SYBASE®

WHITE PAPER

Updating your PowerBuilder® Application for Today's World

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

TABLE OF CONTENTS

INTRODUCTION

User Interface (UI) standards have evolved over the years and custom-developed applications haven't kept pace. Many applications developed in the 1990's now have a dated look and feel. With some minimal effort, applications can be given a fresh look by making a few key User Interface (UI) changes. At stake is no less than your user's satisfaction with your software.

BACKGROUND

Most of us don't remember much about Windows® Version 1 or 2. They were fairly basic windowing systems, and the primary issue was the lack of a multi-threaded, multi-tasking operating system to support a multi-tasking windowing desktop. To compensate for this, Windows developed a queuing mechanism that simulated multi-tasking in a single-tasking OS.

Windows really took off with Windows 3.0 in the 1990 timeframe. Still bootstrapped from MS-DOS, the primary user interface mechanism was a series of child windows with icons nested inside them (remember Program Manager, the forerunner to Explorer?) Clicking on an icon in a child window would either open a nested child window or launch a program. Programs were differentiated from Program Groups in this manner, and hierarchical menu systems developed in this way. One of the primary uses of Child Windows in PowerBuilder® at that time was to mimic Program Manager.

In that era, the most popular word processing program was WordPerfect™ and the most popular spreadsheet was Lotus 123™. Microsoft had a rather cryptic and difficult to understand word processing program called Word™ which was initially character-based. It wasn't until Word for Windows came out that the metaphors used in the prior versions of Word, such as formatting objects, styles, inherited formatting objects, and so forth became intuitively obvious, and Microsoft Word took off.

Following on the heels of its success with Word, Microsoft set out to write a program that would exceed the productivity bar set by Lotus 123 and came out with Microsoft Excel™. One of the first design ideas for Excel was the ability to edit multiple spreadsheets at one time, and hence the Multiple Document Interface (MDI) was created.

Other spreadsheets would also take aim at Lotus 123, such as Borland's Quattro™, which introduced three dimensional spreadsheets, now called pivot tables in Excel. Microsoft subsequently acquired a company that was competitive with Harvard Graphics™ called PowerPoint. This DOS-based program was reworked for Windows and the Microsoft Office productivity suite was born.

Microsoft has rewritten the rule book for UI standards several times. Following on the commercial success of Windows 3.x came Windows 95, 98, the soon forgotten ME, Windows 2000, Windows XP, Windows Vista, and Windows 7.0. Most large organizations have used a combination of Windows 2000 and Windows XP since 2000, many have shunned Vista, and most are eagerly awaiting the adoption of Windows 7.0. So if we developers want to excite and capture the imagination of PC users today we need to be able to offer a user interface experience matched to the underlying operating system. And, today, the term UI is not even in fashion. Now, it's all about the User Experience (UX.)

A TYPICAL POWERBUILDER UPGRADE SCENARIO

In this context, PowerBuilder applications that underwent initial design and development efforts in the mid to late 1990s were developed to look good in the Windows 95 and 98 operating systems, and many were designed to look like extensions to Microsoft Office by using the MDI interface. These efforts also look acceptably fresh under Windows 2000, but start to wear thin starting with Windows XP.

One issue with older PowerBuilder applications compiled under PowerBuilder 8.0 and previous versions is that all windows are rendered under XP differently, and as such, PowerBuilder 8.0 and prior applications clip the bottom and right margins of their windows. This is because of the way Windows measures the size of each window changed in XP. Of course, PowerBuilder 8.0 was optimized for Windows 2000 and changes Microsoft made to the Windows API forced this to happen.

Organizations have also moved to newer versions of Microsoft Office and want good interoperability with Office and the ability to leverage the user's experience with Office and Windows. Microsoft Office has undergone numerous revisions, and the current version is Office 2007, also known as Office 12. Only Excel is technically still MDI-based, and the rest of the Office products are no longer MDI.

Compounding the shift away from MDI is the trend towards browser-based applications and Windows Presentation Foundation (WPF), which is featured in PowerBuilder 12 .NET. It appears that the MDI interface is on borrowed time.

This article will discuss a variety of ways, some simple, some not so simple, to update your applications' user interface and experience standards. Along the way, we will explore Microsoft's stated UX direction and what is achievable through the intelligent use of PowerBuilder.

