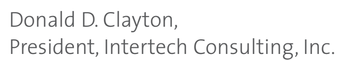
SYBASE®

WHITE PAPER

# Moving Your PowerBuilder® Application to the Web



Donald D. Clayton,
President, Intertech Consulting, Inc.

**TABLE OF CONTENTS**


Introduction

1 Background

- 1 PowerBuilder's .NET strategy
- 4 PowerBuilder .NET Windows Forms
  - 5 Utilize local resources
  - 5 Distributed
  - 5 Offline Capable
  - 6 Intelligent installation and update
  - 6 Client device flexibility
  - 6 PowerBuilder Smart Clients and the .NET Framework
  - 7 Smart Client security
  - 9 PowerBuilder Smart Client version control
- 11 PowerBuilder .NET Web Forms
  - 11 Compiled code
  - 11 Server controls
  - 11 Browser Independence
  - 12 Separation of code and content
  - 12 State management
  - 12 .NET Web Forms architecture
  - 13 Web Forms event model
- 14 PowerBuilder Web DataWindow
- 14 CITRIX or Terminal Services
- 15 Appeon

16 Major considerations

- 16 How much of the application needs to go to the internet?
- 16 What Is the context of the technical environment?
- 16 What are the client requirements?
- 17 Is the application a good fit for the internet?


## INTRODUCTION

Today's Internet-connected world virtually requires that all companies leverage the power of the Internet. There is a wide and sometimes bewildering array of techniques available to PowerBuilder programmers to move all or part of their PowerBuilder logic to the Internet. A process-based approach to examining which aspects of the application should be moved to the Internet, combined with an intelligent refactoring of the application and an understanding of your company's technology stack will maximize your organization's PowerBuilder investment.

Options for PowerBuilder customers moving the all or part of an application to the web include using PowerBuilder .NET Windows Forms, PowerBuilder .NET Web Forms, the PowerBuilder Web DataWindow, CITRIX or Terminal Services, and Appeon. It is also possible to move selected processes or tasks to a web-based role after doing some analysis. This paper will present decision-frameworks for determining how much of the application should be moved to the Internet, as well as a present the tradeoffs between techniques, to help the audience decide which technique will prove most beneficial.

### BACKGROUND

The following sections provide a short review of each technique before moving on to discuss the strengths and weakness of various approaches.

### POWERBUILDER'S .NET STRATEGY

With PowerBuilder 11, PowerBuilder developers became .NET developers. PowerBuilder 11.x allows developers to deploy PowerBuilder applications as a Windows Forms (Windows Forms) application, and optionally use the .NET Smart Client infrastructure to manage and coordinate installations and updates. PowerBuilder can also compile an NVO as a .NET web service or a .NET class. PowerBuilder hides many details of .NET so that PowerBuilder developers can use existing knowledge and skill set to develop .NET applications. More experienced .NET developers can develop many interesting applications using .NET interoperability features.

The PowerBuilder 11.x .NET strategy has unfolded over the last few PowerBuilder releases. .NET is a relatively broad term. Microsoft .NET includes an interpretive managed code runtime environment (somewhat like PCode, but with JIT compilation technologies included), garbage collection, thread-level security, code versioning and signing, various language compilers, a new deployment paradigm using "assemblies" that are packaged and placed in a machine's local or "Global Assembly Cache", (also called the GAC), and a low-level class library called the .NET Framework.

The .NET Framework can be likened to the Java Developer's Kit, or JDK, in that it includes both visual and non-visual classes that can be extended by developers in any .NET language. Figure 1 shows the major packages, called assemblies, in the System Package of the .NET Framework:

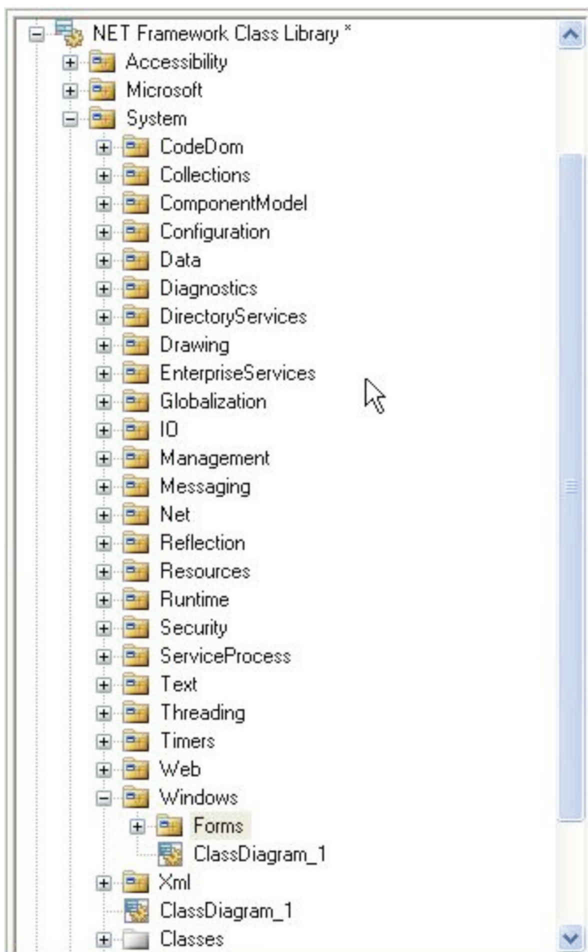



**Figure 1.** Class Hierarchy courtesy of PowerDesigner 12.5

The first task the PowerBuilder engineers faced when developing for .NET was to determine the classes that the PowerBuilder system classes needed to inherit from in the .NET framework. Each PowerBuilder system class was carefully extended from its .NET counterpart to maintain backwards compatibility with earlier versions of PowerBuilder. The major ancestral .NET visual classes reside within the System.Windows.Forms package. Figure 2 shows a partial list of the visual classes found:

List of Classes

| | Name | Code | |
|---|---|---|---|
| → | AccessibleEvents | AccessibleEvents | Packa |
| 2 | AccessibleNavigation | AccessibleNavigation | Packa |
| 3 | AccessibleObject | AccessibleObject | Packa |
| 4 | AccessibleRole | AccessibleRole | Packa |
| 5 | AccessibleSelection | AccessibleSelection | Packa |
| 6 | AccessibleStates | AccessibleStates | Packa |
| 7 | AmbientProperties | AmbientProperties | Packa |
| 8 | AnchorStyles | AnchorStyles | Packa |
| 9 | Appearance | Appearance | Packa |
| 10 | Application | Application | Packa |
| 11 | ApplicationContext | ApplicationContext | Packa |
| 12 | ArrangeDirection | ArrangeDirection | Packa |
| 13 | ArrangeStartingPosition | ArrangeStartingPosition | Packa |
| 14 | AxHost | AxHost | Packa |
| 15 | BaseCollection | BaseCollection | Packa |
| 16 | Binding | Binding | Packa |
| 17 | BindingContext | BindingContext | Packa |
| 18 | BindingManagerBase | BindingManagerBase | Packa |
| 19 | BindingMemberInfo | BindingMemberInfo | Packa |
| 20 | BindingsCollection | BindingsCollection | Packa |
| 21 | BootMode | BootMode | Packa |
| 22 | Border3DSide | Border3DSide | Packa |
| 23 | Border3DStyle | Border3DStyle | Packa |
| 24 | BorderStyle | BorderStyle | Packa |
| 25 | BoundsSpecified | BoundsSpecified | Packa |
| 26 | Button | Button | Packa |
| 27 | ButtonBase | ButtonBase | Packa |
| 28 | ButtonBorderStyle | ButtonBorderStyle | Packa |
| 29 | ButtonState | ButtonState | Packa |
| 30 | CaptionButton | CaptionButton | Packa |
| 31 | CharacterCasing | CharacterCasing | Packa |
| 32 | CheckBox | CheckBox | Packa |
| 33 | CheckedListBox | CheckedListBox | Packa |
| 34 | CheckState | CheckState | Packa |
| 35 | Clipboard | Clipboard | Packa |
| 36 | ColorDepth | ColorDepth | Packa |
| 37 | ColorDialog | ColorDialog | Packa |
| 38 | ColumnClickEventArgs | ColumnClickEventArgs | Packa |
| 39 | ColumnHeader | ColumnHeader | Packa |
| 40 | ColumnHeaderStyle | ColumnHeaderStyle | Packa |
| 41 | ComboBox | ComboBox | Packa |
| 42 | ComboBoxStyle | ComboBoxStyle | Packa |
| 43 | CommonDialog | CommonDialog | Packa |

