Deploying Components as .NET Assemblies or Web Services

Appeon PowerBuilder® 2017

FOR WINDOWS

DOCUMENT ID: DC31081-01-1700-01

LAST REVISED: September 04, 2017

Appeon Limited, 1/F, Shell Industrial Building, 12 Lee Chung Street, Chai Wan District, Hong Kong.

Contents

1 Welcome

This help describes how to deploy non-visual objects, like .NET assemblies, and .NET Web services, from PowerBuilder to a .NET environment.

2 .NET Component Targets

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

- **.NET Assembly Targets**

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

- **.NET Web Service Targets**

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

The .NET Web Service and .NET Assembly targets require that the PowerBuilder IDE be run under the administrator mode, because these targets call external executable files that must be run under the administrator mode.

2.1 .NET Assembly Targets

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

You can create .NET Assembly targets from scratch or by using PBLs from an existing target that contain at least one nonvisual custom class object.

Creating a target from scratch

When you use the .NET Assembly target wizard to create a target from scratch, the wizard also creates an Application object, a project object that allows you to deploy the assembly, and a nonvisual object (NVO). However, you must add and implement at least one public method in the wizard-created NVO before it can be used to create a .NET assembly.

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

Table 2.1:

Wizard field	Description
Project name	Name of the project object the wizard creates.
Library	Name of the library file the wizard creates. By default, this includes the current Workspace path and takes the name you enter for the project object with a PBL extension.
Target	Name of the target the wizard creates. By default, this includes the current Workspace path and takes the name you enter for the project object with a PBT extension.
Library search path	Lets you add PBLs and PBDs to the search path for the new target.
PowerBuilder object name	Name of the nonvisual object the wizard creates. By default this takes the name that you entered for a project object with an "n_" prefix.
Description	Lets you add a description for the project object the wizard creates.

Wizard field	Description
Namespace	Provides a globally unique name to assembly elements and attributes, distinguishing them from elements and attributes of the same name but in different assemblies.
Assembly file name	Name of the assembly created by the wizard. By default, the assembly file name takes the namespace name with a DLL suffix.
Resource file and directory list	List of resource files, or directories containing resource files, that you want to deploy with the project. You can use the Add Files, Add Directories, or Search PBR Files buttons to add files and directories to the list box. You can select a file or directory in the list and click the Delete button to remove that file or directory from the list. When you select a directory, the resource files in all of its subdirectories are also selected by default. However, you can use the Resource Files tab in the Project painter to prevent deployment of subdirectory files. For more information, see Resource Files and Library Files tabs.
Win32 dynamic library file list	Specifies any Win32 DLLs you want to include with your project. Click the Add button to open a file selection dialog box and add a DLL to the list. Select a DLL in the list and click Delete to remove the DLL from the list.
Setup file name	Name of the setup file the wizard creates. You can copy this MSI file to client computers, then double-click the files to install the .NET assembly on those computers.

Creating a target from an existing target

If you select the option to use an existing target, the wizard creates only the .NET Assembly target and a .NET Assembly project. The target you select must include a PBL with at least one nonvisual object having at least one public method. The public method must be implemented by the nonvisual object or inherited from a parent. The AutoInstantiate property of the nonvisual object must be set to false.

Note

All objects from an existing target are visible in the System Tree for the .NET Assembly target created from the existing target, except for any project objects that are incompatible with the new target. Although visual objects, as well as the application object, are not used in a .NET Assembly target, you can view them in the System Tree under the new target's PBLs.

When you use the wizard to create a .NET Assembly target from an existing target, the wizard prompts you for the same information as when you create a target from scratch, except that it omits the PowerBuilder object name and library search path fields. These fields are unnecessary because the existing target must have a usable nonvisual object and the library search path for the target is already set. The wizard does, however, present fields that are not available when you create a target from scratch.

This table describes these additional fields:

Table 2.2:

Wizard field	Description
Choose a target	Select a target from the list of targets in the current workspace.
Specify a project name	Select a name for the project you want to create. You must create a project object to deploy nonvisual objects as .NET components.
Choose a project library	Specify a library from the list of target libraries where you want to store the new project object.
Choose NVO objects to be deployed	Expand the library node or nodes in the list box and select check boxes next to the nonvisual objects that you want to deploy.
Use .NET nullable types	Select this check box to map PowerBuilder standard datatypes to .NET nullable datatypes. Nullable datatypes are not Common Type System (CTS) compliant, but they can be used with .NET Generic classes if a component accepts or returns null arguments or if reference arguments are set to null.
Only include functions with supported datatypes	Select this check box if you do not want to list functions that are not supported in the .NET environment. The functions will be listed in the Select Objects dialog box that you can open for the project from the Project painter.

After you create a .NET Assembly target, you can create as many .NET Assembly projects as you need. You start the .NET Assembly project wizard from the Project tab of the New dialog box. The fields in the wizard include all the fields in the table for creating a project from scratch, except for the "PowerBuilder object name" and "Description" fields. They also include all fields in the table for creating a project from an existing target, except for the "Choose a target" field.

Whether you opt to build a new target from scratch or from an existing target, most of the project-related fields listed in these tables are available for modification in the Project painter.

- **Modifying a .NET Assembly Project**

  You can modify a .NET Assembly project from the Project painter.

- **Supported Datatypes**

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

- **Deploying and Running a .NET Assembly Project**

  After you create a .NET Assembly project, you can deploy it from the Project painter or from a context menu on the project object in the System Tree.

2.1.1 Modifying a .NET Assembly Project

You can modify a .NET Assembly project from the Project painter.

In addition to the values for fields that you entered in the target and project wizards, you can also modify version, debug, and run settings from the Project painter, and select and rename functions of the nonvisual objects that you deploy to a .NET assembly.