TECHNIQUES EXPLORED

Techniques explored in this whitepaper include:

- Moving Away from Windows Classic Style Controls
- Incorporating Contemporary Menus and Toolbars
- Using Treeview Datawindows to Create a Hierarchical View of Data
- Hierarchical Task Lists for Menus
- Upgrading to the New, Improved RTF Controls
- Improving on the Use of MDI
- Beyond MDI – The Windows Presentation Foundation

So let's get started.

Moving Away from Windows Classic Style Controls

Microsoft introduced a new visual style for controls with the Windows XP operating system. In Windows XP 's control panel, the user can choose to use a Windows XP theme that sets properties such as background, sounds, and icons, as well as the default style of controls. You can also choose to use the Windows XP style for controls even if you are not using a Windows XP theme.

The first thing most PowerBuilder developers should do when moving to PowerBuilder 11.5 or higher is to move away from Windows Classic Style controls by doing the following on XP and later development environments:

1. Select Tools, System Options from PowerBuilder's main menu, and uncheck "Use Windows Classic Style on XP". (Figure 1)

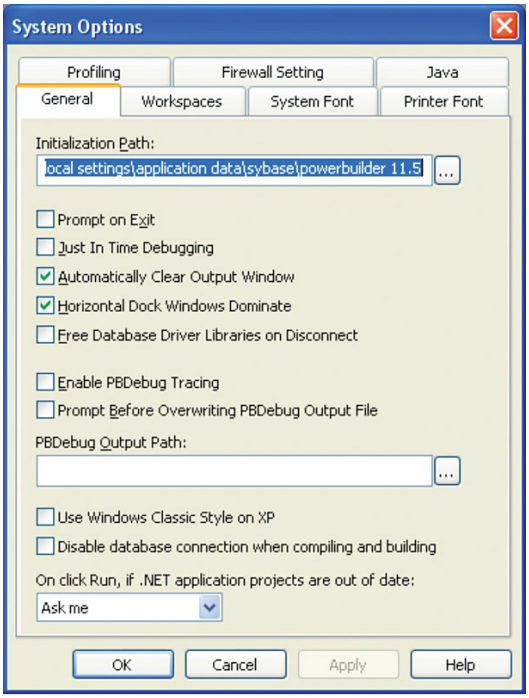

Figure 1: Enabling XP style controls in PowerBuilder

2. Restart PowerBuilder
3. For relevant PowerBuilder Win32-based Project Objects, open each project object and uncheck the "Windows Classic Style" checkbox. (Figure 2)

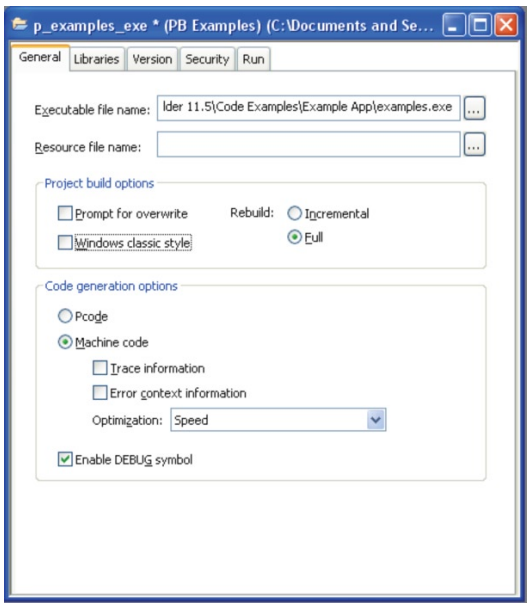

Figure 2: Deselecting the Windows Classic Style option in PowerBuilder Win32 Project Objects

The net effect is that all controls will have an XP look and feel if the operating system supports it. Figure 3 shows a PowerBuilder Example application window in Classic Style mode, and Figure 4 shows a PowerBuilder Example application window in XP Style mode.