OK Cancel Apply Help

**Figure 2:** Visual Classes within the System.Windows.Forms package.

Of interest here is the process of generating the PowerBuilder target as .NET. When deploying a PowerBuilder .NET targets, PowerBuilder actually generates C# code using something called the PowerBuilder C# Emitter, a process similar to performing a PowerBuilder machine code build. This C# code is immediately compiled with Microsoft's C# compiler and then thrown away. The resulting managed code assemblies and other files are then organized for deployment to the .NET runtime. Thus developers are in the PowerBuilder IDE but creating .NET runtimes, as shown in Figure 3.

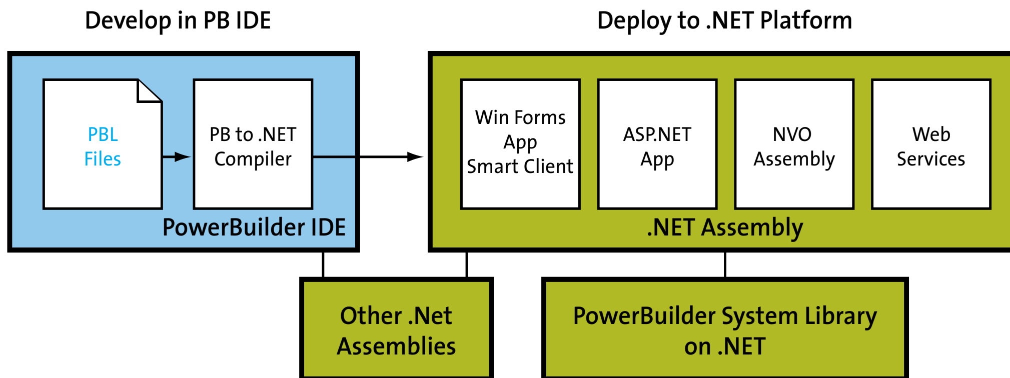



**Figure 3:** Courtesy of William Allison and Lin Cao, Sybase Inc

As shown in Figure 3 above, PowerBuilder .NET assemblies are developed by mapping the PowerBuilder system classes to a set of .NET ancestors and generating and compiling C#. A more detailed look at how the PowerBuilder .NET architecture was accomplished is shown in Figure 4:

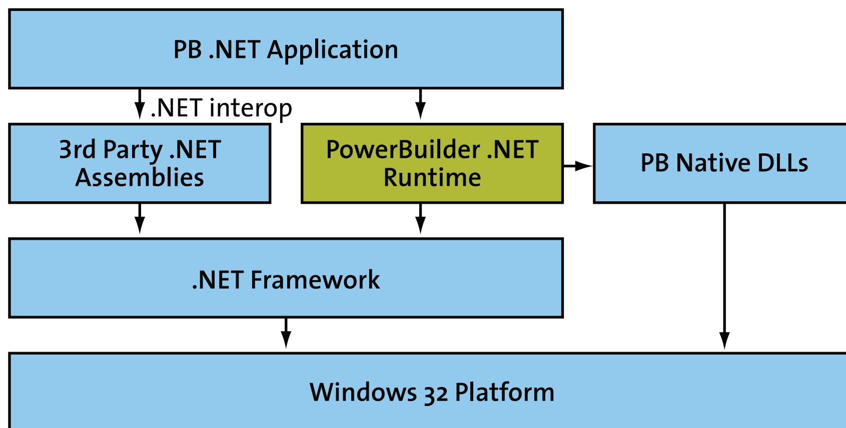



**Figure 4:** The PowerBuilder .NET Windows Form architecture, courtesy of Xue-song Wu, Sybase, Inc.

In order to call third party .NET assemblies or use a .NET class, the assembly that contains the .NET class must be referenced by the target. This effectively increases the Target's .NET Namespace. This is done using the Target properties dialog:

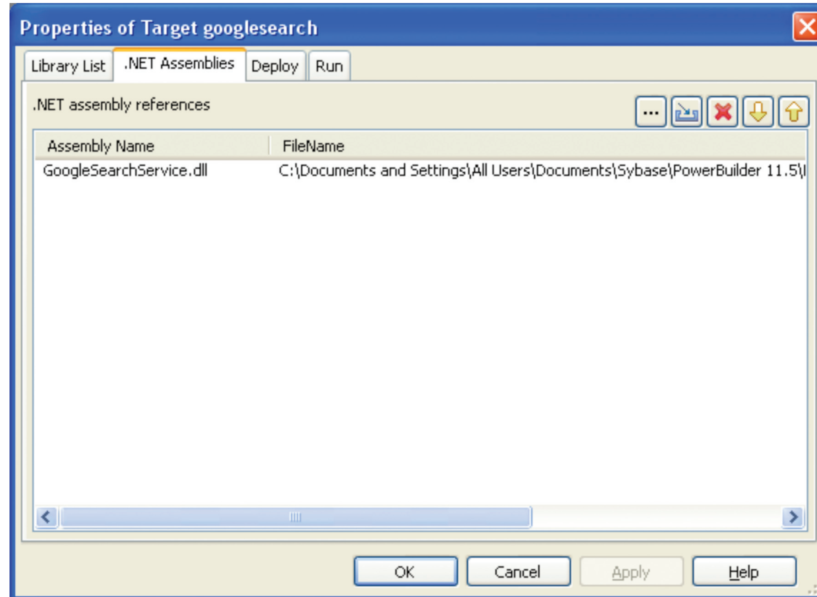



**Figure 4A:** Expanding the default namespace for a PowerBuilder 11.5 Target

The result of this approach is that the managed code runs natively in the .NET runtime, has access to the .NET system classes in the .NET framework, the ability to call other .NET assemblies, and the ability to call legacy unmanaged DLLs and OCXs using techniques built into .NET. The net effect of this transition is that PowerBuilder has positioned itself as the highest-level, most productive .NET language, simultaneously harnessing the power and control available in .NET.

