keep track of the number of times a breakpoint is hit without ever breaking execution, set the hit count to a very high value.
4. Click **OK**.
   The hit count is retained only for the debugging session. When the debugging session ends, the hit count is reset to zero.

Setting a Breakpoint Filter

Set a breakpoint filter to restrict its use to specified machines, processes, and threads.

Breakpoint filters can be useful when you are debugging a parallel application that is spread across multiple processors.

1. Open the Breakpoint Filters dialog box:
   - In the Breakpoints window, right-click a breakpoint and select **Filter**.

- In the Script window or Call Stack window, right-click a breakpoint and select **Breakpoint > Filter**.

2. Specify machines, processes, or threads for which to enable the breakpoint. Specify machines by name and processes, and threads by either name or ID number.
3. Click **OK**.

Specifying a Tracepoint

Define a custom action for a breakpoint.

A tracepoint enables you to associate a custom action with a breakpoint, without having to modify your code. The breakpoint can either break or continue execution after the tracepoint action.

To specify a tracepoint action for a breakpoint:

1. Open the Breakpoint Condition dialog box:
   - In the Breakpoints window, right-click a breakpoint and select **When Hit**.
   - In the Script window or Call Stack window, right-click a breakpoint and select **Breakpoint > When Hit**.
2. Select **Print a message**. Accept the default message or create your own message.

   To include programmatic information in your tracepoint, use the syntax keywords described in the dialog box.

   ```
   In function '{$FUNCTION}', on thread '{$TID}'
   ```

   ```
   Used variable: {varName}, function name:
   {functionName($FUNCTION)}
   ```

   Run a macro is disabled in the Breakpoint Condition dialog box; PowerBuilder does not support running a Visual Studio automation model macro.
3. Specify whether to continue execution (default) or not.
4. Click **OK**.

Setting a Breakpoint on a Function

Set a breakpoint at a specific function.

1. Start the debugger and enter break mode, if you are not already in break mode.
2. Open the Call Stack window.
3. Right-click a function and select **Breakpoint > Insert Breakpoint**.
   PowerBuilder displays a breakpoint icon with the function name, and enables the breakpoint. It also inserts a breakpoint icon in the Script Editor (at the location in your code where the function is called) and the Breakpoints window.

Disabling, Enabling, or Deleting a Breakpoint

Control whether breakpoints are bypassed during the current debugging session.

To bypass a breakpoint, disable it. You can enable a disabled breakpoint at any time. If you no longer need a breakpoint, you can delete it.

- To disable a breakpoint:
  - In the Script Editor, right-click the breakpoint and select **Breakpoint > Disable Breakpoint**, or
  - In the Breakpoints window, unselect the breakpoint's check box.

  The breakpoint icon changes to white, in both the Breakpoints window and Script Editor.
- To disable all breakpoints, select **Breakpoint > Disable All Breakpoints**.
- To enable a disabled breakpoint, click its breakpoint icon.
  The white breakpoint icon changes to red.
- To delete a breakpoint:
  - In the Script Editor, click the breakpoint icon, or right-click the breakpoint and select **Breakpoint > Delete Breakpoint**, or
  - In the Breakpoints window, right-click a breakpoint and select **Delete**, or select one or more breakpoint lines (not their check boxes) and click **Delete**.

  PowerBuilder removes the breakpoint in the Script Editor and the Breakpoints window.
- To delete all breakpoints:
  - Select **Breakpoint > Delete All Breakpoints**, or
  - Click **Delete All** in the Breakpoints window toolbar.

  PowerBuilder removes all breakpoints in the Script Editor and the Breakpoints window.

Running in Debug Mode

Run your application in debug mode to test its execution.

There are many ways to control execution in the debugger; you can specify that it execute the application and stop either at each line (stepping), or at locations that you set (breakpoints). When execution halts temporarily at a line or breakpoint, use the debugger functions to examine the application at that point. When you are done examining the application context, tell the debugger to continue executing to the next line or breakpoint. You can execute through an entire application in this controlled way.

During your debugger session, your application is either running or it is in break mode. Most debugger functions are available only in break mode.

During debugging, you cannot edit your application: you can examine its status, add and remove breakpoints, and even modify variable values, but you cannot modify the code. You must end the debugger session and return to design mode in .

- To start the debugger, use one of these methods:
  - To step into your current application, select **Debug > Step Into**. The application runs to the first executable line of code.
  - If you have defined breakpoints and want to stop at the first breakpoint, select **Debug > Debug *project*** . If your solution has more than one project, click **Debug** in the toolbar to select a different project.

  The debugger breaks execution at the first executable line or breakpoint, depending on your starting method. The debugger and your application are now in break mode. The variables in the current scope are loaded in any open variable and watch windows, and most debugger functions are now available from the menus.
- To continue execution to the next breakpoint, click **Continue** or select **Debug > Continue**.
- To step through an application without entering functions, click **Step Over** or select **Debug > Step Over**.
  Each function is executed as a single statement.
- To enter functions as you step through the application, click **Step Into** or select **Debug > Step Into**.
  Execution stops at statements within functions.
- If you step into a function but do not want to step into its statements, click **Step Out** or select **Debug > Step Out**.
  The debugger continues execution until the function returns.
- To run to a function on the call stack:
  a) If the debugger is not in break mode, click **Break All** or select **Debug > Break All**.
  b) Open the Call Stack window if it is not open: select **Debug > Windows > Call Stack**.
  c) In the Call Stack window, right-click the function name and choose **Run To Cursor**.
  The application runs to the first executable line of the specified function, and opens the function in the Script Editor.
- To bypass a section of code or return to a statement:
  a) In the Script Editor, find the yellow arrow icon in the margin that marks the next statement to be executed.
  b) Click and drag the arrow icon to the new location where you want execution to resume.

  Setting the next statement can be useful for avoiding a section of code that contains a known bug.
- To stop debugging, click **Debug > Stop Debugging**.
  ends the debugging session and returns to design mode.

Examining an Application

Examine your application using debugger functions.

The following methods are useful for debugging your application at a step or breakpoint:

- Find a breakpoint in your source code: right-click a breakpoint in the Breakpoints or Call Stack window and choose **Go To Source Code**
- Examine a variable or expression
- Monitor a function call on the call stack

Examining a Variable or Expression

Examine how a variable or expression is resolved in the debugger.

Use one or more of these methods to examine variables and expressions. Unless otherwise noted, the methods are available only after you start the debugger and break execution:

- Place the edit cursor over a variable in a Script window to display its DataTip.

  The DataTip shows information about the variable in its current scope. If the underlying identifier is a data structure, the DataTip shows a tree that you can expand to view its members and the values of its properties. You can also see the values of variables, but you cannot change values in the DataTip. The DataTip closes when you remove the cursor.

  See the Visual Studio help for information about using DataTips.
- Use QuickWatch to see the current value of a single variable or expression. Select **Debug > QuickWatch**, or right-click in the Script window and select **QuickWatch**.

  See the Visual Studio help for information about evaluating expressions in QuickWatch.
- Use the All Variables window to see global, shared, and instance variables that are in scope. The window is usually open during debugging, but if necessary, you can manually open it by selecting **Debug > Windows > All Variables**.
- Use the Autos window to see variables used in the current line of code and the preceding line of code. This window is also usually open during debugging, but you can manuyally open it by selecting **Debug > Windows > Autos** .
- Use visualizers to examine the contents of data. Visualizers are represented in a Datatip or variable window by a magnifying glass icon. You can click the icon to see its DataWindow or ListView visualizer.
- Create as many as four Watch windows to store variables and expressions to evaluate during the debugging session. To open a Watch window, select **Debug > Windows > Watch > Watch*n*** .

  See the Visual Studio help for information about evaluating expressions in the Watch window.
- Change the value of a variable to see its effect on execution:

a) Right-click a variable in a DataTip, a variable window, or the Watch window, and select **Edit Value**.
b) Change the value.

The new value must conform to the type of the variable. If the variable is a string, do not enclose it in quotes. You cannot change the value of an enumerated variable.

You might want to change the value of a variable to examine different flows through the application, to simulate a condition that is difficult to reach in normal testing, or if you are skipping code that sets a variable's value.
The new value is used when you continue execution.

- Use the Immediate window at design time to evaluate variables or expressions during debugging, or to execute commands, functions, or subroutines at design time. To display the Immediate window, select **Debug > Windows > Immediate**.

  See the Visual Studio help for information about using the Immediate window.

