SYBASE®
An SAP Company

New Features

PowerBuilder® 12.5

DOCUMENT ID: DC00357-01-1250-01
LAST REVISED: July 25, 2011

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

Contents

PowerBuilder Classic and .NET

This section describes new features that are common to both PowerBuilder® Classic and PowerBuilder .NET.

AutoWidth Property

In the PowerBuilder Classic DataWindow and PowerBuilder .NET WPF DataWindow, the AutoWidth property allows you to choose the option to automatically compute the width of a column in a grid style DataWindow.

Applies to
Columns in grid style DataWindows

Usage
The AutoWidth property takes one of these numeric values:

- 0 - No AutoWidth: This is the default value.
- 1 - AutoWidth is computed for visible rows (monotonic) and does not decrease when the widest column is reduced when scrolling.
- 2 - AutoWidth is computed for visible rows (non-monotonic)
- 3 - AutoWidth is computed for all retrieved rows.

You can set the AutoWidth property:

- In the painter - in the Properties view, select one of the values in the drop-down list for the AutoWidth property.
- In scripts - set the AutoWidth property to one of the numeric values.

Support for Tab Sequence, Enabled, and Show Focus Rectangle Properties

These properties are supported in the PowerBuilder Classic DataWindow. PowerBuilder .NET WPF DataWindow supports only Tab Sequence and Enabled.

The Tab Sequence property is not expressionable, but Enabled and Show Focus Rectangle are.

- The default value of Tab Sequence is zero for all non-column controls, replicating the previous behavior. You can edit it as needed.
- The default value for the Enabled property is "yes."

- The default value of the Show Focus Rectangle property is "no."

Note: The Show Focus Rectangle property is supported only in PowerBuilder Classic.

The following DataWindow controls now have the Tab Sequence, Enabled, and Show Focus Rectangle properties:

- Button (Show Focus Rectangle does not apply)
- Computed Field
- Graph
- Picture
- TableBlob
- Text

The Column control does not have the Enabled property because it has existing mechanisms which achieve the same effect.

PowerBuilder Classic supports these properties for OLE Objects and OLE Database Blobs.

Support for New ASE 15.5 Datatypes

PowerBuilder supports the new datatypes in ASE 15.5: BIGTIME and BIGDATETIME.

- BIGTIME: Includes the hour, minute, second, and fraction of a second. The fraction is stored to six decimal places.
- BIGDATETIME: Includes the year, month, day, hour, minute, second, and fraction of a second. The fraction is stored to six decimal places.

PowerBuilder Classic

This section describes new features for the PowerBuilder Classic IDE only.

Window Control transparent Value and transparency property

PowerBuilder 12.5 introduces two new features for window controls

- transparent value for window control BackColor
- transparency property for window control

The following controls support the new value and property:

PictureButton

CheckBox
RadioButton
StaticHyperlink
GroupBox
SingleLineEdit
EditMask
MultiLineEdit
RichTextEdit
DropDownListBox
DropDownPictureListBox
ListView
TreeView
Tab
Graph
UserObject

Sharing Datasources with .NET

New support is added for sharing ADO.NET connections

PowerBuilder 12.5 provides the ability to share ADO.NET connections between PowerBuilder (Win32) and third-party .NET assemblies exposed in COM. A PowerBuilder native application provides the capability to import an ADO.NET connection from a third-party .NET assembly and to export an ADO.NET connection to a third-party assembly. This feature already exists in PowerBuilder.NET, but the PowerBuilder Classic IDE supports a native version of this feature using .NET COM Interop.

Two methods have been added to the PowerBuilder Transaction NVO:

- bool SetAdoConnection(oleobject connectionProxy)
  Imports an external ADO.NET connection.
- oleobject GetAdoConnection()
  Exports an ADO.NET connection from a connected PB Transaction instance.

You can invoke these two method to get or set the proxy to share an ADO.NET DB Connection between a PB app and third-party .NET assembly.

RTF and Images in the DataWindow Object

PowerBuilder 12.5 introduces the RTF, Image, and XPS Database Blobs, as well as RichText and RichTextFile expression functions for Computed Fields.

The RTF, Image, and XPS Database Blob feature is a port of the existing feature from PowerBuilder .NET.

The RichText expression function takes as argument a string expression interpreted as RTF file and renders it as such. If the argument is not RTF, nothing renders.

The RichTextFile takes as argument a string expression interpretation of the name of the RTF file and renders it as such. If the argument is not an RTF file, nothing renders.