Each .NET Assembly project has seven tab pages: General, Objects, Resource Files, Library Files, Version, Post-build, and Run.

General tab

The General tab in the Project painter allows you to modify the namespace, assembly file name, and setup file name for a .NET Assembly project. It also has a check box you can select to use .NET nullable datatypes. These fields are described in .NET Assembly Targets.

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

Table 2.3:

Project painter field	Description
Debug or Release	Options that determine whether the project is deployed as a debug build (default selection) or a release build. You use debug builds for debugging purposes. Release builds have better performance, but when you debug a release build, the debugger does not stop at breakpoints.
Enable DEBUG symbol	Option to activate code inside conditional compilation blocks using the DEBUG symbol. This selection does not affect and is not affected by the project's debug build or release build setting. This option is selected by default.

Objects tab

The Objects tab in the Project painter lists all the nonvisual user objects available for deployment from the current .NET Assembly target. The Custom Class field lists all these objects even if you did not select them in the target or project wizard.

Objects that you selected in the wizard display with a user object icon in the Custom Class treeview. All methods for the objects selected in the wizard are also selected for deployment by default, but you can use the Objects tab to prevent deployment of some of these methods and to change the method names in the deployed component.

This table describes the fields available on the Objects tab:

Table 2.4:

Project painter field	Description
Custom class	Select an object in this treeview list to edit its list of functions for inclusion in or exclusion from the assembly component. You can edit the list for all the objects you want to include in the assembly, but you must do this one object at a time.
Object name, Class name, and Namespace	You can change the object name only by selecting a different object in the Custom Class treeview. By default, the class name is the same as the object name, but it is editable. In the Project painter, the namespace is editable only on the General tab.
Method names and Function prototypes	Select the check box for each function of the selected custom class object you want to deploy to a .NET assembly. Clear the check box for each function you do not want to deploy. You can modify the method names in the Method Names column, but you cannot use

Project painter field	Description
	dashes ("-") in the modified names. The Function Prototype column is for descriptive purposes only.
Change method name and description	You enable these buttons by selecting a method in the list of method names. PowerBuilder allows overloaded functions, but each function you deploy in an assembly class must have a unique name. After you click the Change Method Name button, you can edit the selected method name in the Method Name column. The Change Method Description button lets you add or edit a method description.
Select All and Unselect All	Click the Select All button to select all the functions of the current custom class object for deployment. Click the Unselect All button to clear the check boxes of all functions of the current custom class object. Functions with unselected check boxes are not deployed to a .NET assembly.

Resource Files and Library Files tabs

The fields that you can edit on the Resource Files and Library Files tabs of the Project painter are the same as the fields available in the target and project wizards. These fields are described in the first table in .NET Assembly Targets.

The Resource Files page of the Project painter does have an additional field that is not included in the project or target wizard. The additional field is a Recursive check box next to each directory that you add to the Resource Files list. By default, this check box is selected for each directory when you add it to the list, but you can clear the check box to avoid deployment of unnecessary subdirectory files.

Version, Post-build, and Run tabs

The fields on the Version, Post-build, and Run tabs of the Project painter are not available in the .NET Assembly target or project wizards. This table describes these fields:

Table 2.5:

Project painter field	Description
Version tab: Product name, Company, Description, and Copyright	Use these fields to specify identification, description, and copyright information that you want to associate with the assembly you generate for the project.
Version tab: Product version, File version, and Assembly	Enter major, minor, build, and revision version numbers for the product, file, and assembly.
Post-build tab: Post-build command line list for build type	Select the build type (Debug or Release) and click Add to include command lines that run immediately after you deploy the project. For example, you can include a command line to process the generated component in a code obfuscator program, keeping the component safe from reverse engineering. The command lines run in the order listed, from top to bottom. You can save separate sequences of command lines for debug and release build types.

Project painter field	Description
Run tab: Application	You use this text box to enter the name of an application with code that invokes the classes and methods of the generated assembly. If you do not enter an application name, you get an error message when you try to run or debug the deployed project from the PowerBuilder IDE.
Run tab: Argument	You use this text box to enter any parameters for an application that invokes the classes and methods of the deployed project.
Run tab: Start In	You use this text box to enter the starting directory for an application that invokes the classes and methods of the deployed project.

Sign tab

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

2.1.2 Supported Datatypes

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

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

Table 2.6:

PowerScript datatype	C# datatype
boolean	bool
blob	byte [ ]
byte	byte
int, uint	short, ushort
long, ulong	int, uint
longlong	long
decimal	decimal
real	float
double	double
string	string
user-defined structure	struct
user-defined nonvisual object	class
Date	DateTime
Time	DateTime
DateTime	DateTime

Note

Arrays are also supported for all standard datatypes.

2.1.3 Deploying and Running a .NET Assembly Project

After you create a .NET Assembly project, you can deploy it from the Project painter or from a context menu on the project object in the System Tree.

When you deploy a .NET Assembly project, PowerBuilder creates an assembly DLL from the nonvisual user objects you selected in the wizard or project painter. If you also listed a setup file name, PowerBuilder creates an MSI file that includes the assembly DLL and any resource files you listed in the wizard or Project painter.

Note

You can use the Runtime Packager to copy required PowerBuilder runtime files to deployment computers. For information about the Runtime Packager, see Application Techniques > Deploying Applications and Components.

You can run or debug an assembly project from the PowerBuilder UI if you fill in the Application field (and optionally, the Argument and Start In fields) on the project Run tab in the Project painter.

2.2 .NET Web Service Targets

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

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

Creating a target from scratch

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

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

Table 2.7:

Wizard field	Description
Project name	Name of the project object the wizard creates.
Library	Name of the library file the wizard creates. By default, this includes the current Workspace path and takes the name you enter for the project object with a PBL extension.