Notes: Variables that are declared. or declared and initialized, but not used in a .NET application are discarded when the application is deployed to .NET. As a result, information about unused variables does not appear in the debugger.

Backslashes (\) in expressions appear as double blackslashes (\ \) in QuickWatch, DataTip, and Watch windows. The first backslash is an escape character. It indicates that the second backslash is text in the script rather than a control character.

Monitoring the Call Stack

View the function or procedure calls that are currently on the stack.

The Call Stack window displays the name of each function and the programming language it is written in. A yellow arrow identifies the stack frame where the execution pointer is currently located. By default, this is the frame where information appears in the Script Editor and in the variable (All Variables, Watch, and Autos) windows.

You can monitor items in the Call Stack window using any of these methods:

- To view the source code for a function that is currently on the stack, double-click the function in the Call Stack window, or right-click the function and select **Go To Source Code**.
  The Script Editor opens the function's script, and the variables windows show the variables and expressions in scope in that context.
- To switch to another stack frame:
  a) In the Call Stack window, right-click the frame whose code and data you want to view.
  b) Select **Switch To Frame**.

  A green arrow icon with a curly tail appears in the frame you select. The execution pointer remains in the original frame, which still has the yellow arrow icon. If you select **Debug >**

Step or **Debug > Continue**, execution continues in the original frame, not the selected frame.

- By default, the Call Stack window does not display calls to or from another thread. To display calls to or from another thread, right-click in the Call Stack window and select **Include Calls To/From Other Threads**.
- To load symbols for a module, right-click the frame that shows the module with the symbols you want to reload, and select **Load Symbols**.
- The Call Stack window context menu provides several other standard Visual Studio debugger functions. See the Visual Studio help for information about using those operations.

Debug Windows

The debugger uses several windows, each showing a different kind of information about the current state of your application or the debugging session

This table summarizes the contents and use of each window for debugging.

Table 19. Debugger Windows

Window	What it shows	What you can do
Breakpoints	A list of breakpoints with indicators showing whether the breakpoints are currently active or inactive.	Set, enable, disable, and clear breakpoints, set a condition for a breakpoint, and show source for a breakpoint in the Script Editor.
Call Stack	The sequence of function calls leading up to the function that was executing at the time of the breakpoint. Available only when the debugger is running.	Examine the context of the application at any line in the call stack.
Script Editor	The full text of a script. The Script Editor is updated when the context of your application changes to another script. If you have multiple Script Editors open, only the first one opened is updated.	Go to a specific line in a script, find a string, open another script, including ancestor and descendant scripts, manage breakpoints, and use DataTip and QuickWatch.

Window	What it shows	What you can do
Autos	A list of variables used in the current line of code and the preceding line of code, when the debugger is running. For native C++, the Autos window displays function return values as well.	Show and hide variables, change the value of a variable, and set a watch on a variable.
All Variables	An expandable list of all types of variables in scope when the debugger is running. Includes these types: • **Global Variables –** Values of all global variables defined for the application and properties of all objects (such as windows) that are open • **Shared Variables –** The shared variables associated with objects, such as applications, windows, and menus, that have been opened • **Instance Variables –** Properties of the current object instance (the object to which the current script belongs), and values of instance variables defined for the current object	Show and hide each variable type, change the value of a variable, and set a watch on a variable.
Watch	A list of variables you have selected to watch as the application runs in debug mode.	Change the value of a variable, set a watch on a variable, or add an arbitrary expression to the Watch view. You can maintain as many as four Watch windows.

Window	What it shows	What you can do
QuickWatch	Similar to the Watch window, but QuickWatch can display only one variable or expression at a time. Available only when the debugger is running.	Take a quick look at a variable or expression without bringing up the Watch window. Alternatively, use DataTips.
Immediate	A range of variables, expressions, or executable statements.	While in design mode or debug mode, specify variables or expressions that you want to evaluate during debugging. In design mode, immediately execute functions, subroutines, or Visual Studo commands.

Window	What it shows	What you can do
Visualizers	Display a data object in a manner that is appropriate to its data type. You can learn about Visualizers in the MSDN library for Visual Studio 2010. The PowerBuilder debugger provides two types of visualizers. Each displays the values of primary, deleted, and filtered data buffers in different ways: • **DataWindow visualizer –** displays the contents of the buffer as it might appear in an application that handles its data type, such as a bitmap or HTML document. **Note:** Does not display data for crosstab and composite style DataWindows, and does not support report control. • **ListView visualizer –** displays the contents of the buffer in a list format. **Note:** Displays blog data types as System.Byte[]. Displays only inner data for customer controls. Does not support composite DataWindows or report controls.	When you see the magnifying glass icon in a DataTip, in a debugger variables window, or in the QuickWatch dialog box, click the magnifying glass to see its contents in a visualizer:

Fixing Your Code

Return to design mode to fix errors that you find during debugging.

You cannot edit application source files in PowerBuilder during a debugging session. To fix errors that you find during during debugging, stop the debugger session, which returns

PowerBuilder .NET to design mode. After you have fixed the problems in your scripts or functions, you can reopen the debugger and run the application again in debug mode. The breakpoints and watch points set in your most recent session are still defined.

The DEBUG Preprocessor Directive

The DEBUG directive marks blocks of code that are to be processed during the first compilation pass.

DEBUG is one of the directives for conditional preprocessing in PowerBuilder. It is usually added to code only for debugging purposes. Most of the other directives are used only for .NET targets, but you can use the DEBUG directive in standard PowerBuilder targets as well.

The behavior of the DEBUG preprocessor depends on whether the Enable Debug Symbol option in the General page of the Project painter is selected when you test the application. When you run or debug the application in the development environment, the code is always parsed. When you run the executable file, the code is parsed only if the option is selected.

A DEBUG block in a script begins with the instruction, `#IF DEFINED DEBUG`, and ends with `#END IF`. These markers must each be on a single line.

ELSE clauses

You can add an ELSE clause to a DEBUG block. However, in that case the ELSE clause negates the DEBUG condition, so the preprocessor parses its contents differently than the main body of the IF clause. For example:

- The ELSE clause is parsed by the compiler when you build an executable file, even when the DEBUG directive is disabled.
- If you add a breakpoint in the ELSE clause, the debugger automatically switches the breakpoint to the last line of the clause defining the DEBUG condition.

  For example, consider this code:

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

  In this example, if you add a breakpoint in the ELSE clause, the breakpoint automatically switches to the `/*debugging code*/` line.

Limitations

Conditional compilation is not supported in DataWindow syntax, structure or menu objects. In source code, conditional compilation blocks that span function, event, or variable definition boundaries are invalid.

Using the DEBUG Preprocessor Directive

Use the DEBUG preprocessor directive to embed code in your application that is processed only by the debugger.

The DEBUG directive is frequently used during testing. You do not usually enable the DEBUG directive in a release build, but if a problem is reported in a production application, you can redeploy the release build with the DEBUG directive enabled to help determine the nature or location of the problem.

1. Add a DEBUG block around the code that you want to preprocess. For example, to paste a template into your code, right-click in the Script Editor and select **Paste Special > Preprocessor Statement > #If Defined DEBUG Then**.
2. Rebuild your application.
3. Determine how the DEBUG code is parsed at runtime:
   - If you want the DEBUG code block to always be parsed, whether you run the application in the development environment or from its executable file, select **Enable Debug Symbol** in the General page of the Project painter.
   - To parse the DEBUG code block only when you run the application in the development environment, unselect the DEBUG symbol.
4. Debug the application to examine the results.

The block of conditional code is automatically parsed by the PowerBuilder preprocessor before it is passed to the compiler.

You might add this DEBUG block of code to your application:

```
#if defined DEBUG then
   MessageBox("Debugging","Ctr value is " + string(i))
#end if
```

This code is always parsed and you always see the message box when you run or debug the application in the development environment. To see the message box when you run the executable file, enable the DEBUG symbol in the Project painter.

Breaking into the Debugger when an Exception is Thrown

Control how the debugger responds to exceptions.

When an application throws an exception during debugging, the debugger sees the exception before the program does. The debugger can allow the program to continue, or it can handle the exception itself. 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 how the debugger handles exceptions by setting options in the Exceptions dialog box, which contains expandable groups like **C++ Exceptions** and **Native Run-Time Checks**. These groups and most of the exceptions in them are provided by Visual Studio, but among them is a subgroup of exceptions. By default, all PowerBuilder exceptions are set to continue.

