SYBASE®

An SAP® Company

New Features

PowerBuilder® 12.5.1

DOCUMENT ID: DC00357-01-1251-01
LAST REVISED: January 31, 2012

Contents

64-bit Deployment

You can create PowerBuilder applications to deploy on 32-bit and 64-bit systems.

Migrated applications will default to “32-bit.” You can also set the Platform Target build option to “32-bit or 64-bit.” The latter option allows you to use 32-bit libraries in applications you will deploy as a 64-bit.

Other targets—WCF Service, WPF Web Service DataWindow, WCF Client, REST Client—are all callees, so the application type determines the running mode. You cannot select the platform; they build as “AnyCPU” mode.

Compilation

Depending on the Platform Target setting, the application you developed compiles in different ways.

- 32-bit (default): compiles the application or assembly to run by any 32-bit, x86-compatible common language runtime
- 32-bit or 64-bit: compiles an application or assembly that runs on any Windows platform

Microsoft uses the names "x86" and "AnyCPU."

Performance

64-bit deployment gives you an interface to develop different types of applications, especially applications for 64-bit Windows platforms. This kind of application can use more than 4GB of RAM, and has better performance than their 32-bit counterpart.

Limitation in 12.5.1

Some PowerBuilder features are not available when running a “32-bit or 64-bit” application on a 64-bit platform.

- Some PowerBuilder assemblies developed using C++ will be loaded at runtime accompany these features. We will only provide one 32-bit version of these C++ assemblies with PowerBuilder .NET 12.5.1.

  The assemblies are:
  - Sybase.PowerBuilder.Interop.dll
  - Sybase.DataWindow.Interop.dll
  - Sybase.PowerBuilder.DataSource.Db.dll
  - Sybase.PowerBuilder.DataWindow.Interop.dll
  - Sybase.PowerBuilder.Editmask.Interop.dll
  - Sybase.PowerBuilder.Graph.Interop.dll
  - Sybase.PowerBuilder.RTC.Interop.dll
- The following features are not supported in 64-bit PB WPF application because of the unsupported assemblies (listed above):

    - The OLE related objects such as PBOmObject, PBOleObject, PBOmControl, PBOleControl and PBOleCustomControl
    - PBMailSession
    - Some string functions that end with ‘A’, such as FillA(), LeftA(), LenA(), MidA(), PosA(), ReplaceA() and RightA()
    - Read PSR files and save data in Excel 8 format in DataWindow
    - No non-ADO.NET drivers are supported
    - PBNI
    - MobiLink synchronization
    - Any calls into assemblies listed here are not supported
    - InkEdit feature in DataWindow
    - Pipeline
- For WCF WebService projects, compiled assemblies behave like WPF assemblies. They are compiled using AnyCPU mode. If a WCF WebService references 32-bit libraries or assemblies, the hosting IIS or application pool should be configured as “Enable 32-bit application.” Once configured like this, IIS will run on x64 environment in WOW64 mode. That means the IIS or application pool runs as WOW64 mode, so it can hold 32-bit web services.

Error Messages and Runtime Exceptions

The most common exception caused by this feature is BadImageFormatException. It is thrown when the file image of a dynamic link library (DLL) or an executable program is invalid.

Main application format	Platform	Format of the assemblies loaded at runtime	Remarks
32-bit	32-bit	32-bit, 32-bit or 64-bit	
32-bit	64-bit	32-bit, 32-bit or 64-bit	WOW64
32-bit or 64-bit	32-bit	32-bit, 32-bit or 64-bit	
32-bit or 64-bit	64-bit	64-bit, 32-bit or 64-bit	

The main application determines the runtime environment in the whole application lifecycle. This exception will be raised when the format of an assembly conflict with the format of main application at runtime.

Dynamically Load Assemblies

You can dynamically switch window, visual object, menu, and some other objects defined in different PB assemblies at runtime. There are three new system functions in

PowerBuilder .NET to allow this: **AddAssemblyReference**, **SetAssemblyReference** and **GetAssemblyReference**.

These system functions enable you to add a reference to a PowerBuilder assembly deployed by PowerBuilder .NET at runtime. After adding a reference, all types defined in it can be consumed at runtime. Types in the referenced assembly cannot be directly used to define a variable, a return value, or a parameter. Types are typically used in some system functions as a string parameter where the value is the datatype you want to create.

AddAssemblyReference

Adds new PB assembly files to the reference list of the current application at runtime.

Syntax

Int AddAssemblyReference(string *assemblyFullPath*)

Argument	Description
assemblyFullPath	String. A semicolon-separated list of file names. Specify the full file name with its extension. It can be a relative path if the assembly is in the same directory with the current application. Otherwise, it must be an absolute path.

Returns

`Integer`. Returns 1 if all files in the list are successfully added. If an error occurs, it returns -1. If any argument's value is null, it returns null.

Usage

After the function is invoked successfully at runtime, types in the referenced assembly can be consumed in PB .NET.

Invoking a Dynamically Loading Assembly

```
int ret
ret = addassemblyreference("D:\Dynamic Loading\PB125\pbassembly.out
\bin\release\pbassembly.dll")

if ret = 1 then
    MessageBox("OK", "OK")
elseif ret = -1 then
    MessageBox("Error", "D:\Dynamic Loading\PB125\pbassembly.out\bin
\release\pbassembly.dll Not Found")
else
    MessageBox("Error", "Null argument")
end if
```

SetAssemblyReference

Changes PB assembly files to the reference list of the current application at runtime. The difference with AddAssemblyReference is that all PB assemblies in the reference list will be replaced after invoking the function.

Syntax
Int SetAssemblyReference(string *assemblyFullPath*)

Argument	Description
assemblyFullPath	String. A semicolon-separated list of file names. Specify the full file name with its extension. It can be a relative path if the assembly is in the same directory with the current application. Otherwise, it must be an absolute path.

Returns
`Integer`. Returns 1 if all files in the list are successfully added. If an error occurs, it returns -1. If any argument's value is null, it returns null.

Usage
After the function is invoked successfully at runtime, types in the referenced assembly can be consumed in PB .NET.

GetAssemblyReference

Gets valid PB assembly files from the reference list of the current application at runtime.

Syntax
String GetAssemblyReference()

Returns
`String`. Returns the current referenced assembly list. Multiple assemblies are separated by semicolons. Only valid PB assembly files in the referenced assembly list can be returned.

Working with Dynamically Loaded Assemblies

These are some examples of how to work with dynamically loaded assemblies.

Open a Window

The window “w_test” is defined in a referenced assembly.

```
window w
Open(w, "w_test")
```

Open an Object

The user object “u_cvuo” is defined in a referenced assembly.

```
UserObject u
w_window.OpenUserObject(u, “u_cvuo”)
```

Create a Menu

The menu “m_test” is defined in a referenced assembly.

```
Menu m
m = create using “m_test”
w_window.MenuID = m
```