User-Drawn Controls in DataWindow Objects

The Paint expression functions allow you to draw objects in the DataWindow such as polygons, arrow tips, pie slices, and so on.

The Paint expression function takes one string expression argument and returns the same string. This allows you to paint inside a DataWindow in a way that respects the position and z-order of other DataWindow objects.

Syntax

`Paint(expr)` where *`expr`* can be any valid DataWindow expression. It should contain a function call to a drawing global function with rendering logic. If *`expr`* is a string expression and the value is not null, the Computed Field will render the evaluated string expression.

This feature also provides the following supporting functions:

- GetPaintDC()
- GetPaintRectX()
- GetPaintRectY()
- GetPaintRectWidth()
- GetPaintRectHeight()

`GetPaintDC()` returns the GDI context to which to draw. The clip region of the GDI context is guaranteed to be the same as the rectangle defined by the other functions.

`GetPaintRectX()`, `GetPaintRectY()`, `GetPaintRectWidth()`, and `GetPaintRectWidth()` return the bounds of the computed field. The device context is clipped within these bounds.

This example instantiates the drawing functions and, if the drawing function returns false, the text "Bail out" displays.

```
Paint
(
```

```
        MyDrawPieSlice
        (
                GetPaintDC(),
                GetPaintRectX(),
                GetPaintRectY(),
                GetPaintRectWidth(),
                GetPaintRectHeight(),
                GetRow()*100/RowCount()
        )
)
```

```
Paint
(
        MyDrawPieSlice
        (
                GetPaintDC(),
                GetRow()*100/RowCount()
```

```
Paint
(
if (MyDrawPieSlice(getpaintdc()),"","Bail out")
)
```

PowerBuilder .NET

This section describes new features for the PowerBuilder .NET IDE only.

.NET 4.0 Framework

PowerBuilder upgrades the runtime library for .NET 4.0

PowerBuilder 12.5 was built using the Microsoft .NET 4.0 Framework. This means that the runtime library has been upgraded to take advantage of the improvements offered, such as improved security for .NET applications, improved exception handling within the framework, as well as using WPF 4 enhancements for the WPF Controls.

Visual Studio 2010 Isolated Shell

PowerBuilder .NET 12.5 uses the Visual Studio 2010 isolated shell Service Pack 1.

The new shell comes with enhancements.

- Windows are no longer constrained to the editing frame of the integrated development environment (IDE). You can now dock them to the edges, float outside the IDE, or move them anywhere on the desktop. You can even move them to another monitor.
- The Help Viewer was completely redesigned. The documentation now appears in your Web browser or browser frame.

Script Editor Changes

The PowerScript editor has been enhanced.

Improvements include better semantic checking and support for identifiers, expressions and assignment statements.

Batch Command Processing

PowerBuilder .NET introduces batch build processing.

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

See *Batch Command Processing* in the *PowerBuilder .NET Features Guide* for more information.

Multithreading Support

PowerBuilder enhances runtime support for multithreaded applications.

Applications that use shared objects can run in multithreaded environments. Previously, the PowerBuilder .NET runtime library did not provide a way to synchronize data, so the data was susceptible to corruption. Additionally, locking mechanisms for accessing static fields in previous versions of PowerBuilder made assemblies using NVO instances susceptible to problems associated with reentrance.

In PowerBuilder 12.5, the runtime library has been changed to solve the NVO assembly problem with static fields. You can also use .NET Thread and .NET thread synchronization functions for multithreaded applications.

See *Using Multithreading* for support in the PowerBuilder .NET IDE, or *Using .NET Synchronization* for PowerBuilder Classic.

Targets

This section describes new features for the PowerBuilder .NET IDE only.

WCF Services

A new PowerBuilder target builds applications using the Microsoft WCF model

Currently, PowerBuilder only creates traditional, WS-I profile conformant Web Services in PowerBuilder. This is leveraged through the ASP.NET framework and built on top of PB.NET Web Forms applications and runtime. It is made up of an .asmx file and an ASP.NET Web

Service class that is built into a separate .NET assembly, both deployed to a virtual folder of an IIS server. A traditional ASP.NET Web Service has limited functionalities; for example, it supports only HTTP and HTTPS transports, and XML message encoding and transport level security.