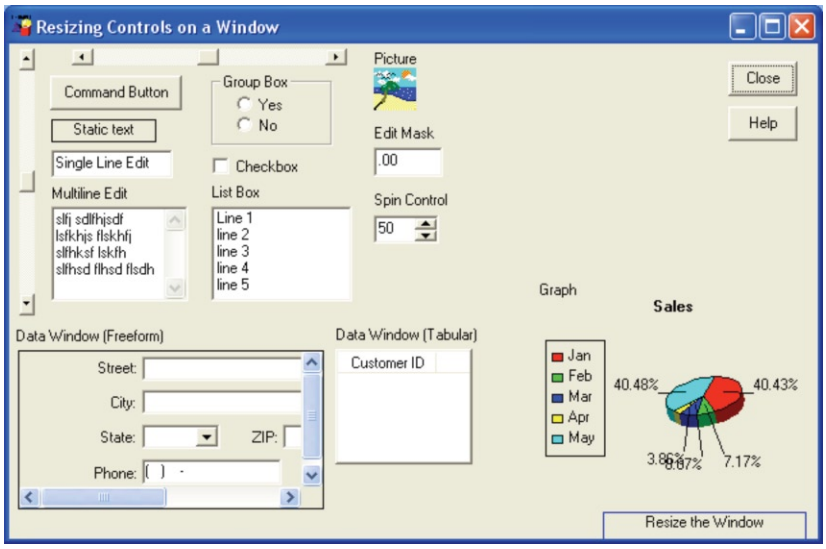

Figure 3: A PowerBuilder Example application window in Classic Style mode

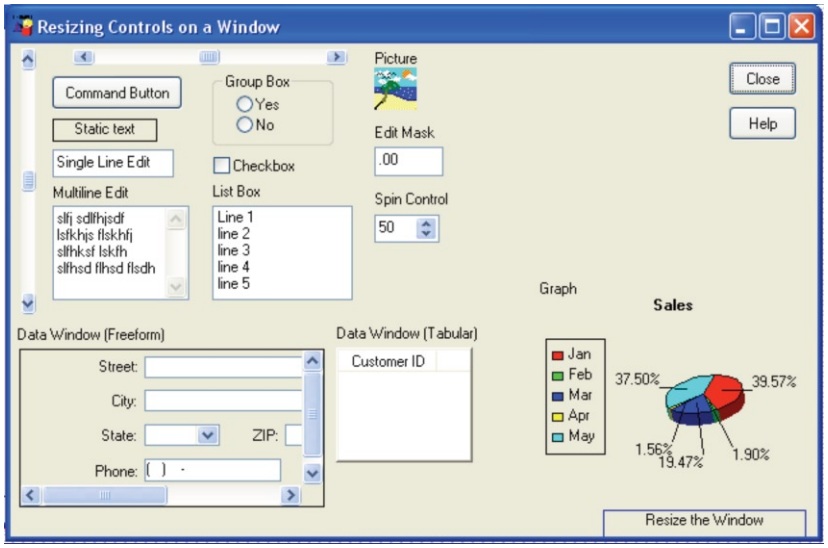

Figure 4: A PowerBuilder Example application window in XP Style mode

Incorporating Contemporary Menus and Toolbars

The next easiest thing to do to update a PowerBuilder application is to incorporate "contemporary" menus, toolbars, tooltips, and other UI mnemonics. These new menu styles were introduced in PowerBuilder 10.5. Menus imported or migrated from earlier versions of PowerBuilder use the Traditional menu style by default. Menus with a Contemporary style have a three-dimensional menu appearance similar to those in Microsoft Office 2003 and Visual Studio 2005, and can include bitmap and menu title bands.

In existing menu objects there is an enumerated property that needs to be set to contemporarymenu! Other menu object and subordinate menu object properties then give the developer additional control over the look and feel of the menu, including animation and more modern looking toolbars. As an example, Figure 5 shows a traditional style menu and Figure 6 shows a contemporary style menu.