### POWERBUILDER .NET WINDOWS FORMS

PowerBuilder applications that have a rich user interface that relies on resources available on the client computer, such as a complex MDI design, graphics, or animations, or that perform intensive data entry or require a rapid response time, make good candidates for deployment as .NET Windows Forms applications. Figure 5 shows the architecture of a .NET Windows Forms application:

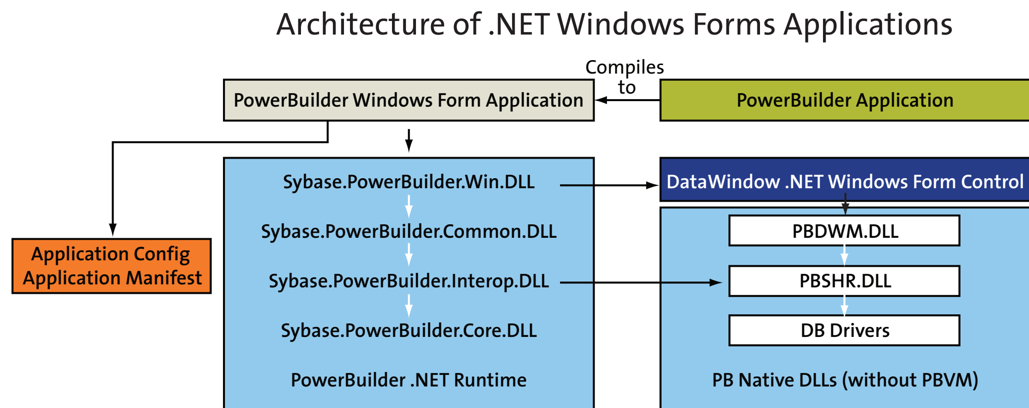



**Figure 5:** Courtesy of Xue-song Wu, Sybase, Inc.

At runtime, the Sybase.PowerBuilder managed code assemblies, collectively known as the Sybase PowerBuilder .NET Runtime, are used to run the managed code generated by the C# code generation process mentioned previously. These managed code assemblies then call two legacy PowerBuilder .DLL files, shown on the right in Figure 5, which in turn invoke the PowerBuilder database drivers and the unmanaged drivers.

.NET Windows Forms are practically indistinguishable from their Win32 brethren. For example, Figure 6 shows the PowerBuilder Sample Application running as a .NET Windows Form. Only minor differences, primarily due to the new ancestral .NET classes can be distinguished by the user.

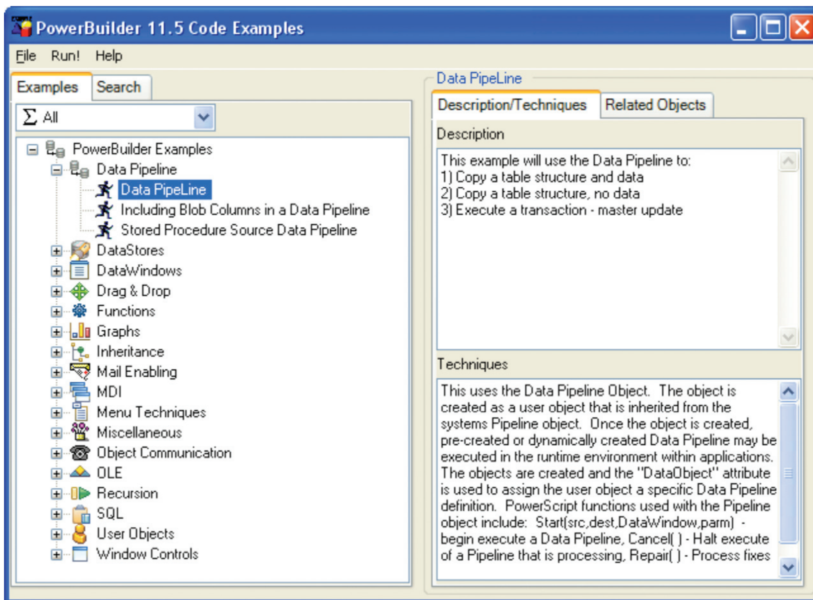



**Figure 6:** The PowerBuilder Sample Application running as a .NET Windows Form over the Internet.

The PowerBuilder Windows Form technology can also make use of another feature built into .NET, the .NET Smart Client Architecture. The term Smart Client highlights the differences between the typical "Rich Client" applications of the client/server era and the next generation of rich Internet applications. To understand these differences, and to understand how they are likely to change the face of client-side computing, it is useful to look at a checklist of what constitutes a Smart Client. These characteristics highlight the features typically provided by Smart Clients over and above those provided by traditional rich client applications. If a client application displays these characteristics, then it can be said to be smart:

### Utilize local resources

A Smart Client application has code artifacts on the client. A Smart Client may take advantage of the local CPU or GPU, local memory or disk, local devices connected to the client such as a telephone, bar-code/RFID reader, and so on. It may also take advantage of local software applications that interact with it.

### Distributed

Smart Client applications form part of a larger distributed solution. The application interacts with various Web Services that provide access to information. Often, the application has access to specific services that help maintain the application and provide deployment and update services.

### Offline capable

Because they are running on the local machine, one of the key benefits that Smart Client applications offer is that they can be made to work even when the user is not connected. When the client is connected, the Smart Client application can improve performance and usability by caching data and managing the connection in an intelligent way.

### Intelligent installation and update

The .NET framework enables application assemblies to be deployed using a variety of techniques, including simple file copy or download over HTTP. Applications can be updated while running and can be deployed on demand by clicking on a URL. The .NET Framework provides a powerful security mechanism that guarantees the integrity of the application and its related assemblies. Assemblies can be given limited permissions in order to restrict their functionality in semi-trusted scenarios, using something called .NET Code Access Security (CAS).

### Client device flexibility

The .NET Framework provides a common platform upon which Smart Client applications can be built. Often, there will be multiple versions of the Smart Client application, targeting different device types (think about PowerBuilder and PocketBuilder sharing business logic, or hybrid PowerBuilder Windows Forms/Web Forms solutions) with each aspect of the system taking advantage of the devices' unique features and providing functionality appropriate to its use.

### PowerBuilder Smart Clients and the .NET Framework

The .NET Framework solved the problem of version conflicts between assemblies shared by more than one application. A .NET application or assembly has a strong coupling to the assemblies and components they depend on, because assemblies have meta-data that specifies both their exact version and the versions of all dependent assemblies. Multiple versions of the same assembly can be installed side-by-side so that all applications can be run with the exact same version of the assembly that they were built and tested against.

With a PowerBuilder Windows Forms application, it is not necessary to deploy the application to the user's desktop. If desired, a user can invoke the application simply by entering or clicking on a URL. The application will download to the client machine, run in a secure execution environment, and can optionally remove itself upon completion. Deployment of Smart Clients is easier because assemblies do not need to be registered on the local machine and can simply be copied to the machine before they are run. This simplifies the deployment of the application.