WCF is the Microsoft unified 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 transports, including HTTP(S), TCP, MSMQ, and Named Pipes. It also provides both transport and message level security, as well as many other WS enhancement specs, for supporting such things as Reliable Messaging, Transaction. PowerBuilder 12.5 provides a new target to take advantage of this new programming model.

Support for CVUOs in .NET Assemblies

The .NET Assembly target exposes custom visual user objects.

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 choose to 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 unselect Export only the CVO first.

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.

Support for Split Assemblies in WPF Targets

The WPF Target provides the means to generate multiple assemblies from a build – one EXE and one or more DLLs – instead of a single EXE.

The Project painter has two additional tabs: Assemblies and Dependencies.

Use the Assemblies tab to determine what assemblies should be built from the target's PBLs. For each PBL in the WPF Target library list, you can select an output assembly. The drop-down lists in the second column of the list view contain all the PBL names with ".DLL" appended. Selecting an assembly name in the second column indicates that the PBL will be included in an output assembly of that name. More than one PBL from the left column can be

built to the same output assembly by selecting the same output name. Leaving the second column value blank indicates the PBL will be built to the executable. You can also type a name in the second column. The name is added to all drop-down lists, allowing multiple PBLs to be built to the same output assembly of the given name.

The read-only Dependencies tab indicates how the output assemblies depend on one another to determine build order. You can update the page by running the dependency checker.

The Dependency Check

Detailed dependencies information is available in the Assembly Dependencies tool window after completing a full dependency check process. The dependency check process can be triggered:

- manually from the menu Design > Check Dependencies or the WPF project object's context menu, or
- automatically by a full build process after changing the settings of the output assemblies.

All dependency errors indicated in the output window need to be solved before doing a full build for the target.

To solve dependency errors (dependency on executable or circular dependency), you can try the following methods.

1. Combine the PBLs that have dependency errors into one output assembly.
2. Move the objects from one PBL to another PBL.
3. Build global variables into an output assembly if they are heavily used in different objects and different PBLs.
4. Refactor the code to use another object; for example, use local or instance variables instead of global variables.
5. Move heavily used objects from many different PBLs into a new PBL and then build to a single output assembly.
6. Solve dependency errors on global external functions by using local external functions instead. This is due to a current limitation.

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 iteself; 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
- Can be identified by having an "AssemblyExtraInfo(true)" attribute
- 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.

REST client

PowerBuilder 12.5 includes a new REST client.

Using the REST client feature involves these steps:

1. Generate a PB proxy NVO using a REST client proxy project.
   PowerBuilder generates a proxy NVO for each remote RESTful Service, using a URI, method, and messaging format that you specify.
2. Parse complex message schemas and generate assemblies.
   PowerBuilder runs XSD.exe from the Windows SDK for .NET Framework 4.0 to parse or generate the schemas, and then to generate C# files. It then uses the C# compiler to build an assembly. PowerBuilder generates one assembly for each request message and one for the response message, if needed. These assemblies are automatically added to the PowerBuilder WPF target references when the project is deployed.
   You might not need to create assemblies for simple request or response messages like integer or string types. Instead, you can skip the assembly generation and use PowerBuilder primitive types.
3. Instantiate the proxy NVO and call the service.

Language Enhancements

This section describes new features for the PowerBuilder .NET IDE only.

Consume .NET Events

You can use PowerScript to dynamically connect methods to .NET events.

You can connect single or multiple methods to a .NET event in PowerScript. The method could be a PowerBuilder global function, an instance function of a PowerBuilder object, or an instance or static function of a .NET object. You can also connect PowerBuilder events to .NET events in PowerScript.

You are not able to declare a variable or a parameter with a .NET event because a .NET event is not a type at all. You are also not able to declare a new .NET event in PowerBuilder code; you

can only consume a .NET event defined in other .NET languages with the += operator in PowerBuilder.

The following examples demonstrate how to dynamically hook up to a .NET event in PowerScript.

'Clicked' is a .NET event defined in the third-party .NET control 'system.windows.controls.button'. Suppose 'OnClick1' is a PowerBuilder event that has exact same signature as the 'Clicked' .NET event of 'System.Windows.Controls.Button' control. With this enhancement, you can support the following use cases.