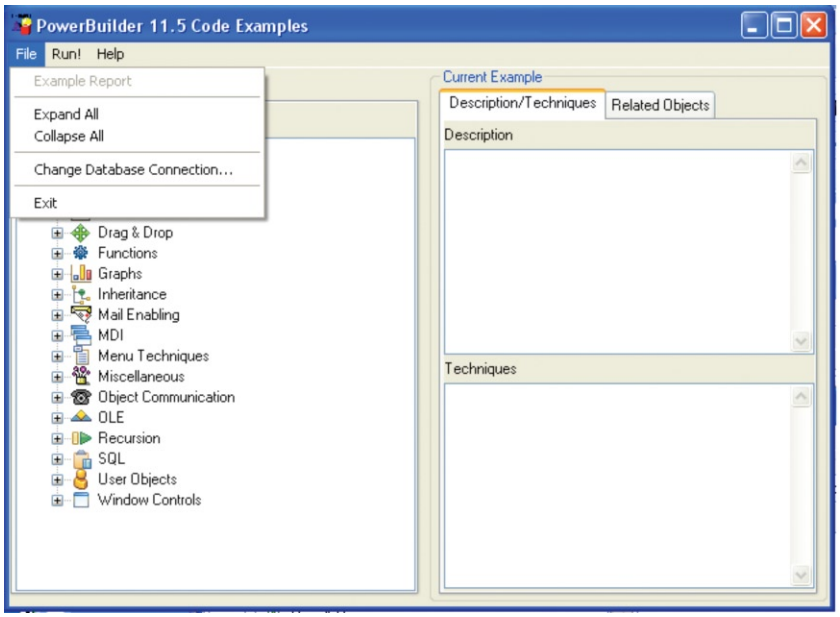

Figure 5: A Traditional Style Menu

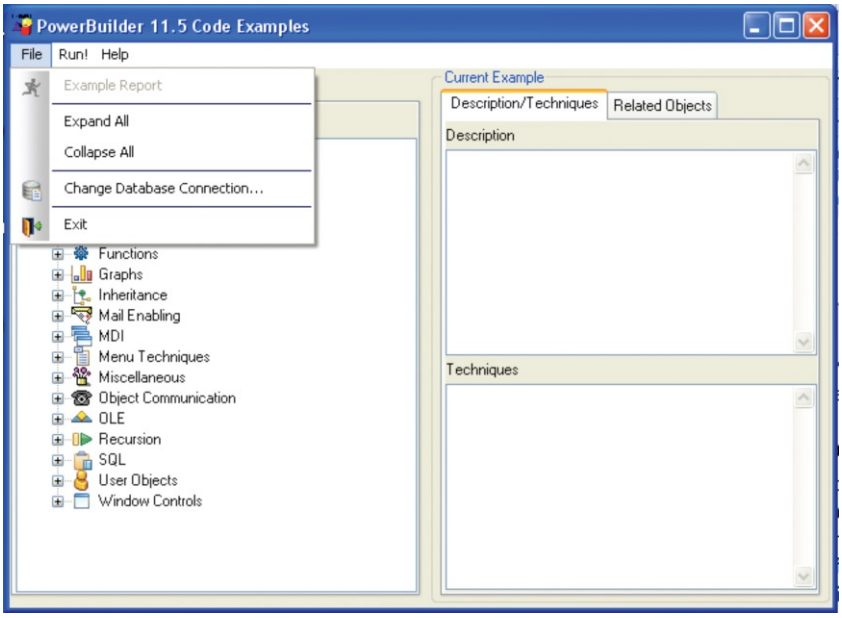

Figure 6: A Contemporary Style Menu

Because there are typically only a relatively small number of menus for a given PowerBuilder application, this is a quick and easy modification that makes a big difference to the user's experience. If you need additional modern-looking icons, just browse the web for examples countless icons suitable for contemporary menus.

Using TreeView DataWindows to Create a Hierarchical View of Data

Unlike the previous categories of UX enhancements, using TreeView DataWindows to create a hierarchical view of data requires a few new coding techniques and the reworking of existing windows. The TreeView presentation style provides an easy way to create DataWindow® objects that display hierarchical data, with rows are divided into groups that can be expanded and collapsed.

This new DataWindow presentation style was introduced in PowerBuilder 11. Using TreeView DataWindows has a significant advantage over using a TreeView control because it is easier to populate dynamically and easier to capture and process related events at runtime.

Creating and using a TreeView DataWindow is similar to creating and using a Group DataWindow, except with the TreeView DataWindow, the user can click the state icon to expand and collapse nodes. The other popular aspect of TreeView DataWindows is that they can have subtotals and group breaks like the Group DataWindow, making for a very powerful data drill-down presentation.