The install on demand deployment feature in .NET called No-Touch Deployment, allows applications to be run from a URL, FTP site, or any Windows shortcut. When the user clicks on the URL or link, the runtime automatically downloads the application and all related assemblies to a special download cache. This mechanism also checks for version updates so that the user never has to worry about whether they are running the latest version of the application or not. Figure 7 show the PowerBuilder Project Painter options for Smart Client .NET Windows Forms deployment:

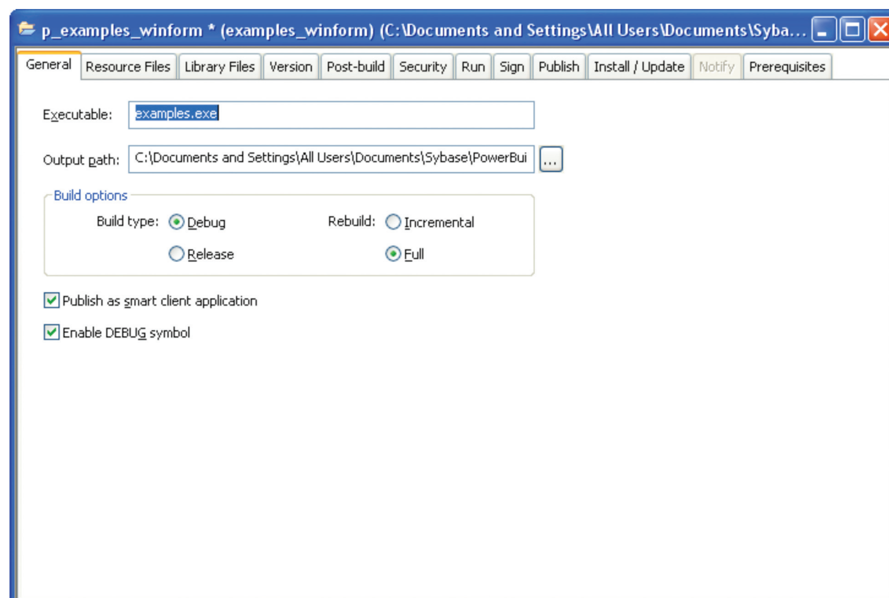



**Figure 7:** Opting to publish a PowerBuilder .NET Windows Forms Application as a Smart Client

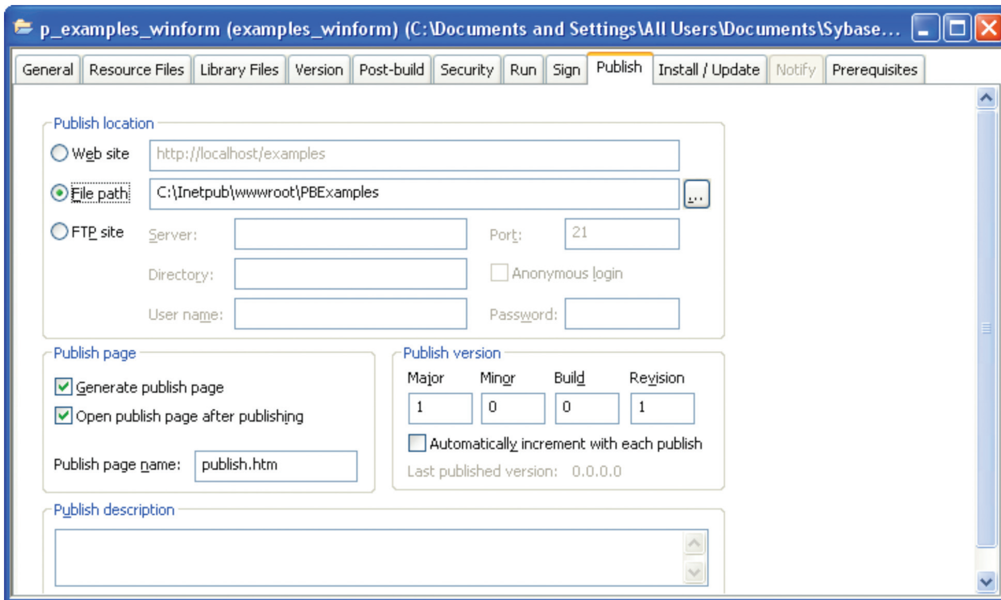



**Figure 8:** Setting publish options for a PowerBuilder .NET Windows Forms Application

.NET provides a number of key features that make the development of PowerBuilder Smart Client applications easier, and the architectural implications of these new features can be exploited by the PowerBuilder developer. It turns out that .NET goes beyond solving the problems that plagued rich clients and enables a whole host of new possibilities.

The flexible binding, deployment, and security models provided by PowerBuilder .NET Windows Forms allow Smart Client applications to be designed in much more interesting ways than traditional rich clients. For example, PowerBuilder .NET Windows Forms provide flexibility in how the application can be hosted: applications can be run as a traditional desktop application or can be hosted from Internet Explorer. Many combinations are possible. For instance, a Windows Forms application can host Internet Explorer components and any host can subsume any other.

### Smart Client security

Applications deployed as Smart Clients are granted a very limited number of permissions by default and are unable to access the local hard disk or access any network services, apart from the service from where they were downloaded. Additional permissions can be granted at the user, machine, or enterprise level if required.

The assembly binding mechanism provides important security safeguards. .NET Code Access Security (CAS) allows assemblies to be granted specific permissions so that they can be used in semi-trusted scenarios. The .NET runtime ensures that assemblies can only carry out operations for which they have been granted permissions. The .NET Framework defines a flexible infrastructure for determining the appropriate permissions to grant an application or assembly.

PowerBuilder applications and components can run in partial trust environments when they are constrained by .NET code access security (CAS) configurations. PowerBuilder lets the developer configure CAS security zones (sandboxes) for .NET Web Forms, .NET Web Service, .NET Windows Forms and Smart Client projects to minimize the amount of trust required before application or component code is run by an end user.

CAS is defined by the .NET System.Security.Permissions namespace. For Smart Client applications, the permission information is stored in the manifest file deployed with the application. When users run a Smart Client application, the application loader process loads the manifest file and creates a sandbox where the application is hosted. Figure 8 shows PowerBuilder .NET Windows Forms CAS options:

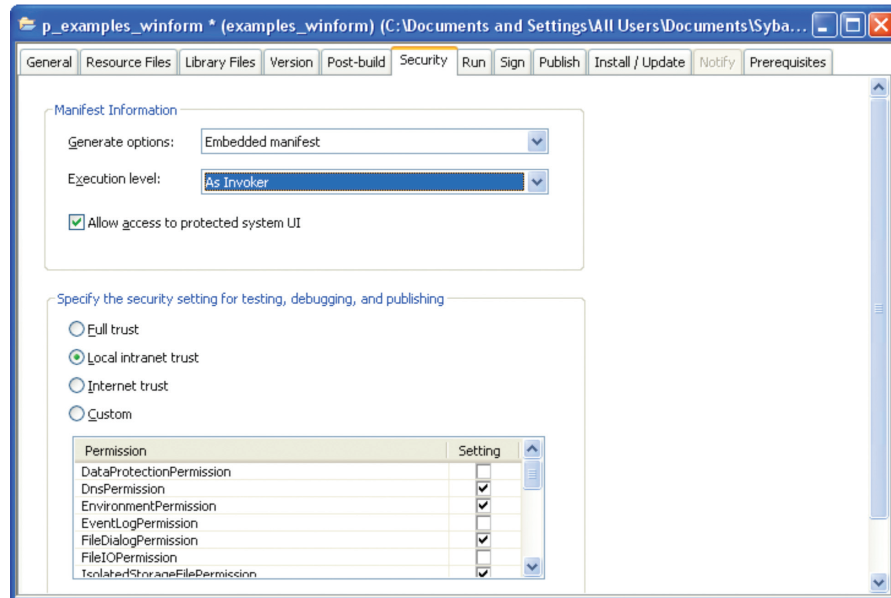



**Figure 9:** PowerBuilder .NET Windows Forms CAS options

Strong-named assemblies can be cryptographically signed to prevent malicious code being run inadvertently, either through tampering or through luring attacks. PowerBuilder can generate strong-named assemblies from all .NET Project painters. A strong name consists of an assembly's identity, including its simple text name, version number, and culture information (when provided)—plus a public key and digital signature. The assembly file contains the assembly manifest that includes the names and hashes of all the files that make up the assembly. Figure 9 shows PowerBuilder .NET Windows Forms code signing options:

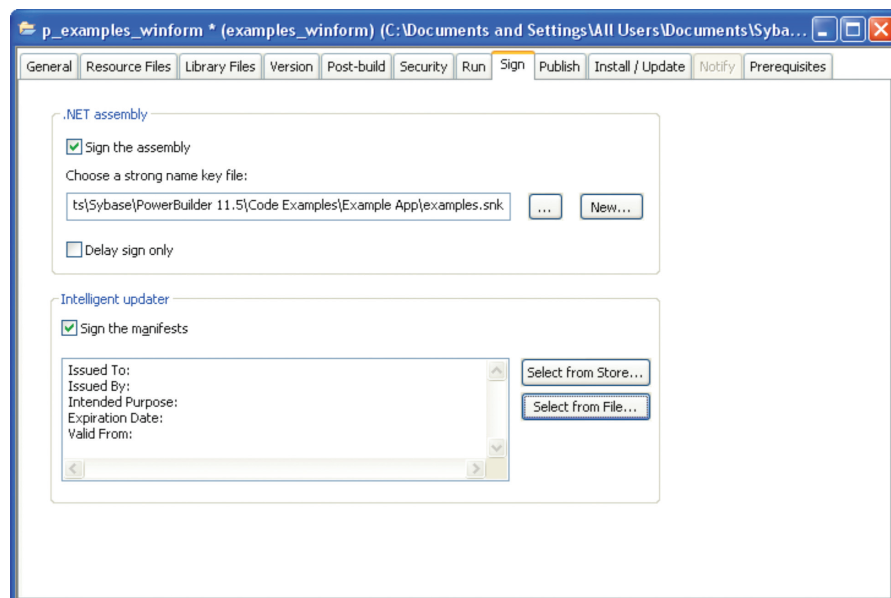



**Figure 10:** PowerBuilder .NET Windows Forms code signing options

### PowerBuilder Smart Client version control

PowerBuilder .NET Windows Forms applications allow for on-the-fly updates. Administrators can use hosts such as web servers, FTP sites, and shared UNC path folders to deploy updates. Because code is deployed in a version-specific manner, multiple versions of a component of application can co-exist, and you can enforce which versions to use. Version enforcement is a feature of the .NET Common Language Runtime, or CLR.

Application logic that is volatile, for example, business rules governing operations, can be factored into assemblies that are downloaded on demand over HTTP, obviating the need to frequently "update" the client application in the traditional sense. Additional (or infrequently used) application features can adopt the same model so that initial application size is kept to a minimum with additional features installed on an as-needed basis. Figure 11 shows the versioning options available for PowerBuilder .NET Windows Forms:

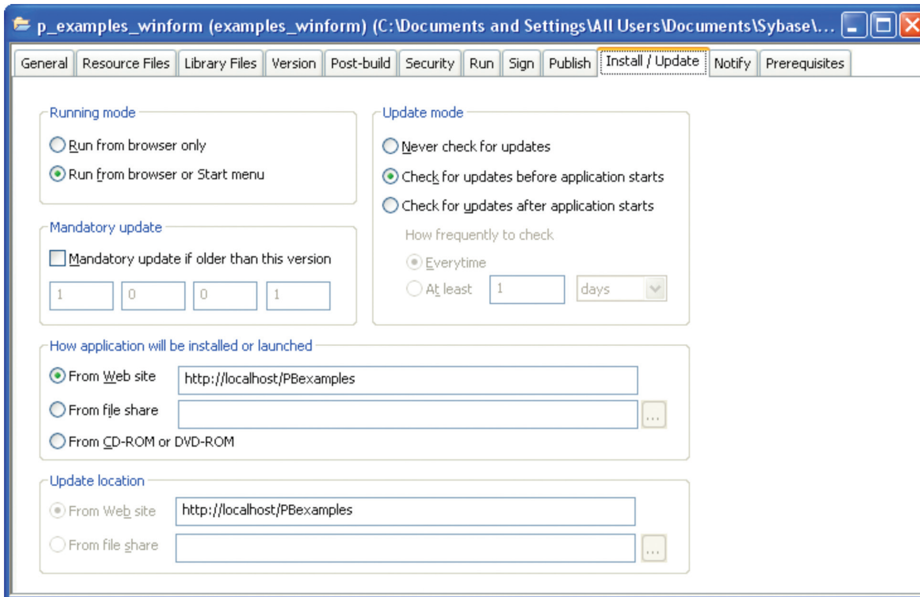



**Figure 11:** The versioning options available for PowerBuilder .NET Windows Forms

The .NET framework includes a common version notifier that is leveraged by PowerBuilder .NET Windows Forms. The PowerBuilder .NET Windows Forms client registers with the notifier agent and the notifier checks the version of the application against the manifest in the location set in Figure 11 above. The impact of the versioned deployment of assemblies is that you can push both updates and rollbacks to your PowerBuilder application without visiting your clients. Figure 12 shows the options available for controlling the updating and rollback of the official version of the application:

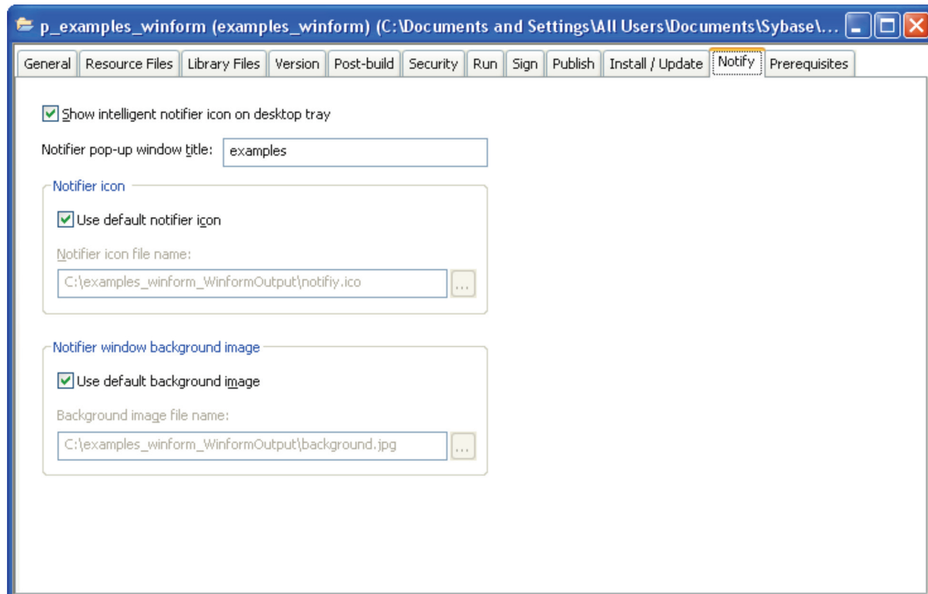



**Figure 12:** The options available for controlling the updating and rolling back of the application.

The techniques presented above, in conjunction with the ability to bundle either managed or unmanaged database drivers in the application's assembly cache mean that you can effectively deploy client/server applications in an extranet environment. The only prerequisites for this technique is that the .NET Framework and the PowerBuilder .NET Framework be installed. Figure 13 show the default generated web page for a PowerBuilder .NET Windows Forms application. Of course, this page can be modified as desired to achieve the desired look and feel and functionality of the organization.