Wizard field	Description
Target	Name of the target the wizard creates. By default, this includes the current Workspace path and takes the name you enter for the project object with a PBT extension.
Library search path	Lets you add PBLs and PBDs to the search path for the new target.
PowerBuilder object name	Name of the nonvisual object the wizard creates. By default this takes the name that you entered for a project object with an "n_" prefix.
Description	Lets you add a description for the project object the wizard creates.
Web service virtual directory name	The directory path you want to use as the current directory in the virtual file system on the server. By default, this is the full path name for the current PowerBuilder target.
Web service URL preview	Address for accessing the .NET Web service from an application.
Resource file and directory list	List of resource files, or directories containing resource files, that you want to deploy with the project. You can use the Add Files, Add Directories, or Search PBR Files buttons to add files and directories to the list box. You can select a file or directory in the list and click the Delete button to remove that file or directory from the list. When you select a directory, the resource files in all of its subdirectories are also selected by default. However, you can use the Resource Files tab in the Project painter to prevent deployment of subdirectory files. For more information, see "Resource Files and Library Files tabs".
Win32 dynamic library file list	Specifies any Win32 DLLs you want to include with your project. Click the Add button to open a file selection dialog box and add a DLL to the list. Select a DLL in the list and click Delete to remove the DLL from the list.
Generate setup file	Select this option to deploy the Web service in an MSI file. When you select this option, you must provide a name for the setup file.
Setup file name	Name of the setup file the wizard creates. You can copy this MSI file to client computers, then double-click the files to install the .NET Web service on those computers.
Directly deploy to IIS	Select this option to deploy the Web service directly to an IIS server. When you select this option, you must provide an IIS server address. By default, the server address is "localhost".

When you click Finish in the wizard for a target you are creating from scratch, the wizard generates an Application object, a project object, a target, and a nonvisual object. You must add and implement a public method in the nonvisual object generated by the wizard before you can deploy it as a Web service.

Creating a target from an existing target

As with the other .NET target wizards, you can use the .NET Web Service target wizard to create a target from an existing PowerBuilder target. The existing target must be added to

the current workspace and must include a PBL with at least one nonvisual object having at least one public method. The public method must be implemented by the nonvisual object or inherited from a parent. The AutoInstantiate property of the nonvisual object must be set to false.

When you click Finish in the wizard for a target you are creating from an existing target, the wizard creates a .NET Web Service target and a .NET Web Service project. The .NET Web Service target uses the same library list as the existing target from which you select nonvisual user objects.

As with the .NET Assembly target wizard, the .NET Web Service target wizard has additional fields for selecting nonvisual user objects when you use the existing target option. This table describes these additional fields:

Table 2.8:

Wizard field	Description
Choose a target	Select a target from the list of targets in the current workspace.
Specify a project name	Select a name for the project you want to create. You must create a project object to deploy nonvisual objects as .NET components.
Choose a project library	Specify a library from the list of target libraries where you want to store the new project object.
Choose NVO objects to be deployed	Expand the library node or nodes in the list box and select check boxes next to the nonvisual objects that you want to deploy.
Use .NET nullable types	Select this check box to map PowerBuilder standard datatypes to .NET nullable datatypes. Nullable datatypes are not Common Type System (CTS) compliant, but they can be used with .NET Generic classes if a component accepts or returns null arguments or if reference arguments are set to null.
Only include functions with supported datatypes	Select this check box if you do not want to list functions that are not supported in the .NET environment. After you create the .NET project, the functions are listed on the Objects tab for the project when you open it in the Project painter.
Only include functions with supported datatypes	Select this check box if you do not want to list functions that are not supported in the .NET environment. The functions will be listed in the Select Objects dialog box that you can open for the project from the Project painter.

- **Modifying a .NET Web Service Project**

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

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

  Configure .NET Web Service projects.

- **Global Web Configuration Properties**

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

- **Deploying and Running a .NET Web Service Project**

  After you create a .NET Web Service project, you can deploy it from the Project painter or from a context menu on the project object in the System Tree.

- **.NET Web Service Deployment Considerations**

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

2.2.1 Modifying a .NET Web Service Project

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

The Project painter shows all the values you selected in .NET Web Service target or project wizards. However, you can also modify version, debug, and run settings from the Project painter, and select and rename functions of the nonvisual objects that you deploy to a .NET Web Service component.

.NET Web Service project tab pages

Each .NET Web Service project has these tab pages:

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

  Table 2.9:

Project painter field	Description
Debug or Release	Options that determine whether the project is deployed as a debug build (default selection) or a release build. You use debug builds for debugging purposes. Release builds have better performance, but when you debug a release build, the debugger does not stop at breakpoints.
Enable DEBUG symbol	Option to activate code inside conditional compilation blocks using the DEBUG symbol. This selection does not affect and is not affected by the project's debug build or release build setting. This option is selected by default.

- Deploy tab - the fields on the Deploy tab are all available in the .NET Web Service project wizard. For descriptions of fields available on the Deploy tab, see the first table in .NET Assembly Targets.

- Objects tab - allows you to select the methods to make available for each nonvisual object you deploy as a Web service. You can rename the methods as Web service messages. This table describes the Objects tab fields for a .NET Web Service project:

  Table 2.10:

Objects tab field	Description
Custom class	Select an object in this treeview list to edit its list of methods for inclusion in or exclusion from the Web service component. You can edit the list for all the objects you want to include in the component, but you must do this for one object at a time.