The Exceptions dialog box enables you to control breaking on thrown and user-unhandled exceptions. You can also add and remove exceptions. See the Visual Studio help to learn about these and other features of the Exceptions dialog box.

To control how the debugger handles exceptions, you can change settings in the Exceptions dialog box at any time (not just in debug mode):

1. Select **Debug > Exceptions**.
2. Expand the **Common Language Runtime Exceptions** category, then expand the PowerBuilder **Extensions** category to see its subgroups and exceptions.
3. Set the **Thrown** or **User-unhandled** property for any exception or category. Your settings are associated with the current workspace.

WCF Client Proxy Reference

The WCF Client Proxy project relies on a series of classes and enumerations to connect to a variety of services.

When you build a WCF Client Proxy project, the Project painter creates a WCFConnection object that it automatically instantiates with information obtained from an ASMX, WSDL, SVC, or XML file for the services you want to access.

The WCFConnection object has the WCFConnection class type, and invokes a number of other classes that can store information about binding types, transport and message security protocols, and other properties of the services you want to access for your client applications. The WCFConnection object allows you to create a service client without worrying about these connection details.

All classes listed in this reference are members of the PBWCF namespace in the Sybase.PowerBuilder.WCF.Runtime assembly. The classes required by your selections in the WCF Client Proxy wizard are automatically instantiated and initialized when you generate the corresponding proxy object. You can modify the initialized properties directly, or you can instantiate new instances of these classes with different property settings.

If you create new instances of these classes, you must use the PBWCF namespace or add it to a Using Namespace directive. For example, to change the proxy server settings for computers behind a firewall, you can create an instance of the WCFProxyServer class with code like this:

```
PBWCF.WCFProxyServer myProxySettings
myProxySettings= create PBWCF.WCFProxyServer
```

After you create the object instance, you can set its properties, and assign it to the ProxyServer property of the WCFConnection object instance that is assigned to the wcfConnectionObject property of the WCF client proxy object:

```
myProxyNVO.wcfConnectionObject.ProxyServer=myProxySettings
```

In the PowerBuilder .NET Script Editor, class and property names are case insensitive. You do not need to follow the capitalization visible in the Solution Explorer or in this reference.

Although you need not call WCF reference classes or select enumerated values for proxy server settings or service binding parameters, you can do so to override default values. You can also query class properties to view the parameters that the proxy object uses to connect to services. The reference topics that follow provide information about the WCFConnection object, and the classes and enumerations it uses to access these services.

WCFConnection Object

The WCFConnection object is an instance of the WCFConnection class, and includes all options for calling Web and WCF services.

The WCF Client Proxy project creates a WCFConnection object instance that is assigned, by default, to the wcfConnectionObject property of the proxy NVO generated when you select Generate Proxy from the project object's context menu or from the Design menu of the Project painter.

If necessary, you can replace the default WCFConnection object by creating a separate instance and assigning it to the wcfConnectionObject property of the generated proxy NVO, or you can modify individual properties of the default instance.

Properties

WCFConnection property	**Type**	**Description**
BindingType	WCFBindingType (enumeration)	Specifies the binding type and communciation format to use when accessing a service. The default value is generated from the service contract. If you delete this value, the connection is attempted using BasicHttpBinding as the default binding type. Values are: `BasicHttpBinding`, `wsHttpBinding`, `NetTcpBinding`, and `NetNamedPipeBinding`. WCF connection bindings are described on the MSDN Web site: *http://msdn.microsoft.com/en-us/library/ms731092.aspx*.
ClientCredential	WCFClientCredential (class)	Specifies the WCF credential to use for communication with the service.
EndpointAddress	WCFEndpointAddress (class)	Specifies a URL for the remote service, and tells the service engine where the service resides. If the endpoint is not explicitly set, the service engine uses the default endpoint embedded in the service contract file and in the generated service assembly.

WCFConnection property	Type	Description
ProxyServer	WCFProxyServer (class)	Allows you to specify credentials for a proxy server if the client machine is behind a firewall. If the client machine is directly connected to the Internet, you need not set this property.
Timeout	String	Specifies how long the WCF engine waits, while invoking the service, before timing out. The default value is 10 minutes ("00:10:00"). The format for the time is "hh:mm:ss". The service might have a different timeout value on the server side.
BasicHttpBinding	WCFBasicHttpBinding (class)	Defines binding settings for BasicHttpBinding. This property is used only when the BindingType property is BasicHttpBinding.
wsHttpBinding	WCFwsHttpBinding (class)	Defines binding settings for wsHttpBinding. This property is used only when the BindingType property is wsHttpBinding.
netTcpBinding	WCFNetTcpBinding (class)	Defines binding settings for NetTcpBinding. This property is used only when the BindingType property is NetTcpBinding.
netNamedPipeBinding	WCFNetNamedPipeBinding (class)	Defines binding settings for NetNamedPipeBinding. This property is used only when the BindingType property is NetNamedPipeBinding.
SoapMessageHeader	WCFSoapMessageHeader (class)	Defines SoapHeader items for the SOAP request message.

Events

WCFConnection event	Occurs
Constructor	Immediately before the Open event occurs in the window
Destructor	Immediately after the Close event occurs in the window

Classes Supporting WCF Client Connections

Several PowerScript system classes provide all the support required for WCF client connections from PowerBuilder .NET applications.

The objects created from these classes enable you to select and modify the binding types, transport modes, and message security settings for WCF or Web service connections.

BasicHttpMessageSecurity Class

The BasicHttpMessageSecurity class enables you to get and configure the message security settings of a BasicHttpBinding binding for a WCF client.

Properties

BasicHttpMessageSecurity property	**Type**	**Description**
SecurityAlgorithm	SecurityAlgorithmType (enumeration)	Specifies the algorithm suite to use for Http message security. Values are: `DEFAULT`, `BASIC128`, `BASIC128RSA15`, `BASIC128HA256`, `BASIC128SHA256RSA15`, `BASIC192`, `BASIC192RSA15`, `BASIC192HA256`, `BASIC192SHA256RSA15`, `BASIC256` (default), `BASIC256RSA15`, `BASIC256HA256`, `BASIC256SHA256RSA15`, `TRIPLEDES`, `TRIPLEDESRSA15`, `TRIPLEDESSHA256`, and `TRIPLEDESSHA256RSA15`.
ClientCredentialType	BasicHttpMessageCredentialType (enumeration)	Specifies the credential type the client uses for authentication. Values are: `UserName` (default) and `Certificate`.

BasicHttpSecurity Class

The BasicHttpSecurity class configures the security settings of a BasicHttpBinding binding for a WCF client.

Properties

BasicHttpSecurity property	Type	Description
SecurityMode	BasicHttpSecurityMode (enumeration)	Gets or sets the security mode for a BasicHttpBinding binding. Values are: `None` (default), `Transport`, `Message`, `TransportCredentialOnly`, and `TransportWithMessageCredential`.
Transport	HttpTransportSecurity (class)	Gets the transport-level security settings for a BasicHttpBinding binding.
Message	BasicHttpMessageSecurity (class)	Gets the message-level security settings for a BasicHttpBinding binding.

ClientCertificateCredential Class

The ClientCertificateCredential class enables you to select a client certificate for a secure connection to a WCF or Web service.

Properties

ClientCertificateCredential property	Type	Description
SubjectName	String	Specifies the full path file name or the distinguished subject name of the certificate to use. If you use a full path file name, the StoreLocation and StoreName properties are ignored.
StoreLocation	CertStoreLocation (enumeration)	Specifies the location of the X.509 certificate store where the certificate resides.
StoreName	CertStoreName (enumeration)	Specifies the name of the X.509 certificate store where the certificate resides.

Note: The distinguished subject name of a certificate that you can specify for the SubjectName property is a comma-separated textual representation of the certificate details in the format "E=*mailAddress*, CN=*commonName*, OU=*orgUnit*, O=*organization*, L=*locality*,

S=*stateOrProvince*,C=*countryCode*". Determine the distinguished subject name by double-clicking the certificate you want to use in the Certificate Manager (accessible from the Content tab of the Internet Explorer Internet Options dialog box), switching to the Details tab, and selecting Subject in the Field column of the list view on the Details tab.

HttpTransportSecurity Class

The HttpTransportSecurity class enables you to get and configure transport security settings for a WCF or Web service that uses BasicHttpBinding or wsHttpBinding bindings.