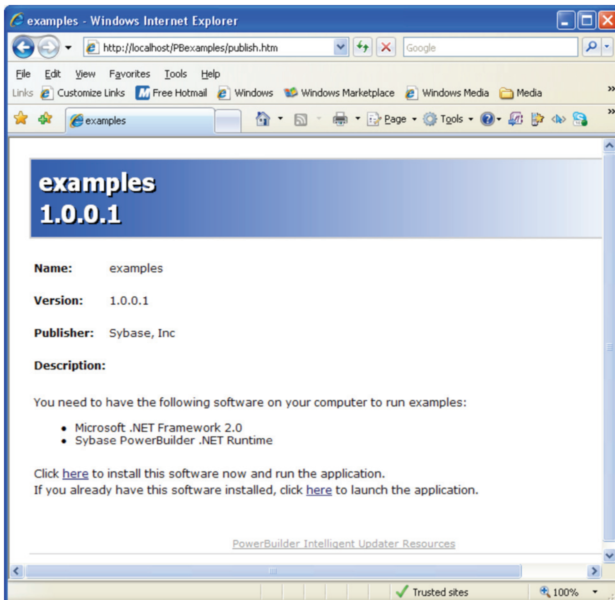



**Figure 13:** The default generated web page for a Smart Client-published PowerBuilder .NET Windows Forms application

### POWERBUILDER .NET WEB FORMS

PowerBuilder .NET Web Forms are yet another way that Sybase has leveraged the power of .NET. Generally speaking, .NET Web Forms are part of the .NET application architecture and are available to other .NET languages as well. Since PowerBuilder 11.5 is a .NET language now, the features of .NET Web Forms are available to PowerBuilder developers. PowerBuilder .NET Windows Forms are developed in conjunction with the .NET DataWindow,® which is available separately for Visual Studio developers who desire the functionality and productivity of the DataWindow.

Web Forms applications have several advantages over traditional client-server and Windows Forms applications. Web Forms applications do not require client-side installation, are easy to upgrade, have no distribution costs, and offer broad-based user access. Any user with a Web browser and an online connection can run Web Forms applications.

An important part of PowerBuilder .NET Web Forms is the ability to create Web applications through a technology known as ASP.NET. ASP.NET provides a programming model and infrastructure that facilitates developing web applications. Part of this infrastructure is the .NET runtime and framework. Server-side code is written in PowerBuilder for ASP .NET consumption and deployed to IIS. Two main programming models are supported by ASP.NET:

- Web Forms help developers build form-based Web pages. Using PowerBuilder as a WYSIWYG development environment enables you to place controls on PowerBuilder windows to be deployed as web pages. Special server-side controls provide an event model similar to what is provided by controls in ordinary PowerBuilder programming.
- Web Services make it possible for a Web site to expose PowerBuilder functionality via the web services APIs and protocols so that PowerBuilder logic can be called remotely by other applications. Data is exchanged using standard Web protocols and formats such as HTTP and XML, which will tunnel through firewalls since web services are typically deployed to listen for requests on port 80.

Both PowerBuilder .NET Web Forms and Web services can take advantage of the facilities provided by .NET, such as the compiling your code and accessing the .NET runtime features. In addition, ASP.NET provides a number of infrastructure services, including state management, security, configuration, caching, and tracing.

### Compiled code

.NET Web Forms (and Web services) can be written in any .NET language that runs in the CLR, including PowerBuilder. Since the code is compiled, this offers better performance than HTML pages with code written in an interpreted scripting language. All the benefits of compilation, such as a managed execution environment are available and of course the entire .NET Framework class library is available. Legacy unmanaged code can be called through the .NET interoperability services of PowerBuilder.

### Server controls

When PowerBuilder generates a .NET Web Forms application, it generates what are known as ASP.NET server-side controls. Server-side code interacts with these controls, and the ASP.NET runtime generates straight HTML that is sent to the Web browser. The result is a PowerBuilder programming model that is as easy to use as PowerBuilder itself yet produces standard HTML that can run in any browser.

### Browser independence

Although the World Wide Web is built on standards, unfortunately browsers are not compatible and many have special features. A typical Web page designer has the option of either writing to the lowest common denominator of all browsers or writing special code for different browsers. .NET Server controls remove some of this pain. PowerBuilder and the ASP.NET 2.0 AJAX Extensions, in conjunction with the Telerik Web RAD controls (some of which ship with PowerBuilder 11.5) take care of browser compatibility issues when it generates code for a server control. If the requesting browser is upscale, the generated HTML can take advantage of these features, otherwise the generated code will be vanilla HTML. In the v11.x generation, Web Form-deployed PowerBuilder applications are supported to function only within the Internet Explorer browser.

### Separation of code and content

Typical ASP pages have a mixture of scripting code interspersed with HTML elements. In web applications there should be a clean separation of code and presentation content. Of course, this implies that the PowerBuilder developer adhered to some type of Model-View-Controller (MVC) pattern. While modern coding practices in object-oriented programming languages practically dictate the use of MVC, the unfortunate truth is that much PowerBuilder code was developed without the MVC pattern in mind and thus there is frequently not a clean separation of code and content in legacy PowerBuilder applications. PowerBuilder Web Forms applications can help with some of the partitioning of logic as far as the physical tier is concerned, but will not partition your application's code for you.

### State management

HTTP is a stateless protocol. If a user enters information in various controls on a form and sends the filled-out form to the server, the information will be lost if the form is displayed again unless the web server preserves the state. PowerBuilder and ASP.NET make this kind of state preservation transparent. There are also convenient facilities for managing other types of state, including session and application state.

### .NET Web Forms architecture

A Web Form consists of two parts, the visual content or presentation, typically specified by HTML elements, and code that contains the logic for interacting with the visual elements. A Web Form is physically expressed by a file with the extension .aspx. Any HTML page could be renamed to have this extension and could be accessed using the new extension with identical results to the original. Web Forms are thus upwardly compatible with HTML pages and can be used in conjunction with Web Forms. Figure 14 shows the PowerBuilder 11.5 Code Examples running as a .NET Web Form.

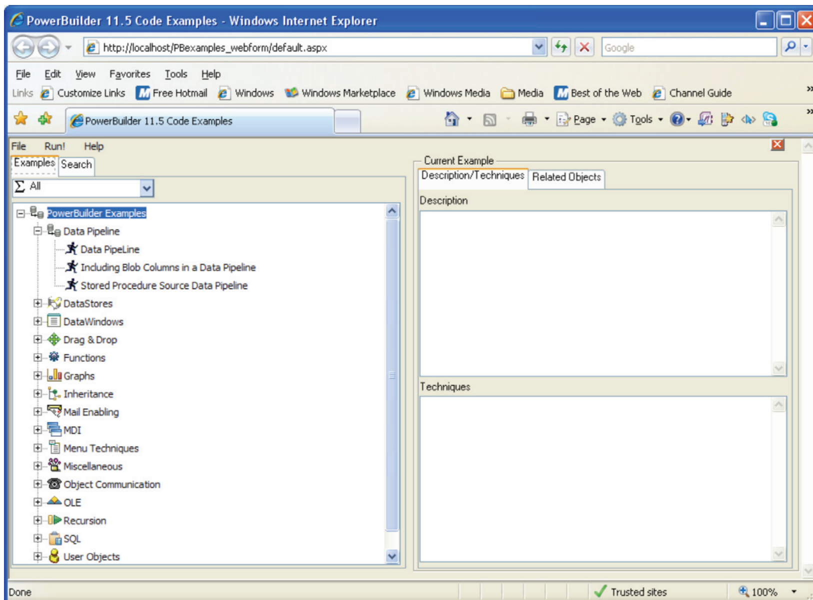



**Figure 14:** The PowerBuilder 11.5 Code Example application running as a .NET Web Form

### Web Forms event model

From the standpoint of the programmer, the event model for Web Forms is very similar to the event model for Windows Forms. This similarity is what makes programming for the web with PowerBuilder so easy. What happens in the case of Web Forms is that some events get raised on the client and some of these events get processed on the server.