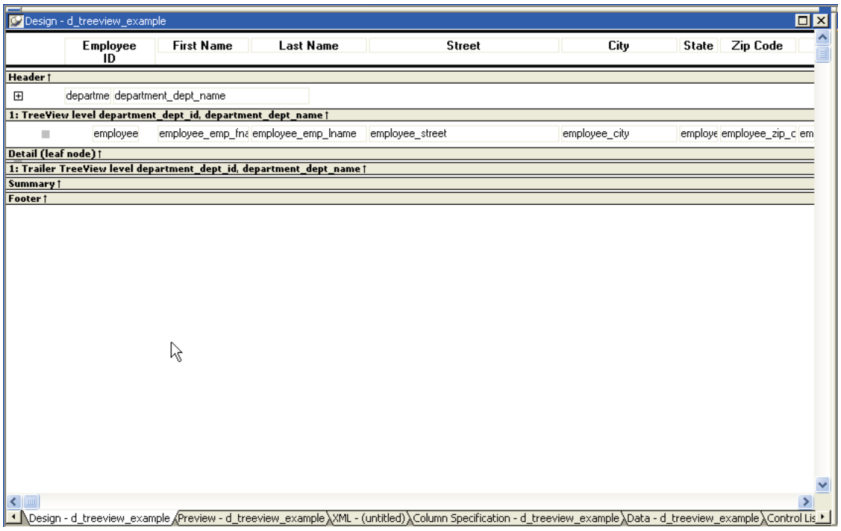

Figure 7: The TreeView DataWindow in design mode

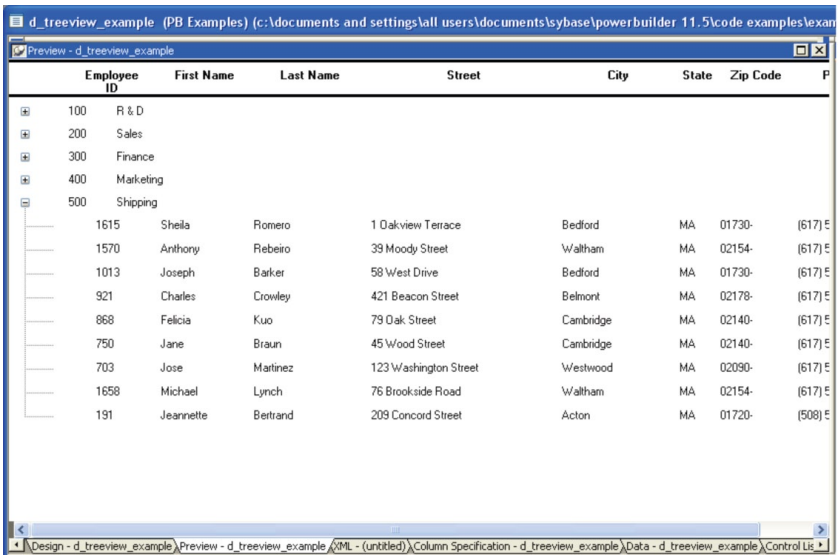

Figure 8: The TreeView DataWindow in preview mode

Upgrading to the New, Improved RTF Controls

PowerBuilder's aging RTF control was replaced in PowerBuilder 10.5 with a much more powerful control. This new control supports the current RTF specifications in the RichTextEdit control, the RichText DataWindow, and the new RTF column style. The new RTF control brings a modern look and includes some new features, including the ability to name and use formatting styles. Most of the properties and functions of rich text objects in previous versions of PowerBuilder continue to be supported by the new rich text editor.

Of no small importance is the added ability to save the contents of the RTF control in more modern formats. FileTypeDoc! saves the file in Microsoft Word format, FileTypeHTML! saves the file in HTML format, while FileTypePDF! saves the file in PDF format. In PowerBuilder 11.5 the RTF column style in the DataWindow allows for RTF to be easily stored and retrieved in a DataWindow's column, which renders as a rich text control once retrieved. Figure 9 shows a sample of the RTF DataWindow column style.