Objects tab field	Description
Object name	You can change the object name only by selecting a different object in the Custom Class treeview.
Web service name	Specifies the name for the Web service. By default, this takes the name of the current custom class user object.
Target namespace	Specifies the target namespace. The default namespace for an IIS Web service is: http://tempurl.org. Typically you change this to a company domain name.
Web service URL	Specifies the deployment location for the current custom class user object. This is a read-only field. The location combines selections on the General, Deploy, and Objects tabs for the current project.
Web service WSDL	Specifies the WSDL file created for the project. This is a read-only field. It appends the WSDL suffix to the Web service URL.
Browse Web Service	If you have previously deployed the project to the named IIS server on the Deploy tab of the current project, you can click this button to display a test page for the existing Web service. If a Web service has not been deployed yet for the current custom class object, a browser error message displays. The button is disabled if you selected the option to deploy the current project to a setup file.
View WSDL	If you previously deployed the project to the named IIS server on the Deploy tab of the current project, you can click this button to display the existing WSDL file. If a Web service has not been deployed yet for the current custom class object, a browser error message displays. The button is disabled if you selected the option to deploy the current project to a setup file.
Message names and Function prototypes	Select the check box for each function of the selected custom class object that you want to deploy in a .NET Web service component. Clear the check box for each function you do not want to deploy. You can modify the message names in the Message Names column. The Function Prototype column is for descriptive purposes only.
Change message name	You enable this button by selecting a function in the list of message names. PowerBuilder allows overloaded functions, but each function you deploy in a component class must have a unique name. After you click the Change Message Name button, you can edit the selected function name in the Message Name column.
Select All and Unselect All	Click the Select All button to select all the functions of the current custom class object for deployment. Click the Unselect All button to clear the check boxes of all functions of the current custom class object. Functions with unselected check boxes are not deployed as messages for a Web service component.

- Resource Files tab - the fields on this tab are the same as those in the project wizard. However, as for the .NET Assembly project, there is one additional field that is not included in the project or target wizard. This field is a Recursive check box next to each

directory you add to the Resource Files list. By default, this check box is selected for each directory when you add it to the list, but you can clear the check box to avoid deployment of unnecessary subdirectory files.

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

Table 2.11:

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

- Post-build tab - the items on this tab cannot be set in the target or project wizards. Select a build type and click Add to include command lines that run immediately after you deploy the project. For example, you can include a command line to process the generated component in a code obfuscator program, keeping the component safe from reverse engineering. The command lines run in the order listed, from top to bottom. You can save separate sequences of command lines for debug and release build types.
- Security tab - on this tab, configure CAS security zones for Web Service components, minimizing the amount of trust required before component code is run from a user application. A radio button group field on the Security tab allows you to select full trust (default) or a customized trust option. The list box below the radio button group is disabled when full trust is selected, but it allows you to select or display the permissions you want to include or exclude when the custom option is selected.

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

Table 2.12:

Run tab field	Description
Application	Use this text box to enter the name of an application with code that invokes the classes and methods of the generated assembly. If you do not enter an application name, you get an error message when you try to run or debug the deployed project from the PowerBuilder IDE.
Argument	Use this text box to enter any parameters for an application that invokes the classes and methods of the deployed project.

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

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

2.2.2 Configuring ASP.NET for a .NET Web Service Project

Configure .NET Web Service projects.

IIS and ASP.NET

ASP.NET configuration includes making sure the Web server has a compatible version of IIS and that the 4.x version of ASP.NET is selected for your Web service components.

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

SQL Anywhere database connections

Set up a database connection for your Web service components. See Setting Up a SQL Anywhere Database Connection.

Global properties

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

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

- PBMaxSession
- PBEventLogID
- PBDeleteTempFileInterval

See Global Web Configuration Properties.

2.2.3 Global Web Configuration Properties

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

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

This table lists global properties that you can set for .NET Web Service targets:

Table 2.13:

Property	Default value	Description
LogFolder	`WebAppDir..` \`appName`_root\log	Folder that contains the PBTrace.log file.
FileFolder	`WebAppDir..` \`appName`_root\file	Base directory for the virtual file manager. It contains the File\Common directory structure and files that mirror paths for the application resource files on the development computer. If you switch to Copy mode, a `sessionID` directory is created under the File\Session directory that mirrors the File \Common directory structure and file contents.
PrintFolder	`WebAppDir..` \`appName`_root\print	Base directory for files that your application prints in PDF format.
PBWebFileProcessMode	Share	Share mode maintains files in a read-only state when a write file operation is not explicitly coded. If an application requires multiple file operations, you might want to change this property setting to Copy mode.
PBCurrentDir	c:\	Specifies the current directory for the Web Service.

Property	Default value	Description
PBTempDir	c:\temp	A temporary directory under the virtual file root on the server.
PBLibDir	c:\~pl_	The directory on the server where dynamic libraries are generated.
PBDenyDownloadFolders	c:\~pl_	A semicolon-delimited string of directory names.
PBTrace	Enabled	Indicates whether to log exceptions thrown by the application. Values are Enabled or Disabled.
PBTraceTarget	File	Defines where to log exceptions thrown by the application. Values are File or EventLog.
PBTraceFileName	PBTrace.log	Name of the file that logs exceptions thrown by the application. By default, this file is saved to the `applicationName`_root \Log directory under the virtual root directory on the server.
PBEventLogID	1100	The event ID if exceptions are logged to the EventLog.
PBDeleteTempFileInterval	600 (minutes)	Sets the number of minutes before temporary files created by composite DataWindows are deleted. A value of 0 prevents the temporary files from being deleted.

2.2.4 Deploying and Running a .NET Web Service Project

After you create a .NET Web Service project, you can deploy it from the Project painter or from a context menu on the project object in the System Tree.

When you deploy directly to an IIS server, PowerBuilder creates an application directory under the IIS virtual root and creates an ASMX file in the application directory. The ASMX file created by the project is an ASP.NET executable file rather than a true WSDL file, so you might need to add the WSDL suffix to the URL when you try to access this Web service from certain types of applications.