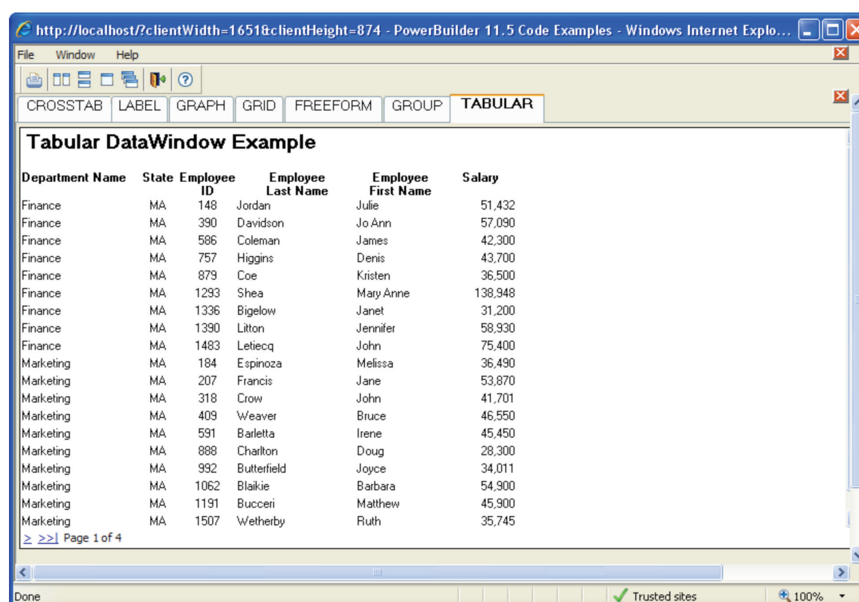
Round trips to the web server are typically expensive; so many PowerBuilder events do not automatically cause a postback to the server. Many web controls have what is known as an intrinsic set of events that automatically cause a postback to the web server. The most common example of an intrinsic event is a button click. Other events such as selecting an item in a list box do not cause an immediate postback to the web server. These events are cached until a button click causes a post to the server. The server then processes the events, in no particular order, and the button-click event that caused the post is then processed.

This is an important consideration for PowerBuilder Web Forms deployment. Care must be taken to ensure that the Web Forms application does not make excessive postbacks to the web server. This can be accomplished through design and by leveraging the PowerBuilder Web DataWindow client-side event model, where events such as ItemChanged are coded in JavaScript and executed on the client. Unfortunately, the Web DataWindow client-side event model and the use of JavaScript with the Web DataWindow are beyond the scope of this article.

## POWERBUILDER WEB DATAWINDOW

The PowerBuilder Web DataWindow is the heart of the PowerBuilder .NET Web Forms architecture and is available in non .NET environments as well. A good example is the use of the PowerBuilder Web DataWindow with .JSP pages, pure HTML pages, and the like. Although there is no pushbutton deployment of a PowerBuilder application to a JSP site (JSP web page target support was dropped in PowerBuilder 11.5), it is not difficult to construct a .JSP web site the connects to PowerBuilder components running in any of the major J2EE application servers supported by the PowerBuilder Application Server Plug-in.

The PowerBuilder Application Server Plug-In version 1.1 for Windows-based JEE containers includes support for PowerBuilder 11.5, is optimized for JDK 1.5, and runs on BEA WebLogic 9.2 and 10.1, IBM WebSphere 6.1, JBoss 4.0.4, and is built into EAServer. For .NET developers, the approach would be to deploy PowerBuilder NVOs and DataWindows as .NET assemblies and code the .ASPX pages to call the .NET DataWindow. Figure 15 shows a PowerBuilder Web DataWindow running in a .NET environment:




| Department Name | State | Employee ID | Employee Last Name | Employee First Name | Salary |
|---|---|---|---|---|---|
| Finance | MA | 148 | Jordan | Julie | 51,432 |
| Finance | MA | 390 | Davidson | Jo Ann | 57,090 |
| Finance | MA | 586 | Coleman | James | 42,300 |
| Finance | MA | 757 | Higgins | Denis | 43,700 |
| Finance | MA | 879 | Coe | Kristen | 36,500 |
| Finance | MA | 1293 | Shea | Mary Anne | 138,948 |
| Finance | MA | 1336 | Bigelow | Janet | 31,200 |
| Finance | MA | 1390 | Litton | Jennifer | 58,930 |
| Finance | MA | 1483 | Letiecq | John | 75,400 |
| Marketing | MA | 184 | Espinoza | Melissa | 36,490 |
| Marketing | MA | 207 | Francis | Jane | 53,870 |
| Marketing | MA | 318 | Crow | John | 41,701 |
| Marketing | MA | 409 | Weaver | Bruce | 46,550 |
| Marketing | MA | 591 | Barletta | Irene | 45,450 |
| Marketing | MA | 888 | Charlton | Doug | 28,300 |
| Marketing | MA | 992 | Butterfield | Joyce | 34,011 |
| Marketing | MA | 1062 | Blaikie | Barbara | 54,900 |
| Marketing | MA | 1191 | Bucceri | Matthew | 45,900 |
| Marketing | MA | 1507 | Wetherby | Ruth | 35,745 |

**Figure 15:** A PowerBuilder Web DataWindow running in a .NET environment

## CITRIX OR TERMINAL SERVICES

The use of CITRIX or Terminal Services are essentially comparable technologies for providing remote presentation of the user interface of PowerBuilder applications while abstracting local machine resources such as file systems and peripherals, and either product is equally viable for the distribution of PowerBuilder applications over the Internet with a few product feature-based exceptions. Terminal emulation solutions accomplish their work by projecting an "image" of the Windows desktop application running on the server to the client plug-in software, which needs to be downloaded and installed to the client computer.

The most notable exceptions are that CITRIX solutions happen to have more advanced deployment features, but require specialized (CITRIX) specific software packages to be installed on the client. This is sometimes a non-starter for organizations that require a zero-install client. The major advantage to Terminal Services solutions is that the Terminal Services Client (TSC) is more widely deployed as it is part of Windows XP and Vista. Thus, administrators can usually safely assume that the users will either already have the TSC installed if running Windows XP Professional or higher.

Generally speaking, there are few limitations placed on PowerBuilder applications using this deployment technique. What limitations exist typically are in the form of application scalability and interoperability with PowerBuilder. For example, an OLE or OCX call will occur on the CITRIX or Terminal Services server, so it is important that these components be properly configured and abstracted from the PowerBuilder code if the application is also to run on a traditional Windows client. Client-side integration is limited to abstracting the peripherals of the application, such as printers and local disks.

There are significant scalability issues with either CITRIX or Terminal Services. Although enterprise versions of both products offer clustering and load balancing, the amount of hardware required to support a given load is far higher than with other solutions. The load on the database is also higher than with other solutions because each terminal session maintains one or more stateful connections to the database. Application connectivity can also become unreliable over poor quality network connections since HTTP protocol is not used with terminal emulation software.