Properties

HttpTransportSecurity property	Type	Description
Realm	String	Gets or sets the authentication realm for digest or basic authentication. The authentication realm string must include the name of the host performing the authentication, and can also list the collection of users with access permissions.
ClientCredentialType	HttpClientCredentialType (enumeration)	Specifies the credential type for HTTP client authentication. Values are: `None` (default), `Basic`, `Digest`, `NTLM`, `Windows`, and `Certificate`.

HttpDigestCredential Class

The HttpDigestCredential class provides credential information that allows a WCF client to use a proxy server, or connect to a WCF or Web service that requires digest authentication.

Invoke objects of the HttpDigestCredential class from the HttpDigest property of WCFClientCredential class objects.

Properties

HttpDigestCredential property	Type	Description
UserName	UserNameCredential (class)	Specifies the user name, password, and domain of the client.

HttpDigestCredential property	Type	Description
AllowImpersonationLevel	ImpersonationLevel (enumeration)	Determines the impersonation level of the WCF client. Values are: `None` (default), `Anonymous`, `Identification`, `Impersonation`, and `Delegation`.

MessageSecurityOverTcp Class

The MessageSecurity class configures message-level security settings for a message sent using TCP transport.

Properties

MessageSecurityOverTcp property	Type	Description
AlgorithmSuite	SecurityAlgorithmType (enumeration)	Specifies the algorithm suite used for message-level security. Valid values are: `DEFAULT`, `BASIC128`, `BASIC128RSA15`, `BASIC128HA256`, `BASIC128SHA256RSA15`, `BASIC192`, `BASIC192RSA15`, `BASIC192HA256`, `BASIC192SHA256RSA15`, `BASIC256` (default), `BASIC256RSA15`, `BASIC256HA256`, `BASIC256SHA256RSA15`, `TRIPLEDES`, `TRIPLEDESRSA15`, `TRIPLEDESSHA256`, and `TRIPLEDESSHA256RSA15`.
ClientCredentialType	MessageCredentialType (enumeration)	Specifies the credential type used for client authentication. Values are: `NONE`, `WINDOWS` (default), `USERNAME`, `CERTIFICATE`, and `ISSUEDTOKEN`.

NamedPipeTransportSecurity Class

The NamedPipeTransportSecurity class provides a property to control the protection level for a named pipe.

Properties

NamedPipeTransportSecurity property	Type	Description
ProtectionLevel	ProtectionLevel (enumeration)	Gets or sets the protection level for a named pipe. Values are: `NONE`, `SIGN`, and `ENCRYPTANDSIGN` (default).

NetNamedPipeSecurity Class

The NetNamedPipeSecurity class configures the security settings for endpoints with the NetNamedPipeBinding binding.

Properties

NetNamedPipeSecurity property	Type	Description
SecurityMode	NetNamedPipeSecurityMode (enumeration)	Gets or sets the security mode for the binding. Values are: `None` and `Transport` (default).
Transport	NamedPipeTransportSecurity (class)	Provides properties to control authentication parameters and the protection level for named pipe transport.

NetTcpSecurity Class

The NetTcpSecurity class configures the security settings of a NetTcpBinding binding for a WCF client and WCF service.

Properties

NetTcpSecurity property	Type	Description
SecurityMode	wsSecurityMode (enumeration)	Gets or sets the security mode for the binding. Values are: `None`, `Transport` (default), `Message`, and `TransportWithMessageCredential`.
Transport	TcpTransportSecurity (class)	Provides properties to control authentication parameters and the protection level for TCP transport.
Message	MessageSecurityOverTcp (class)	Provides properties that control authentication parameters and the protection level for TCP messages.

NoDualHttpMessageSecurity Class

The NoDualHttpMessageSecurity class represents message-level security settings over an HTTP protocol where security is not set for two-way communication.

Properties

NoDualHttpMessageSecurity property	Type	Description
SecurityAlgorithm	SecurityAlgorithmType (enumeration)	Specifies the algorithm suite to use with NoDualHttpMessageSecurity. Values are: `DEFAULT`, `BASIC128`, `BASIC128RSA15`, `BASIC128HA256`, `BASIC128SHA256RSA15`, `BASIC192`, `BASIC192RSA15`, `BASIC192HA256`, `BASIC192SHA256RSA15`, `BASIC256` (default), `BASIC256RSA15`, `BASIC256HA256`, `BASIC256SHA256RSA15`, `TRIPLEDES`, `TRIPLEDESRSA15`, `TRIPLEDESSHA256`, and `TRIPLEDESSHA256RSA15`.
ClientCredentialType	MessageCredentialType (enumeration)	Specifies the credential type to use for client authentication. Values are: `NONE`, `WINDOWS` (default), `USERNAME`, `CERTIFICATE`, and `ISSUEDTOKEN`.

NoDualHttpMessageSecurity property	Type	Description
EstablishSecurityContext	Boolean	Controls whether a security context token is established through a WS-SecureConversation exchange between a client and service. Values are: • `true` (default) – requires the remote party to support the WS-SecureConversation protocol. • `false` – the WS-SecureConversation protocol is not used for the communication.
NegotiateServiceCredential	Boolean	Specifies whether the service credential is supplied by the service through a negotiated process. Values are: • `true` (default) – service credential is obtained through a negotiated process. • `false` – service credential is supplied by the client out of band.

ServiceCertificateCredential Class

The ServiceCertificateCredential class provides a client certificate for a secure connection to a service. Use this class instead of the ClientCredentialService class when message-level security is enabled for the service.

Properties

ServiceCertificateCredential property	Type	Description
SubjectName	String	Specifies the full path file name or the distinguished subject name of the certificate to use. If you use a full path file name, the StoreLocation and StoreName properties are ignored.
StoreLocation	CertStoreLocation (enumeration)	Specifies the location of the X.509 certificate store where the certificate resides.

ServiceCertificateCredential property	Type	Description
StoreName	CertStoreName (enumeration)	Specifies the name of the X.509 certificate store where the certificate resides.

Note: The distinguished subject name of a certificate that you can specify for the SubjectName property is a comma-separated textual representation of the certificate details in the format "E=*mailAddress*, CN=*commonName*, OU=*orgUnit*, O=*organization*, L=*locality*, S=*stateOrProvince*,C=*countryCode*". Determine the distinguished subject name by double-clicking the certificate you want to use in the Certificate Manager (accessible from the Content tab of the Internet Explorer Internet Options dialog box), switching to the Details tab, and selecting Subject in the Field column of the list view on the Details tab.

TcpTransportSecurity Class

The TcpTransportSecurity class provides properties to control authentication parameters and the protection level for TCP transport. Use it, instead of the HttpTransportSecurity class, to set the transport-level security for service bindings that use TCP for message delivery.

Properties

TcpTransportSecurity property	Type	Description
ClientCredentialType	TcpClientCredentialType (enumeration)	Specifies the credential type for TCP client authentication. Values are: `NONE`, `WINDOWS` (default), and `CERTIFICATE`.
ProtectionLevel	ProtectionLevel (enumeration)	Gets or sets the protection level for the TCP stream. Values are: `NONE`, `SIGN`, and `ENCRYPTANDSIGN` (default).

UserNameCredential Class

The UserNameCredential class provides a user name, password, and domain name that enable you to authenticate a client to a WCF or Web service, or to a proxy server.

Invoke objects of the UserNameCredential class from the UserName property of WCFClientCredential, HTTPDigestCredential, or WindowsCredential class objects.

Properties

UserNameCredential property	Type	Description
UserName	String	Specifies a user name
Password	String	Specifies a password
Domain	String	Specifies a domain that the user belongs to

WCFBasicHttpBinding Class

Use the WCFBasicHttpBinding class for communication between a WCF client and ASMX-based Web services or other services that conform to the WS-I Basic Profile 1.1.

Properties

WCFBasicHttpBinding property	Type	Description
TransferMode	WSTransferMode (enumeration)	Gets or sets a value to indicate whether messages are buffered or streamed. Values are: `BUFFERED` (default), `STREAMED`, `STREAMEDREQUEST`, and `STREAMEDRESPONSE`.
MessageEncoding	WSMessageEncoding (enumeration)	Specifies encoding for SOAP messages. Values are: `Text` (default), and `MTOM` (Message Transmission Optimization Mechanism).
TextEncoding	WSTextEncoding (enumeration)	Specifies character encoding for the SOAP message text. Values are: `ASCII`, `BIGENDIANUNICODE`, `UNICODE`, `UTF32`, `UTF7`, and `UTF8` (default).
Security	BasicHttpSecurity (class)	Gets or configures the security settings of a BasicHttpBinding connection.