In addition to the application directory and the ASMX file, deploying the project creates an additional assembly containing the Web service wrapper class. The file name for this

assembly is generated by appending the characters "ws" to the file name of the main application assembly. It is generated with the main assembly in the application's bin directory.

Note

In some versions of IIS, ASPNET Web services use the Temp system directory during method processing. If the IIS_IUSRS user group (IIS 7.5 or later) does not have read or write access to the Temp directory on the server, applications invoking methods on those services receive an error message stating that temporary classes cannot be generated. You can prevent this error by granting appropriate user or user group permissions to the Temp directory in the same way you grant permissions for the Appeon and database directories. See Setting Up a SQL Anywhere Database Connection.

When you deploy to a setup file in a .NET Web Service project, the project builds an MSI file that includes the ASMX file, PowerBuilder system libraries for .NET, and any resource files you listed in the project wizard or painter.

Note

You can use the Runtime Packager to copy required PowerBuilder runtime files to deployment servers. After you install the package created by the runtime packager, you must restart the server. For information about the Runtime Packager, see Application Techniques > Deploying Applications and Components.

You can run or debug a .NET Web Service project from the PowerBuilder UI if you fill in the Application field (and optionally, the Argument and Start In fields) on the project Run tab in the Project painter. The Application field is typically filled in automatically with the name of the Internet Explorer executable on the development computer.

2.2.5 .NET Web Service Deployment Considerations

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

When a .NET Web Service project is open in the Project painter and no other painters are open, you can select Design > Deploy Project from the Project painter to deploy the project.

When all painters are closed, including the Project painter, you can right-click a Web Service project in the System Tree and select Deploy from its context menu.

The Output window shows the progress of the deployment and provides a list of application functions, events, and properties that are not supported in the Web Service version of the application. Most of these warnings are benign and do not prevent users from running the application as a Web Service.

If a supported version of the Microsoft .NET Framework is the only version of the .NET Framework installed on the server, or if you configured the server to use a supported version (4.x) for all Web sites by default, you can run the application immediately after you deploy it.

You can run the application from PowerBuilder by selecting Design > Run Project from the Project painter menu or selecting the Run Project toolbar icon from the Project painter

toolbar. The System Tree context menu for the Web Service project also has a Run Project menu item.

2.2.5.1 Deployment to a setup file

If you are deploying a .NET project to an MSI file, you must have a file named License.rtf in the PowerBuilder DotNET\bin directory. The PowerBuilder setup program installs a dummy License.rtf file in this directory, but you should modify this file's contents or replace the file with another file of the same name.

The License.rtf file should contain any license information you want to distribute with your application. You can run the .NET Assembly or Web Service only after the setup file is extracted to an IIS server. The contents of the License.rtf file appear in the setup file extraction wizard.

After you create and distribute the MSI file to an IIS server, you must extract the MSI file on the server. By default the extraction directory is set to C:\Program Files \webservice\`applicationName`, and the extraction wizard creates the C:\Program Files\webservice\`applicationName`\`applicationName` and C:\Program Files \webservice\`applicationName`\`applicationName`_root virtual directories, where `applicationName` is the name of your application.

Although you do not need to modify the default extraction directory to run the application, the extraction wizard does let you change the location of the application directories you extract. If you prefer to keep all your applications directly under the server's virtual root, you could set the extraction directory to server's Inetpub\wwwroot directory.

2.2.5.2 Deployment to a production server

You can deploy a .NET Assembly or Web Service to a production server either by:

- Extracting an MSI file that you build from a Web Service project
- Deploying directly from the development computer to a mapped server
- Copying all application folders and files from IIS on a local server to IIS on a production server

Production servers must meet the requirements described in ASP.NET Configuration for a .NET Project. You must install all database clients and have access to all data sources on the production computer. For applications that you deploy to a production server, you should add required database driver DLLs to the Win32 dynamic library list on the Library Files tab page of your Web Service projects. If you are using ODBC to connect to a database, you should add the PBODB170.INI file to the list of resource files on the Resource Files tab page of Web Service projects.

The production server must have the following DLLs in its system path: atl80.dll, msvcr80.dll, msvcp80.dll, msvcp100.dll, msvcr100.dll, pbshr170.dll, and if your application uses DataWindow objects, pbdwm170.dll. You can also use the Runtime Packager to deploy required PowerBuilder runtime files to the ASP.NET server. After you install the package created by the Runtime Packager, you must restart the server.

For a complete list of required runtime files and for information on the Runtime Packager, see Application Techniques > Deploying Applications and Components.

2.2.5.3 Deployment to a remote server

You can deploy directly to a mapped server only if the server is in the same domain or workgroup as the development computer. In addition, you must add the development computer user's Windows login ID as a member of the Administrators group on the remote computer hosting the IIS server.

If you copy a Web Service from a development computer to a production server, you must copy both the `applicationName` and `applicationName`_root folders (and their contents) that were created when you deployed the application locally. Direct deployment to a mapped server automatically adds the necessary ASP.NET user permissions to access these directories, but if you copy files to the server, you must add these permissions manually.

2.2.5.4 ASP .NET user permissions

If you copy files to a production server, or extract your Web Service from an MSI file, you can use Windows Explorer to grant ASP.NET permissions to the application directories. This method is described in Setting Up a SQL Anywhere Database Connection. You can also grant ASP.NET permissions from a command line. The commands are different depending on the version of IIS that your server is running:

Table 2.14:

IIS version	Commands for granting appropriate user permissions
7.5 and later	`cacls applicationName\temp /t /e /c /g IIS_IUSRS:f` `cacls applicationName_root /t /e /c /g IIS_IUSRS:f`

2.2.5.5 Event logging on the production server

If you log Web Service events to a production server's event log (by setting the PBTraceTarget global property to "EventLog"), you must have a registry entry key for PBExceptionTrace. If you use an MSI file to deploy an application to a production server, the PBExceptionTrace key is created automatically. If you deploy directly to a mapped production server or if you copy a Web Service to a production server, you must import the PBExceptionTrace key or create it manually.