**APPEON**

Appeon for PowerBuilder is an intriguing product. Available since 2001 and now in its fifth release, the product originally transformed a PowerBuilder application into a combination of Java, XML, DHTML, and JavaScript. The Appeon product has two parts – Appeon Developer and Appeon Server. The Appeon Developer is an add-on to PowerBuilder that provides productivity and deployment features for the Appeon user via a toolbar within the PowerBuilder IDE.

The current version, Appeon for PowerBuilder 6.1, has been re-architected around HTML, JavaScript, AJAX and a proprietary ActiveX browser plug in, originally termed the Appeon Xcelerator in Appeon for PowerBuilder 5.0. The use of the plug in allows for significantly faster runtime execution, more faithful client rendition, and more application compatibility than before. The overall quality of the experience is second only to running CITRIX or terminal services on an intranet connection.

Appeon Developer features include a configuration wizard, Appeon configuration-specific property sheets, a feature called Code Insight, (a plug into PowerBuilder's AutoScript feature that highlights supported and unsupported PowerScript); an interactive debugger that looks like PowerBuilder's debugger but uses the Microsoft Script Debugger as the underlying engine, an Unsupported Features Analyzer (UFA), a runtime packaging tool to ease deployment, an administrative website for the Appeon Enterprise Manager (AEM), and help files. The net effect is a highly productive development environment that allows developers to perform virtually all tasks from within PowerBuilder.

The second part of the Appeon product is the Appeon Server. The Appeon Server is available in two versions: a JEE version that is deployed to Sybase EAServer®, BEA WebLogic®, and IBM WebSphere®, and a .NET version that deploys to IIS. There are some minor variations in features by both application server vendor and by OS platform, but the core functionality is the same. Supported OSs include: Windows, UNIX (IBM AIX, Sun Solaris, HP UNIX), and Linux (RedHat Linux).

Figure 16 shows the PowerBuilder 11.5 code examples running under Appeon. Note the version number is 0.0. This is because there is a global public function, f_getversion(), returns a global string from the application object, that has not been properly initialized.

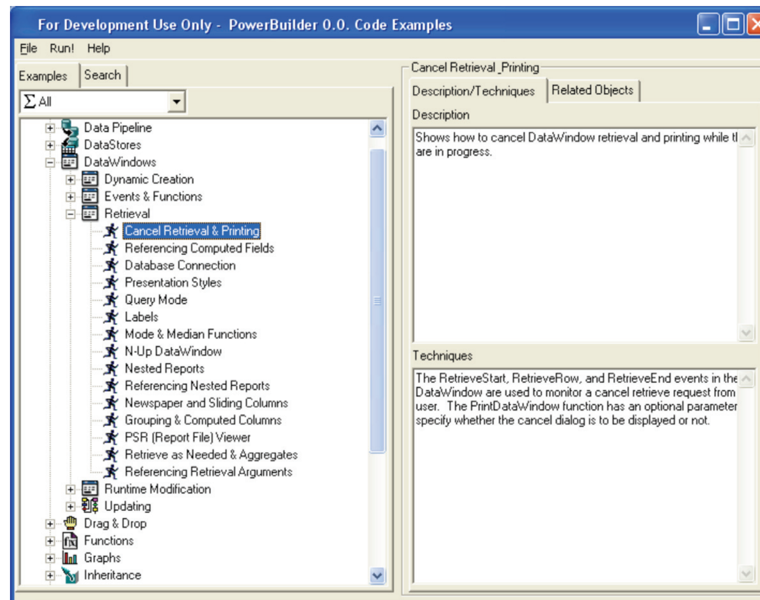



**Figure 16:** The PowerBuilder 11.5 code examples application running under Appeon

## Major Considerations

The major considerations for moving all or part of a PowerBuilder application to the web are: how much of the application needs to go to the Internet, the context of the company's technical environment, client-side requirements, and is the application a good fit for the Internet.

### HOW MUCH OF THE APPLICATION NEEDS TO GO TO THE INTERNET?

The first question to be answered is how much of the application's functionality needs to be Internet-enabled? Obviously, the fewer processes, windows, and reports that need to be exposed to the Internet the better. A good way to answer this is to examine the categories of users of your application and their needs. Another good way to answer this question is to examine which business processes would benefit from Internet enablement.

### WHAT IS THE CONTEXT OF THE TECHNICAL ENVIRONMENT?

The context of the technical environment of the organization must be considered as well. For example, if an organization is a Java shop, some options, like the Appeon for PowerBuilder versions that use various JEE containers, like the WebLogic ®, WebSphere ®, or EAServer ® versions would be given preference. On the other hand, a strict .NET shop would only want to look at Microsoft .NET solutions, while a best of breed shop with both Java and .NET should seriously consider EAServer 6.2, with its .NET, Java, PowerBuilder, C, and CORBA object model support.

### WHAT ARE THE CLIENT REQUIREMENTS?

Web client requirements can be divided into several categories. Categories include the need for cross-browser support, scripting language requirements or restrictions, availability or restrictions on the use of plug-ins, protocol(s) to support, and the like.

**IS THE APPLICATION A GOOD FIT FOR THE INTERNET?**

Not all PowerBuilder applications will move smoothly to the Internet with every approach described here. Variables such as the size and complexity of the application, the use of Windows-only features such as OLE, DDE, and third party OCXs, the number of DataWindows used in each Window, the size of the SQL result sets retrieved, the PowerBuilder framework used, and even coding practices and the general quality of the code will all determine the options open to you. Other variables that will all impact your decision include the complexity of the UI presentation style and the use of drag and drop in the application. The chart below should serve as only a general guideline and not an indication of which approach is best for your organization.

| Technique | PowerBuilder Windows Forms | PowerBuilder Web Forms | PowerBuilder Web DataWindow | CITRIX or Terminal Svcs. | Appeon |
|---|---|---|---|---|---|
| **MDI Interface** | Y | Y, using tabs | N/A | Y | Y |
| **Drag and Drop** | Y | N | N | Y | Y |
| **OLE, OCX** | Y | N | N | Y (server-side) | Y |
| **Client-Side Integration** | Y | Via JavaScript | Via JavaScript | Disks and printers only | Y |
| **Server-Side Integration** | Y | Y | Y | Y | Y |
| **Level of Effort** | Minor | Significant | Highest | Least | Minor |
| **Fidelity of UI** | Highest | Good | Good | Excellent | Excellent |
| **Strengths** | Easy, good UI, broad capability | Best for simple UI, ground up true web dev | Scalability, good for Java and low level control | Easy to set up and deploy | Client & server integration, scalability, performance |
| **Weaknesses** | Bandwidth, footprint on client | Event postback model inherent to ASP .NET 2.0 | Strong need for good 3-tier architecture, more hand coding | Scalability, cost | Technology created by a Sybase Partner. |

Sybase, Inc.
Worldwide Headquarters
One Sybase Drive
Dublin, CA 94568-7902
U.S.A
1 800 8 sybase

www.sybase.com

Copyright © 2009 Sybase, Inc. All rights reserved. Unpublished rights reserved under U.S. copyright laws. Sybase, the Sybase logo, Appeon, DataWindow and PowerBuilder are trademarks of Sybase, Inc. or its subsidiaries. All other trademarks are the property of their respective owners. ® indicates registration in the United States. Specifications are subject to change without notice. 04/09

SYBASE®