WCFBasicHttpBinding property	Type	Description
ReaderQuotas	WCFReaderQuotas (class)	Gets or sets processing constraints based on the SOAP message complexity for endpoints with this binding type.
AllowCookies	Boolean	Indicates whether the client accepts cookies. Values are: • `true` – accepts cookies. • `false` (default) – does not accept cookies.
BypassProxyOnLocal	Boolean	Indicates whether to bypass the proxy server for local addresses. Values are: • `true` – bypasses the proxy server. • `false` (default) – does not bypass the proxy server.
HostNameComparisonMode	WCFHostNameComparisonMode (enum)	When matching the URI, indicates whether to use the host name to reach the service. Values are: `StrongWildCard` (default), `Exact`, and `WeakWildCard`.
MaxBufferPoolSize	Int64 (PowerScript longlong)	Specifies the maximum amount of memory allocated for the manager of the buffers required by the endpoints using this binding. The default value is 524288.
MaxBufferSize	Int32 (PowerScript long)	Specifies the maximum size for a buffer that receives messages from the channel. The default value is 65536.
MaxReceivedMessageSize	Int64 (PowerScript longlong)	Specifies the maximum size for a message that can be processed by the binding. The default value is 65536.

Methods

WCFBasicHttpBinding method	Description
AddHttpRequestHeader	Adds an HTTP header to the HTTP request. Use this method to add or change any header in the HTTP message.
RemoveHttpRequestHeader	Removes a specified HTTP header when the HTTP request message is sent.
GetHttpResponseHeader	Gets an HTTP header value (based on the header type) from the HTTP response message.

WCFClientCredential Class

The WCFClientCredential class provides client credentials for a service or a proxy server that is behind a firewall.

Properties

WCFClientCredential property	Type	Description
UserName	UserNameCredential (class)	Gets a credential object you can use to set the user name and password for a service or proxy server
HTTPDigest	HttpDigestCredential (class)	Gets the current HTTP digest credential
Windows	WindowsCredential (class)	Gets an object with the Windows credential you want to use to authenticate a client to a service
ClientCertificate	ClientCertificateCredential (class)	Specifies an object that provides an X.509 certificate that the client can use for authentication to a service
ServiceCertificate	ServiceCertificateCredential (class)	Specifies an object that provides an X.509 certificate when the message security mode is set for message-level encryption

WCFConnection Class

The WCFConnection class specifies the properties required for a WCF client to connect to a WCF or Web service. WCFConnection is the base class for the WCFConnection object.

Properties

WCFConnection property	Type	Description
BindingType	WCFBindingType (enumeration)	Specifies the binding type and communciation format to use when accessing a service. The default value is generated from the service contract. If you delete this value, the connection is attempted using BasicHttpBinding as the default binding type. Values are: `BasicHttpBinding`, `wsHttpBinding`, `NetTcpBinding`, and `NetNamedPipeBinding`. WCF connection bindings are described on the MSDN Web site: *http://msdn.microsoft.com/en-us/library/ms731092.aspx*.
ClientCredential	WCFClientCredential (class)	Specifies the WCF credential to use for communication with the service.
EndpointAddress	WCFEndpointAddress (class)	Specifies a URL for the remote service, and tells the service engine where the service resides. If the endpoint is not explicitly set, the service engine uses the default endpoint embedded in the service contract file and in the generated service assembly.
ProxyServer	WCFProxyServer (class)	Allows you to specify credentials for a proxy server if the client machine is behind a firewall. If the client machine is directly connected to the Internet, you need not set this property.

WCFConnection property	Type	Description
Timeout	String	Specifies how long the WCF engine waits, while invoking the service, before timing out. The default value is 10 minutes ("00:10:00"). The format for the time is "hh:mm:ss". The service might have a different timeout value on the server side.
BasicHttpBinding	WCFBasicHttpBinding (class)	Defines binding settings for BasicHttpBinding. This property is used only when the BindingType property is BasicHttpBinding.
wsHttpBinding	WCFwsHttpBinding (class)	Defines binding settings for wsHttpBinding. This property is used only when the BindingType property is wsHttpBinding.
netTcpBinding	WCFNetTcpBinding (class)	Defines binding settings for NetTcpBinding. This property is used only when the BindingType property is NetTcpBinding.
netNamedPipeBinding	WCFNetNamedPipeBinding (class)	Defines binding settings for NetNamedPipeBinding. This property is used only when the BindingType property is NetNamedPipeBinding.
SoapMessageHeader	WCFSoapMessageHeader (class)	Defines SoapHeader items for the SOAP request message.

These binding types are not currently supported: `WS2007HttpBinding`, `WSFederationHttpBinding`, `WS2007FederationHttpBinding`, `NetMsmqBinding`, `WebHttpBinding`, `WSDualHttpBinding`, `NetPeerBinding`, and `MsmqIntegrationBinding`.

WCFEndpointAddress Class

The WCFEndpointAddress class provides a unique network address that a client uses to communicate with a service endpoint.

Properties

WCFEndpointAddress property	Type	Description
URL	String	Specifies the URI that identifies the endpoint location.
Identity	WCFEndpointIdentity (class)	Specifies the identity for the endpoint.

WCFEndpointIdentity Class

The WCFEndpointIdentity class defines the identity type and value of an endpoint, enabling authentication by clients exchanging messages with it.

Properties

WCFEndpointIdentity property	Type	Description
Type	WCFEndpointIdentityType (enumeration)	Specifies the type of identity. Values are: `NONE` (default), `UPN`, `SPN`, `DNS`, `RSA`, and `CERTIFICATE`.
IdentityValue	String	Specifies the identity value.

WCFSoapMessageHeader Class

WCFSoapMessageHeader adds a SOAP header for the service request. This class enables you to change the encryption algorithm and key used in the header.

Methods

WCFSoapMessageHeader method	Description
AddMessageHeaderItem	Adds a SoapHeader item. The WCF client then sets the SOAP header before sending out the request. The format for a SoapHeader item is: `<ItemName xmlns="ItemNamespace">EncryptedItemValue</ItemName>`
RemoveMessageHeaderItem	Removes the SoapHeader item with the name that you pass in the ItemName parameter of this method.
RemoveAllMessageHeaderItems	Removes all SoapHeader items.

WCFnetNamedPipeBinding Class

The WCFnetNamedPipeBinding class provides a secure binding that is optimized for on-machine use.

Properties

Note: NetNamedPipeBinding supports transactions and uses binary message encoding, transport security, and named pipes for message delivery. The default configuration for the NetNamedPipeBinding is for on-machine communication only.

WCFnetNamedPipeBinding property	Type	Description
MaxConnections	Int32 (PowerScript long)	Specifies a value for the maximum number of connections to be pooled for subsequent reuse on the client. The default is 10.
TransactionFlow	Boolean	Specifies whether transaction flowing is supported. Values are: • `true` – supported. • `false` (default) – not supported.

WCFnetNamedPipeBinding property	Type	Description
TransactionProtocol	TransactionProtocolType (enumeration)	Specifies the transaction protocol to use in flowing transactions. Values are: `DEFAULT`, `OLETRANSACTIONS` (default), `WSATOMICTRANSACTION11`, and `WSATOMICTRANSACTION_OCTOBER2004`.
TransferMode	WSTransferMode (enumeration)	Gets or sets a value to indicate whether messages are buffered or streamed. Values are: `BUFFERED` (default), `STREAMED`, `STREAMEDREQUEST`, and `STREAMEDRESPONSE`.
Security	NetNamedPipeSecurity (class)	Specifies the type of security to use with services configured for NetNamedPipeBinding. Values are: `NONE` and `TRANSPORT` (default).
ReaderQuotas	WCFReaderQuotas (class)	Gets or sets processing constraints based on the SOAP message complexity for endpoints with this binding type.
HostNameComparisonMode	WCFHostNameComparisonMode (enum)	When matching the URI, indicates whether to use the host name to reach the service. Values are: `StrongWildCard` (default), `Exact`, and `WeakWildCard`.
MaxBufferSize	Int32 (PowerScript long)	Specifies the maximum size for a buffer that receives messages from the channel. The default value is 65536.
MaxBufferPoolSize	Int64 (PowerScript longlong)	Specifies the maximum amount of memory allocated for the manager of the buffers required by the endpoints using this binding. The default value is 524288.
MaxReceivedMessageSize	Int64 (PowerScript longlong)	Specifies the maximum size for a message that can be processed by the binding. The default value is 65536.