When you deploy to a local computer, PowerBuilder creates the following key: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application \PBExceptionTrace. You can export this key to a .REG file and import it to the production server's registry.

For information on the PBTraceTarget global property, see Global Web Configuration Properties.

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

3 Deploying .NET Component Targets

PowerBuilder not only supports traditional client-server applications, it also provides you with the ability to create and deploy .NET component targets with relative ease.

- **How .NET Deployment Works**

  When you deploy a .NET project, PowerBuilder compiles existing or newly developed PowerScript code into .NET assemblies.

- **Security Settings**

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

- **Strong-Named Assemblies**

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

- **ASP.NET Configuration for a .NET Project**

  You can configure ASP.NET for a .NET project before or after you deploy the project to an IIS 7.5 or later server.

- **Checklist for Deployment**

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

3.1 How .NET Deployment Works

When you deploy a .NET project, PowerBuilder compiles existing or newly developed PowerScript code into .NET assemblies.

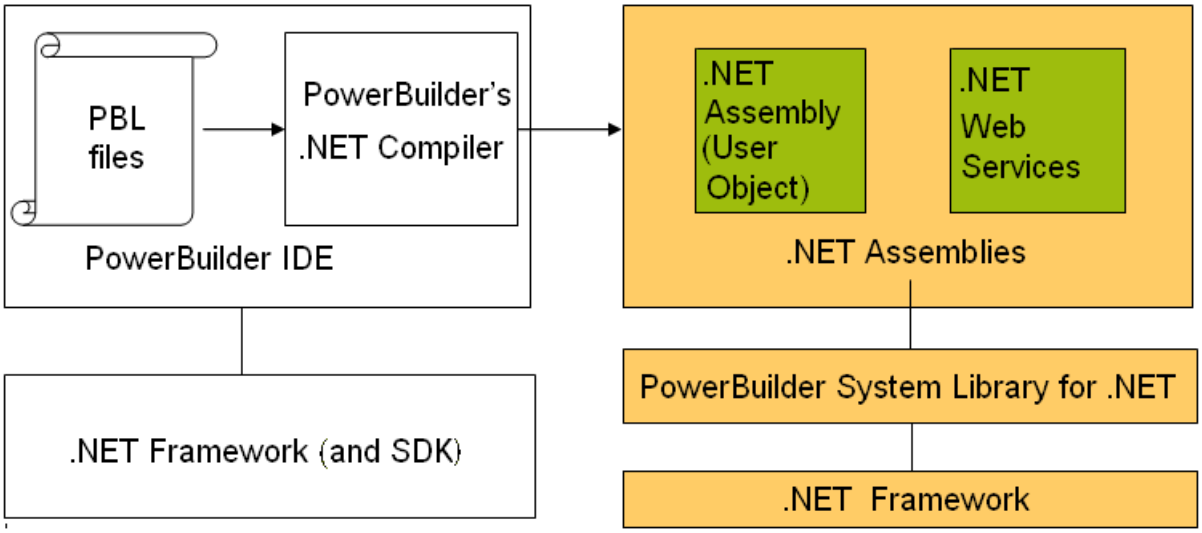
At runtime, the generated .NET assemblies execute using the .NET Common Language Runtime (CLR). PowerBuilder's .NET compiler technology is as transparent as the P-code compiler in standard PowerBuilder client-server applications.

If you generate assemblies from a component target type, the assemblies are deployed as independent .NET components or as Web services.

Note

For .NET Assemblies and Web Services, you must install the .NET Framework redistributable package on the deployment computer or server. The SDK and the redistributable package are available as separate downloads from the Microsoft .NET Framework Developer Center at http://msdn.microsoft.com/en-us/netframework/aa731542.aspx.

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



3.2 Security Settings

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

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

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

3.2.1 Permission error messages

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

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

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

All .NET targets must include a reference to the mscorlib.dll .NET Framework assembly in order to catch a System.Security.SecurityException exception. PowerBuilder .NET Web

Service components generate PBTrace.log files that log critical security exceptions by default.

If you do not catch the exception when it is thrown, the PowerScript SystemError event is triggered.

3.2.2 Debugging and tracing with specified security settings

You can debug and run .NET Assembly or Web Service from the PowerBuilder IDE with specified security settings. (The CAS settings generated in the Web.config file determine the security permissions used by .NET Web Service components.)

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

3.3 Strong-Named Assemblies

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

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

3.3.1 Project painter Sign tab

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

Table 3.1:

Assembly group box field	Description
Sign the assembly	Select this check box to enable the "Choose a strong name key file" single line edit box, the browse and New buttons, and the "Delay sign only" check box.
Choose a strong name key file	Name of the key file you want to attach to the generated assembly. This field is associated with a browse (ellipsis) button and a New button. The browse button opens a Select File dialog box where you can select a key file with the .snk extension. The New button lets you create a key file with the .snk extension. PowerBuilder uses the Sn.exe tool from the .NET Framework to create the key file.
Delay sign only	Select this check box if your company's security considerations require the step of signing the assembly to be separate from

Assembly group box field	Description
	the development process. When this check box is selected, the project will not run and cannot be debugged. However, you can use the strong name tool Sn.exe (in the .NET Framework) with the -Vr option to skip verification during development.
Mark the assembly with AllowPartiallyTrustedCallerAttribute (.NET Web Service and .NET Assembly projects only)	By default, a strong-named assembly does not allow its use by partially trusted code and can be called only by other assemblies that are granted full trust. However, you can select this check box to allow a strong-named assembly to be called by partially trusted code.