Define a PowerBuilder event 'OnClick1':

```
Event OnClick1(system.object sender, RouteEventArgs e)
```

Connect 'OnClick1' to the 'Clicked' .NET event of an instance of 'System.Windows.Controls.Button' control:

```
System.Windows.Controls.Button cb1
cb1 = create System.Windows.Controls.Button()
cb1.clicked += OnClick1
```

Connect a PowerBuilder event to a .NET event of the InnerControl of a PowerBuilder control:

```
System.Windows.Controls.Button b
b = cb_1.InnerControl
b.Click += OnClick1
```

Enhancement of the .NET Assembly Target

The .NET Assembly is expanded to expose additional PB language elements for use in .NET application development.

With the enhanced Assembly target, you are able to restructure your applications and provide a development framework to your users.

The Objects tab of the Project painter exposes additional language elements:

- Functions with generic types or delegates as parameters
- Parameterized constructors
- Events
- Indexers
- .NET properties
- Instance variables

Indexers and constructors cannot be renamed.

Table 1. Additional language elements in .NET Assembly

Additional language elements	NVOs	VOs	Can be renamed?	Can have description?
Functions with generic types or delegates as parameters	Yes	Yes	Yes	Yes
Parameterized constructors	Yes	No	No	Yes
Events	No	Yes	Yes	Yes
Indexers	Yes	Yes	No	Yes
.NET properties	Yes	Yes	Yes	Yes
Instance variables	Yes	Yes	Yes	Yes

PowerScript Syntax

The following examples demonstrate the PowerScript syntax for those additional language elements that can be exposed in the .NET assembly.

- Functions with generic types or delegates as parameters: `public function integer f1(System.Collection.List<string> s)`
- Parameterized constructors: `event constructor(string s)`
- Indexers: `public integer this[Integer i][get,set]`
- .NET properties: `public integer p1[get,set]`
- Instance variables: `public string s1`

WPF DataWindow Enhancements

This section describes new features for the PowerBuilder .NET IDE only.

Optimized SRD Syntax

Based on the design of the native DataWindow, the redundant syntax of Number, Boolean, and String properties has been omitted in generated SRD syntax.

This rule is applied to expression properties.

Data Buffers in the Debugger

The debugger visualizers for DataWindows can be used to view the primary, filtered, and deleted data buffers.

The DataWindow visualizer and ListView visualizer display the values of the corresponding buffer using either the DataWindow style or a simple list.

The DataWindow visualizer does not support crosstab, composite styles and report controls. The ListView visualizer does not support composite style or report controls. The blob data is displayed as System.Byte[] in the ListView visualizer.

Child DataWindow Control Support

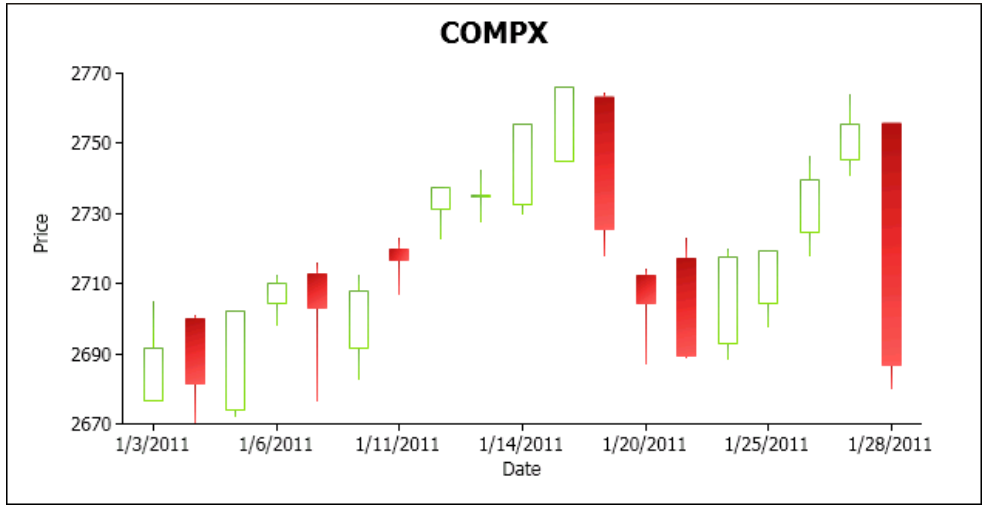
The Child DataWindow control can be nested for many levels.

When a nested report is inserted into a DataWindow, a print preview-mode DataWindow control is used. A Child DataWindow is editable if you use the control in a Composite style DataWindow.

Candlestick Graphs

Use the candlestick graph to visually display 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.

Global Functions List

Global functions are listed in the expression dialog for DataWindow objects.