WCFnetTCPBinding Class

The WCFnetTCPBinding class enables you to communicate with WCF Web services from a WCF client using NetTcpBinding. This binding is intended only for WCF-to-WCF communication.

Properties

WCFnetTCPBinding property	Type	Description
MaxConnections	Int32 (PowerScript long)	Specifies a value for the maximum number of connections to be pooled for subsequent reuse on the client. The default for netTcpBinding is 10.
ReliableSession	WCFReliableSession (class)	Specifies whether to enable reliable sessions (using the WS-ReliableMessaging protocol) and defines settings for these sessions when enabled.
TransactionFlow	Boolean	Specifies whether transaction flowing is supported. Values are: • `true` – supported. • `false` (default) – not supported.
TransactionProtocol	TransactionProtocolType (enumeration)	Specifies the transaction protocol to use in flowing transactions. Values are: `DEFAULT` (default), `OLETRANSACTIONS`, `WSATOMICTRANSACTION11`, and `WSATOMICTRANSACTION_OCTOBER2004`.
TransferMode	WSTransferMode (enumeration)	Gets or sets a value to indicate whether messages are buffered or streamed. Values are: `BUFFERED` (default), `STREAMED`, `STREAMEDREQUEST`, and `STREAMEDRESPONSE`.

WCFnetTCPBinding property	Type	Description
Security	NetTcpSecurity (class)	Specifies the type of security to use with services configured for NetTcpBinding. The default security mode is `TRANSPORT`.
ReaderQuotas	WCFReaderQuotas (class)	Gets or sets processing constraints based on the SOAP message complexity for endpoints with this binding type.
HostNameComparisonMode	WCFHostNameComparisonMode (enum)	When matching the URI, indicates whether to use the host name to reach the service. Values are: `StrongWildCard` (default), `Exact`, and `WeakWildCard`.
MaxBufferSize	Int32 (PowerScript long)	Specifies the maximum size for a buffer that receives messages from the channel. The default value is 65536.
MaxBufferPoolSize	Int64 (PowerScript longlong)	Specifies the maximum amount of memory allocated for the manager of the buffers required by the endpoints using this binding. The default value is 524288.
MaxReceivedMessageSize	Int64 (PowerScript longlong)	Specifies the maximum size for a message that can be processed by the binding. The default value is 65536.
ListenBackLog	Int32 (PowerScript long)	Specifies the maximum number of queued connection requests permitted. The default value is 10.

WCFProxyServer Class

If you connect to a service from behind a proxy server, you must first instantiate an object of the WCFProxyServer class. This class provides credential information to the proxy server.

Properties

WCFProxyServer property	Type	Description
ProxyAddress	String	Specifies a URL for the proxy server or firewall.
CredentialType	HttpProxyCredentialType (enumeration)	Specifies the credential type to use for the firewall. Values are: `None` (default), `Basic`, `Digest`, `NTLM`, `Certificate`, and `Windows`.
ClientCredential	WCFClientCredential (class)	Specifies the WCF credentials to be used for the firewall.
UseDefaultWebProxy	Boolean	Determines whether to use the default Web proxy settings, as defined for Internet Explorer. The default value is true.

WCFReaderQuotas Class

The WCFReaderQuotas class defines restrictions (quotas) for the complexity of message exchanges with service endpoints that you can set differently according to the binding type used.

Properties

WCFReaderQuotas property	Type	Description
MaxDepth	Int32 (PowerScript long)	Gets or sets a maximum node depth. The default value is 32.
MaxStringContentLength	Int32 (PowerScript long)	Gets or sets a maximum length for message strings returned from the service. The default value is 8192.
MaxArrayLength	Int32 (PowerScript long)	Gets or sets the maximum length for arrays returned from the service. The default value is 16384.

WCFReaderQuotas property	Type	Description
MaxBytesPerRead	Int32 (PowerScript long)	Gets or sets the maximum number of bytes returned from the service during a single call. The default value is 4096.
MaxNameTableCharCount	Int32 (PowerScript long)	Gets or sets the maximum number of characters permitted in a table name. Setting this value can help prevent buildup of large amounts of character data. The default value is 16384.

WCFReliableSession Class

The WCFReliableSession class allows you to use reliable messaging for your service connections, as long as the binding type you are using supports the WS-ReliableMessaging protocol. WCFReliableSession also lets you get and set message ordering and duration.

Properties

WCFReliableSession property	Type	Description
Enabled	Boolean	Specifies whether the connection uses the WS-ReliableMessaging protocol. The default for this property depends on the binding type used for the service connection. Values are: • `true` – reliable messaging is enabled. • `false` – reliable messaging is not enabled.
Ordered	Boolean	Specifies whether messages are delivered in the order they are sent. Values are: • `true` (default) – messages are delivered in the order sent. • `false` – message order is not specified.
Duration	Int64 (PowerScript longlong)	Specifies the duration, in seconds, of a reliable messaging sequence. The default value is 600 seconds (10 minutes).

WCFwsHttpBinding Class

The WCFwsHttpBinding class enables you to communicate with WCF or Web services using the wsHttpBinding binding. This binding provides transaction capability, reliable messaging, WS-Addressing, and message security.

Properties

WCFwsHttpBinding property	Type	Description
MessageEncoding	WSMessageEncoding (enumeration)	Specifies encoding for SOAP messages. Values are: `TEXT` (default) and `MTOM` (Message Transmission Optimization Mechanism).
TextEncoding	WSTextEncoding (enumeration)	Specifies character encoding for SOAP message text. Values are: `ASCII`, `BIGENDIANUNICODE`, `UNICODE`, `UTF32`, `UTF7`, and `UTF8` (default).
ReliableSession	WCFReliableSession (class)	Specifies whether to enable reliable sessions (using the WS-ReliableMessaging protocol), and defines settings for these sessions when enabled.
TransactionFlow	Boolean	Specifies whether transaction flowing is supported. Values are: • `true` – supported. • `false` (default) – not supported.
Security	wsHttpSecurity (class)	Gets or configures the security settings of a wsHttpBinding connection.
ReaderQuotas	WCFReaderQuotas (class)	Gets or sets processing constraints based on the SOAP message complexity for endpoints with this binding type.

WCFwsHttpBinding property	Type	Description
AllowCookies	Boolean	Indicates whether the client accepts cookies. Values are: • `true` – accepts cookies. • `false` (default) – does not accept cookies.
BypassProxyOnLocal	Boolean	Indicates whether to bypass the proxy server for local addresses. Values are: • `true` – bypasses the proxy server. • `false` (default) – does not bypass the proxy server.
HostNameComparisonMode	WCFHostNameComparisonMode (enum)	When matching the URI, indicates whether to use the host name to reach the service. Values are: `StrongWildCard` (default), `Exact`, and `WeakWildCard`.
MaxBufferPoolSize	Int64 (PowerScript longlong)	Specifies the maximum amount of memory allocated for the manager of the buffers required by the endpoints using this binding. The default value is 524288.
MaxReceivedMessageSize	Int64 (PowerScript longlong)	Specifies the maximum size for a message that can be processed by the binding. The default value is 65536.

Methods

WCFwsHttpBinding method	Description
AddHttpRequestHeader	Adds an HTTP header to the HTTP request. Use this method to add or change any header in the HTTP message.
RemoveHttpRequestHeader	Removes a specified HTTP header when the HTTP request message is sent.

WCFwsHttpBinding method	Description
GetHttpResponseHeader	Gets an HTTP header value (based on the header type) from the HTTP response message.

WindowsCredential Class

The WindowsCredential class provides credential information that allows a WCF client to use a proxy server or connect to a service that requires integrated Windows authentication.

The WindowsCredential class can be invoked by the Windows property of the WCFClientCredential class.

Properties

WindowsCredential property	Type	Description
UserName	UserNameCredential (class)	Specifies the user name, password, and domain of the client.
AllowsImpersonationLevel	ImpersonationLevel (enumeration)	Determines the impersonation level of the WCF client. Values are: `None` (default), `Anonymous`, `Identification`, `Impersonation`, and `Delegation`.
AllowNtlm	boolean	Indicates whether NT LAN Manager (NTLM) authentication is allowed as Windows Security Support Provider Interface (SSPI) Negotiate authentication. Values are: • `true` (default) – authentication is allowed. • `false` – authentication is not allowed.

wsHttpSecurity Class

The wsHttpSecurity class provides the security settings for communications between a WCF client and a service using the wsHttpBinding binding.