3.3.2 Error messages

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

3.4 ASP.NET Configuration for a .NET Project

You can configure ASP.NET for a Web service project before or after you deploy the project to an IIS 7.5 or later server.

All files and directories that you access from a Web service application on a Web server must have appropriate IIS_IUSRS (IIS 7.5 or later) user permissions.

Note

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

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

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

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

- **IIS Installation**
- **Selecting the Default ASP.NET Version**

  If you installed multiple versions of the .NET Framework on the target Web server, you should make sure that IIS uses a supported version for the .NET assembilies or Web services.
- **Setting Up a SQL Anywhere Database Connection**

  Full control permissions are required for directories containing databases that you need to access from your .NET Web services.
- **Configuration Requirements for Windows 7 and Later**

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

3.4.1 IIS Installation

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

If IIS 7.5 or later is installed after the .NET Framework, you must register IIS with ASP.NET manually or reinstall the .NET Framework. To manually register IIS with ASP.NET, go to the .NET Framework path, run aspnet_regiis.exe -i in the command line console, and restart IIS.

3.4.2 Selecting the Default ASP.NET Version

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

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

In IIS 7.5 and later, set the .NET Framework version for the application pool your applications use. For more information, see Configuration Requirements for Windows 7 and Later.

1. Select Start > Run from the Windows Start menu.
2. Type InetMgr in the Run dialog box list.
3. In the left pane of the IIS Manager, expand the local computer node and its Web Sites sub-node.
4. One of the following:
   - For all new Web sites, right-click the Default Web Site node and select Properties.
   - For already deployed projects, expand the Web site node and right-click the .NET Assembly or Web Service that you deployed from PowerBuilder.

5. Specify the ASP.NET or .NET Framework version.
   - On Windows 7 and later, set the .NET Framework version used by your NVO Web service deployment:
     a. In the IIS Manager, open the Application Pools node underneath the machine node.
     b. Right-click the PBDotNet4AppPool filter and choose Advanced Settings.
     c. Set the .NET Framework Version to 4.x.

3.4.3 Setting Up a SQL Anywhere Database Connection

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

Before the .NET Web Service connects to a SQL Anywhere database, you must either start the database manually or grant the IIS_IUSRS (IIS 7.5 or later) default permissions for the Appeon\Shared and SAP SQL Anywhere directories, making sure to replace permissions of all child objects in those directories.

Note

If your database configuration uses a server name, you must provide the database server name in the start-up options when you start the database manually, in addition to the name of the database file you are accessing.

If you do not grant the appropriate user permissions for Appeon directories and your database configuration is set to start the database automatically, your application will fail to connect to the database. SQL Anywhere cannot access files unless the IIS_IUSRS user group has the right to access them.

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

Note

To show the Security tab of the Select Users, Computers, or Groups dialog box, you might need to modify a setting on the View tab of the Folder Options dialog box for your current directory. You open the Folder Options dialog box by selecting the Tools > Folder Options menu item from Windows Explorer. To display the Security tab, you must clear the check box labeled "Use simple file sharing (Recommended)"

---

3. Click Locations, choose the server computer name from the Locations dialog box, and click OK.
4. Type IIS_IUSRS (IIS 7.5 or later) in the list box labeled "Enter the object names to select" and click OK.

If valid for your server, the account name you entered is added to the Security tab for the current directory. You can check the validity of a group or user name by clicking Check Names before you click OK.

5. Select the new account in the top list box on the Security tab, then select the check boxes for the access permissions you need under the Allow column in the bottom list box.

   You must select the Full Control check box for a directory containing a database that you connect to from your application.

6. Click Advanced.

7. Select the check box labeled "Replace permission entries on all child objects with entries shown here that apply to child objects" and click OK.

   A Security dialog box appears, and warns you that it will remove current permissions on child objects and propagate inheritable permissions to those objects, and prompts you to respond.

8. Click Yes at the Security dialog box prompt, then click OK to close the Properties dialog box for the current directory.

   The pbtrace.log file is created in the `applicationName`_root directory. This file records all runtime exceptions thrown by the application and can be used to troubleshoot the application.

3.4.4 Configuration Requirements for Windows 7 and Later

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

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

Table 3.2:

Settings for	Required changes
Windows firewall	Enable exceptions for WMI and file and printer sharing
UAC (When you are not running PowerBuilder with the built-in Administrator account)	If the development and deployment computers are in the same domain, connect to the remote computer using a domain account that is in its local Administrators group. Then UAC access token filtering does not affect the domain accounts in the local Administrators group. You should not use a local, nondomain account on the remote computer because of UAC filtering, even if the account is in the Administrators group. If the development and deployment computers are in the same workgroup, UAC filtering affects the connection to the remote computer even if the account is in the Administrators group. The only exception is the native "Administrator" account of the remote computer, but you should not use this account because of security issues. Instead, you can turn off UAC on

Settings for	Required changes
	the remote computer. and if the account you use has no remote DCOM access rights, you must explicitly grant those rights to the account.
DCOM	Grant remote DCOM access, activation, and launch rights to a nondomain user account in the local Administrators group of the remote computer if that is the type of account you are using to connect to the remote computer.

3.5 Checklist for Deployment

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

3.5.1 Checklist for all .NET targets

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

- The Windows 7 or later operating system (including Windows 7/8.1/10 and Windows Server 2008/2012/2016)
- .NET Framework 4.x
- The Microsoft Visual C++ runtime libraries msvcr80.dll, msvcp80.dll, msvcp100.dll, msvcr100.dll, and the Microsoft .NET Active Template Library (ATL) module, atl80.dll
- PowerBuilder .NET assemblies in the global assembly cache (GAC)
- PowerBuilder runtime dynamic link libraries in the system path

3.5.2 Checklist for .NET Web Service targets

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

- IIS 7.5 or later (See IIS Installation)
- ASP.NET (See Selecting the Default ASP.NET Version)
- ASP.NET permissions for all files and directories used by your applications

  For an example of how to grant ASP.NET permissions, see Setting Up a SQL Anywhere Database Connection.

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

3.5.3 Installing assemblies in the global assembly cache

When the Common Language Runtime (CLR) is installed on a computer as part of the .NET Framework, a machine-wide code cache called the global assembly cache (GAC) is created. The GAC stores assemblies that can be shared by multiple applications. If you do not want

or need to share an assembly, you can keep it private and place it in the same directory as the application.

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

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

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

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

4 Appendix

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

- **Custom Permission Settings**

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

4.1 Custom Permission Settings

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

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

- **Adding Permissions in the .NET Framework Configuration Tool**

  The list of permissions that display in the Security tab permissions list box is the same as the list in the "Everything" permission set of the .NET Framework 4.x SDK Configuration tool runtime security policy.

- **EnvironmentPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **EventLogPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **FileDialogPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **FileIOPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **PrintingPerission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **ReflectionPermission**

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

- **RegistryPermission**

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

- **SecurityPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **SMTPPermission**

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

- **SocketPermission**

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

- **SQLClientPermission**

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

- **UIPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **WebPermission** (obsolete)

  This setting is obsolete, as .NET Windows Forms application is no longer supported since PowerBuilder 2017.

- **Custom Permission Types**

  Permission types that you can customize on the Security tab page of the Project painter (besides the permissions described elsewhere in this appendix) have no direct impact on PowerScript functions or properties in .NET targets.

4.1.1 Adding Permissions in the .NET Framework Configuration Tool

The list of permissions that display in the Security tab permissions list box is the same as the list in the "Everything" permission set of the .NET Framework 4.x SDK Configuration tool runtime security policy.

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

1. Close PowerBuilder if it is open, and create an XML file with the permission settings you want to add. For example, by default, the SMTPPermission setting is not included in the assigned permissions in the "Everything" permission set. To create this permission, save a file named SMTPPermission.xml with the following content:

```
<IPermission class=
   "System.Net.Mail.SmtpPermission, System,
   Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089"
```

```
   version="1" Unrestricted="true"/>
```

2. Open the .NET Framework SDK Configuration tool from the Administrative Tools folder in your computer Control Panel.

3. In the left pane of the configuration tool, select My Computer > Runtime Security Policy > Machine > Permission Sets > Everything, then select the Action > Change Permissions menu item.

4. In the Create Permission Set dialog box, click Import to open the Import a Permission dialog box, browse to the SMTPPermission.xml file, and click OK.

5. Click Finish, close the configuration tool, and open a .NET project in PowerBuilder to the Security tab page. The SMTPPermission displays in the list box of the Security tab page. You can scroll the list to see it when you select any radio button option other than Full Trust.

4.1.2 ReflectionPermission

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

Table 4.1: ReflectionPermission required in .NET targets

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

Example 1

This permission setting allows reflection for members of a type that are not visible:

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

4.1.3 RegistryPermission

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

Table 4.2: Required RegistryPermission settings for system functions

System function	Permission required
RegistryGet, RegistryKeys, RegistryValues	Read
RegistrySet	Write; if registry key does not exist, requires Create
RegistryDelete	Read and Write

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

Table 4.3: Required RegistryPermission settings for MLSync functions

MLSync function	Permission required
GetObjectRevisionFromRegistry, GetsSyncRegistryProperties	Read on HKEY_CURRENT_USER registry key
GetDBMLSyncPath	Read on the Software\SAP\SQL Anywhere registry keys under HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE
SetsSyncRegistryProperties	Unrestricted on HKEY_CURRENT_USER registry key

Example 1

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

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

4.1.4 SMTPPermission

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

Table 4.4: SMTPPermission required in .NET targets

MailSession object function	Permission required
MailLogon	Connect (if using default port) or ConnectToUnrestrictedPort

Example 1

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

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

4.1.5 SocketPermission

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

The SocketPermission class belongs to the System.Net namespace described on the Microsoft Web site at http://msdn.microsoft.com/en-us/library/system.net.aspx.

Table 4.5: SocketPermission required in .NET targets

Connection object function	Permission required
ConnectToServer	Connect

Example 1

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

```
<IPermission class="System.Net.SocketPermission,
   System, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" version="1">
   <ConnectAccess>
      <ENDPOINT host="10.42.144.40" transport="Tcp"
      port="2000"/>
   </ConnectAccess>
   </IPermission>
```

4.1.6 SQLClientPermission

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

Table 4.6: SQLClientPermission required in .NET targets

Feature	Permission required
Database connect	Unrestricted

Example 1

This permission setting allows database connections:

```
<IPermission class=
   "System.Data.SqlClient.SqlClientPermission,
   System.Data, Version=4.0.0.0, Culture=neutral,
   PublicKeyToken=b77a5c561934e089" version="1"
   Unrestricted="true" />
```

4.1.7 Custom Permission Types

Permission types that you can customize on the Security tab page of the Project painter (besides the permissions described elsewhere in this appendix) have no direct impact on PowerScript functions or properties in .NET targets.

However, if you use the language interoperation feature of PowerBuilder, this may also require customized permissions for the following permission types:

- ASPNETHostingPermission
- ConfigurationPermission
- DataProtectionPermission
- DNSPermission
- IsolatedStoragePermission
- KeyContainerPermission
- OleDBPermission
- PerformanceCounterPermission

- StorePermission