Properties

wsHttpSecurity property	Type	Description
SecurityMode	wsSecurityMode (enumeration)	Gets or sets the security mode for the binding. Values are: `None`, `Transport`, `Message` (default), and `TransportWithMessageCredential`.
Transport	HttpTransportSecurity (class)	Specifies the transport-level security for the binding.
Message	NoDualHttpMessageSecurity (class)	Specifies the message-level security for the binding.

WCF Client Methods

WCF client methods enable you to add and remove headers to HTTP and SOAP messages.

AddHttpRequestHeader Method

Adds a header to the HTTP request. Use this method to add or change any header in the HTTP message.

Syntax

```
client.wcfConnectionObject.Binding.AddHttpRequestHeader
(PBWCF.HTTPRequestHeaderType HeaderType!, string HeaderValue)
```

Parameters

- **Binding –** an instance of the WCFBasicHttpBinding or WCFwsHttpBinding class.
- **HeaderType –** an enumerated value of type PBWCF.HttpRequestHeaderType.
- **HeaderValue –** a string for the header value.

Returns

None.

Examples

- **–** Add a cookie and a user agent name to the HTTP request header for a BasicHttpBinding connection with code like this:

```
client.wcfConnectionObject.BasicHttpBinding.AddHttpRequestHeader(
PBWCF.HttpRequestHeaderType.Cookie!, "testing")
client.wcfConnectionObject.BasicHttpBinding.AddHttpRequestHeader(
PBWCF.HttpRequestHeaderType.UserAgent!, "john")
```

AddMessageHeaderItem Method

Adds a SoapHeader item to the SOAP header before the WCF client sends a request.

Syntax

```
client.wcfConnectionObject.SoapMessageHeader.AddMessageHeaderItem
(string ItemName, string ItemNamespace, string ItemValue,
PBWCF.WCFHMAC ItemEncryptAlgorithm!, string EncryptKey)
```

Parameters

- **SoapMessageHeader –** an instance of the WCFSoapMessageHeader class.
- **ItemName –** a string for the name of the SoapHeader item to add.
- **ItemNamespace –** a string for the namespace of the SoapHeader item.
- **ItemValue –** a string for the value of the SoapHeader item.
- **ItemEncryptAlgorithm –** an enumerated value of the type PBWCF.WCFHMAC for the algorithm to encrypt the ItemValue.
- **EncryptKey –** a string for the key to encrypt the ItemValue.

Returns

Boolean. Returns true if successful, otherwise returns false.

Examples

- **Setting values in the wcfConnectionObject –** Add a header to a request message sent to Amazon Web Services with code like this:

```
client.wcfConnectionObject.SoapMessageHeader.AddMessageHeaderItem
("AWSAccessKeyId","http://security.amazonaws.com/doc/
2007-01-01/","My access key",PBWCF.WCFHMAC.NONE!,"")
client.wcfConnectionObject.SoapMessageHeader.AddMessageHeaderItem
("Signature","http://security.amazonaws.com/doc/2007-01-01/","My
secret key", PBWCF.WCFHMAC.HMACSHA256!, "")
```

- **Setting values in an instance of WCFSoapMessageHeader –** Add a header sent to the same service using:

```
WCFSoapMessageHeader Header
Header = create WCFSoapMessageHeader
Header.AddMessageHeaderItem("AWSAccessKeyId","http://
security.amazonaws.com/doc/2007-01-01/","My access
key",PBWCF.WCFHMAC.NONE!,"")
Header. .AddMessageHeaderItem("Signature","http://
security.amazonaws.com/doc/2007-01-01/","My secret key",
PBWCF.WCFHMAC.HMACSHA256!, "")
client.wcfConnectionObject.SoapMessageHeader = Header
```

Usage

If you use encryption, the format for a SoapHeader item in the request sent by the WCF client is:

```
<ItemName xmlns="ItemNamespace">EncryptedItemValue</ItemName>
```

GetHttpResponseHeader Method

Returns a header from an HTTP request.

Syntax

```
svc.wcfConnectionObject.Binding.GetHttpResponseHeader
(PBWCF.HttpResponseHeaderType.HeaderType!)
```

Parameters

- **Binding –** an instance of the WCFBasicHttpBinding or WCFwsHttpBinding class.
- **HeaderType –** an enumerated value of type PBWCF.HttpResponseHeaderType.

Returns

String. If the header does not exist in the response message, GetHttpResponseHeader returns an empty string.

Examples

- **–** Return a date from the HTTP response header with code like this:

```
String dt
dt =
svc.wcfConnectionObject.BasicHttpBinding.GetHttpResponseHeader(PB
WCF.HttpResponseHeaderType.Date!)
```

RemoveAllMessageHeaderItems Method

Removes all SoapHeader items from the SOAP header before the WCF client sends a request.

Syntax

```
client.wcfConnectionObject.SoapMessageHeader.RemoveAllMessageHeader
Items ()
```

Parameters

- **SoapMessageHeader –** an instance of the WCFMessageHeader class.

Returns

Boolean. Returns true if successful, otherwise returns false.

RemoveHttpRequestHeader Method

Removes a header from an HTTP request.

Syntax

```
client.wcfConnectionObject.Binding.RemoveHttpRequestHeader
(PBWCF.HTTPRequestHeaderType HeaderType!)
```

Parameters

- **Binding –** an instance of the WCFBasicHttpBinding or WCFwsHttpBinding class.
- **HeaderType –** an enumerated value of type PBWCF.HttpRequestHeaderType.

Returns

None.

RemoveMessageHeaderItem Method

Removes a SoapHeader item from the SOAP header before the WCF client sends a request.

Syntax

```
client.wcfConnectionObject.SoapMessageHeader.RemoveMessageHeaderIte
m (string ItemName)
```

Parameters

- **SoapMessageHeader –** an instance of the WCFSoapMessageHeader class.
- **ItemName –** a string for the name of the SoapHeader item to remove.

Returns

Boolean. Returns true if successful, otherwise returns false.

WCF Client System Constants

PowerBuilder .NET system enumerations enable you to get and set values for connecting a WCF client to a proxy server or a Web service.

Like other types of PowerScript enumerations, you must use the "bang" (exclamation point) character for all WCF client enumerated values.

BasicHttpMessageCredentialType Enumeration

Specifies the credential type required by a binding for authentication. Used only for BasicHttpBinding bindings.

Enumerated values

BasicHttpMessageCredentialType value	Meaning
UserName	Specifies client authentication using UserName
Certificate	Specifies client authentication using a certificate

BasicHttpSecurityMode Enumeration

Stores or supplies security mode settings for a WCF client using BasicHttpBinding to bind to a Web service.

Enumerated values

BasicHttpSecurityMode value	Meaning
None	The SOAP message is not secured during transfer. This is the default behavior for WCF client binding.
Transport	HTTPS provides security for the SOAP message, and the client authenticates the service using the service's SSL certificate. The ClientCredentialType property of the BasicHttpMessageSecurity or the HttpTransportSecurity class controls client authentication to the service.
Message	SOAP message security provides client authentication. The server SSL certificate must be provided to the client separately.
TransportWithMessageCredential	HTTPS provides for message integrity, confidentiality, and server authentication. SOAP message security assures client authentication. The service must be configured with an SSL certificate.

BasicHttpSecurityMode value	Meaning
TransportCredentialOnly	Client authentication is provided by HTTP, but message integrity and confidentiality are not provided. Use this mode only when other means of transfer security (such as IPSec) are available.

CertStoreLocation Enumeration

Specifies the location of the X.509 certificate store containing a certificate to use for secure communication from a WCF client.

Enumerated values

CertStoreLocation value	Meaning
CurrentUser	Specifies the X.509 certificate store assigned to the current user.
LocalMachine	Specifies the X.509 certificate store assigned to the local machine.

CertStoreName Enumeration

Specifies the name of the X.509 certificate store containing a certificate to use for secure communication from a WCF client.

Enumerated values

CertStoreName value	Meaning
AddressBook	Specifies the X.509 certificate store for other users
AuthRoot	Specifies the X.509 certificate store for third-party certificate authorities
CertificateAuthority	Specifies the X.509 certificate store for intermediate certificate authorities
Disallowed	Specifies the X.509 certificate store for revoked certificates
My	Specifies the X.509 certificate store for personal certificates
Root	Specifies the X.509 certificate store for trusted root certificates
TrustedPeople	Specifies the X.509 certificate store for directly trusted people and resources
TrustedPublisher	Specifies the X.509 certificate store for directly trusted publishers

HttpClientCredentialType Enumeration

Provides the type of credential to use for transport-level security when connecting to a Web service that uses the BasicHttpBinding binding.

Enumerated values

HttpClientCredentialType value	Meaning
None	Specifies anonymous authentication
Basic	Specifies Basic authentication
Digest	Specifies Digest authentication
NTLM	Specifies client authentication using NTLM (NT LAN Manager)
Windows	Specifies client authentication using Windows
Certificate	Specifies client authentication using a certificate

HttpProxyCredentialType Enumeration

Provides the type of credential required by a proxy server for clients that connect to a Web service from behind a firewall.

Enumerated values

HttpProxyCredentialType value	Meaning
None	No credentials are presented or used
Basic	Client uses basic authentication to connect to the proxy server
Digest	Client uses digest authentication to connect to the proxy server
NTLM	Client uses NTLM (NT LAN Manager) authentication to connect to the proxy server
Windows	Client uses integrated Windows authentication to connect to the proxy server

HttpRequestHeaderType Enumeration

Used by the AddHttpRequestHeader method to add a header to an HTTP message, and by the RemoveHttpRequestHeader method to remove a header from an HTTP message.

Enumerated values

The HttpRequestHeaderType enumeration supports these header types, which are described on the MSDN Web site at *http://msdn.microsoft.com/en-us/library/system.net.httprequestheader.aspx*:

Accept
AcceptCharset
AcceptEncoding
AcceptLanguage
Allow
Authorization
CacheControl
Connection
ContentEncoding
ContentLanguage
ContentLength
ContentLocation
ContentMd5
ContentRange
ContentType
Cookie
Date
Expect
Expires
From
Host
IfMatch
IfModifiedSince
IfNoneMatch
IfRange
IfUnmodifiedSince
KeepAlive
LastModified
MaxForwards
Pragma

ProxyAuthorization
Range
Referer
Te
Trailer
TransferEncoding
Translate
Upgrade
UserAgent
Via
Warning

Note: Enumerated values must be followed by an exclamation mark (!) when passed as method parameters.

HttpResponseHeaderType Enumeration

Used by the getHttpResponseHeader method to specify a header type and get its header value from an HTTP response message.

Enumerated values

The HttpResponseHeaderType enumeration supports these header types:

AcceptRanges
Age
Allow
CacheControl
Connection
ContentEncoding
ContentLanguage
ContentLength
ContentLocation
ContentMd5
ContentRange
ContentType
Date
ETag
Expires
KeepAlive
LastModified
Location

Pragma
ProxyAuthenticate
RetryAfter
Server
SetCookie
Trailer
TransferEncoding
Upgrade
Vary
Via
Warning
WwwAuthenticate

Note: Enumerated values must be followed by an exclamation mark (!) when passed in a method parameter.

ImpersonationLevel Enumeration

Supplies identification values to a proxy server or a Web service for a WCF client using integrated Windows or digest authentication.

Enumerated values

ImpersonationLevel value	Meaning
None	An impersonation level is not assigned.
Anonymous	The server process cannot impersonate the client and cannot obtain identification information about the client.
Identification	The server process cannot impersonate the client, but can obtain identification and privileges information about the client. This allows other services on the server to use the client security context for access and validation decisions.
Impersonation	The server process can impersonate the client security context on its local system, but not on remote systems.
Delegation	The server process can impersonate the client security context on both local and remote systems.

MessageCredentialType Enumeration

Specifies the credential type required by a binding for authentication. Used by all standard binding types except BasicHttpBinding.

Enumerated values

MessageCredentialType value	Meaning
None	Specifies anonymous authentication
Windows	Specifies client authentication using Windows
UserName	Specifies client authentication using UserName
Certificate	Specifies client authentication using a certificate
IssuedToken	Specifies client authentication using an issued token

ProtectionLevel Enumeration

Provides the security service requested for an authenticated stream. This enumeration is used by the TcpTransportSecurity class for NetTcpBinding bindings, and by the NamedPipeTransportSecurity class for NetNamedPipeBinding bindings.

Enumerated values

ProtectionLevel value	Meaning
None	Specifies authentication only
Sign	Signs data to ensure integrity of transmitted data
EncryptAndSign	Encrypts and signs data to ensure confidentiality and integrity of transmitted data

TcpClientCredentialType Enumeration

Provides the type of credential to use for transport-level security when connecting to a Web service that uses the NetTcpBinding binding.

Enumerated values

TcpClientCredentialType value	Meaning
None	Specifies anonymous authentication
Windows	Specifies client authentication using Windows
Certificate	Specifies client authentication using a certificate

WCFBindingType Enumeration

Defines the binding and communication formats to use when accessing a WCF service.

Enumerated values

WCFBindingType value	Description
BasicHttpBinding	Suitable for communication with services, such as ASP.NET (ASMX-based) Web services, that conform to the Basic Profile 1.0 protocol. Uses HTTP for transport, and text/XML for message encoding.
NetTcpBinding	Suitable for communication between WCF applications. Uses Transport Layer Security (TLS) over TCP for message transport security, and supports duplex contracts.
NetNamedPipeBinding	Binding that is similar to the NetTcpBinding binding, but supports only on-machine communication. Uses named pipes for message delivery and binary message encoding.
wsHttpBinding	Suitable for communication with nonduplex Web services (services without a callback contract). Implements WS-Reliable Messaging and WS-Security protocols. Uses HTTP for transport, and text/XML for message encoding.

WCFEndpointIdentity Enumeration

Defines the identity to use with an endpoint for authentication.

Enumerated values

WCFEndpointIdentity value	Meaning
None	No identity is needed.
UPN	User Principal Name (UPN) is an identity that is used with the `SSPINegotiate` authentication mode. The value for a UPN identity takes the format of an e-mail address, with a user account name, and a name that identifies the domain of the user separated by the commercial at (@) character. For example, UserName@Sybase.com.

WCFEndpointIdentity value	Meaning
SPN	Service Principal Name (SPN) is a name by which a client uniquely identifies a service instance. You can use this identity type with three different authentication modes: `SSPINegotiate`, `Kerberos`, and `KerberosOverTransport`. The value for a UPN identity takes the format of an e-mail address, with a user account name, and a name that identifies the domain of the user separated by the @ character. For example, UserName@Sybase.com.
DNS	Domain Name System (DNS) specifies the expected identity of the server. The identity value is a domain name, such as Sybase.com.
RSA	RSA is a public-key encryption algorithm for signing and encrypting messages. It also requires a private key for decrypting messages. For an RSA identity, you must specify the public key for the identity value.
CERTIFICATE	Use the Certificate identity when a certificate is required for authentication to the service endpoint. For the identity value, specify a Base64 encoded string representing the raw data of the certificate.

WCFHMAC Enumeration

Defines all supported encryption algorithms for the SoapHeader used in SOAP request messages from the WCF client.

Enumerated values

MessageCredentialType value	Meaning
None	No encryption is used
HMACMD5	Specifies the MD5 hash function
HMACRIPEMD160	Specifies the RIPEMD160 hash function
HMACSHA1	Specifies the SHA1 hash function
HMACSHA256	Specifies the SHA256 hash function
HMACSHA384	Specifies the SHA384 hash function
HMACSHA512	Specifies the SHA512 hash function
MACTRIPLEDES	Specifies TripleDES for the input data

Note: The encryption key must be specified in the SOAP header. If an encryption key is not provided, PowerBuilder uses the default key. The default key combines the Web service method name with the timestamp in the format “yyyy-MM-ddTHH:mm:ssZ”. For example,

for the ItemSearch method from Amazon.com that you call at 11:58:10AM (GMT) on April 27, 2010, the default key is “ItemSearch2010-04-27T11:58:00Z”.

wsSecurityMode Enumeration

Defines security mode settings for a WCF client using wsHttpBinding to bind to a Web service.

Enumerated values

wsSecurityMode value	Meaning
None	Security is disabled
Transport	Security is provided using secure transport, such as HTTPS
Message	SOAP message security is enabled
TransportWithMessageCredential	Transport security provides integrity and confidentiality, and SOAP message security provides client authentication

WSTransferMode Enumeration

Gets or sets a value indicating whether SOAP request and response messages are buffered or streamed.

Enumerated values

WSTransferMode value	Meaning
Buffered	The request and response messages are both buffered
Streamed	The request and response messages are both streamed
StreamedRequest	The request message is streamed and the response message is buffered
StreamedResponse	The request message is buffered and the response message is streamed

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X