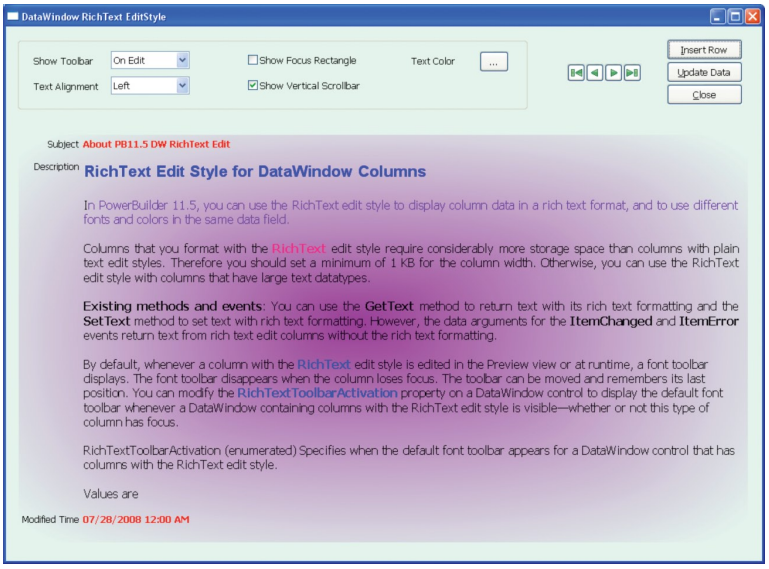

Figure 9: A Sample of the RTF DataWindow Column Style

Hierarchical Task Lists for Menus

Hierarchical task lists for menus can be created in a number of ways. The first way would be to use a TreeView DataWindow as described above, with hyperlinks that open other windows instead of a large MDI-style navigation menu. The advantage of such a list is that the developer could populate the DataWindow from a database based upon the user's security rights.

Another way would be to use a more modern control, such as Brad Wery's XPListBar control. Figure 10 shows an example from Brad Wery's PBGUIControls framework. Brad created his framework and the **www.PowerToTheBuilder.com** website to showcase PowerBuilder's capabilities in the area of GUI design and development.

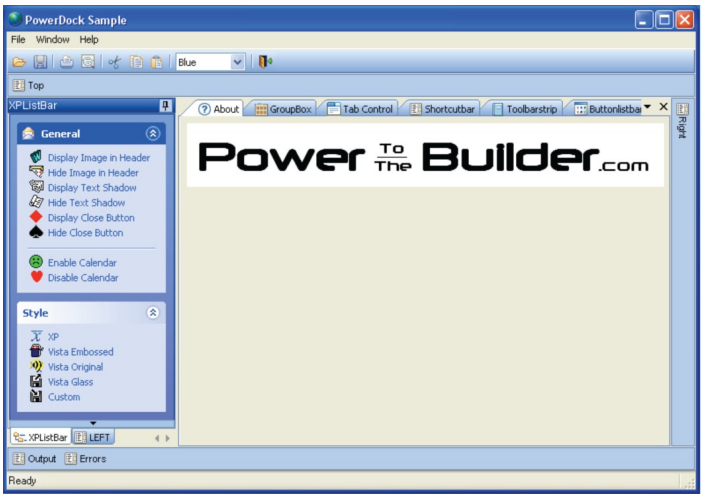

Figure 10: A Hierarchical Task List Developed the XPListBar Control

The advantage of adopting this approach to application navigation is that it would begin the process of moving away from using menus to open sheets in an MDI frame, thus laying valuable groundwork to moving either to a PowerBuilder Web Forms environment or a PowerBuilder 12 .NET WPF environment.

Other Modern UI Techniques

There are myriad other techniques for creating or modifying applications. These include the more modern controls that now ship with PowerBuilder, as well as some advanced frameworks. Many of these techniques result in an application that looks more "webby" and thus enhances the application's potential to be moved to the web using .NET Web Forms or WPF.

One of the easiest techniques is to use underlined hyperlinks controls in DataWindows and Window classes for navigation and data drill down. Hyperlinks in windows typically open either other windows or web pages and can be coded to open windows in a Win32/Win Form environment or browser windows or pages contextually. This typically involves little or no user training as hyperlinks have become ubiquitous with Internet navigation.

Many PowerBuilder applications were developed with developer-created progress bars to indicate the progress of long running processes. These can easily be replaced with the newer PowerBuilder progress bar controls, as illustrated in Figure 11. Track Bars were also frequently user-developed and can also be replaced with the new PowerBuilder-equivalent controls.

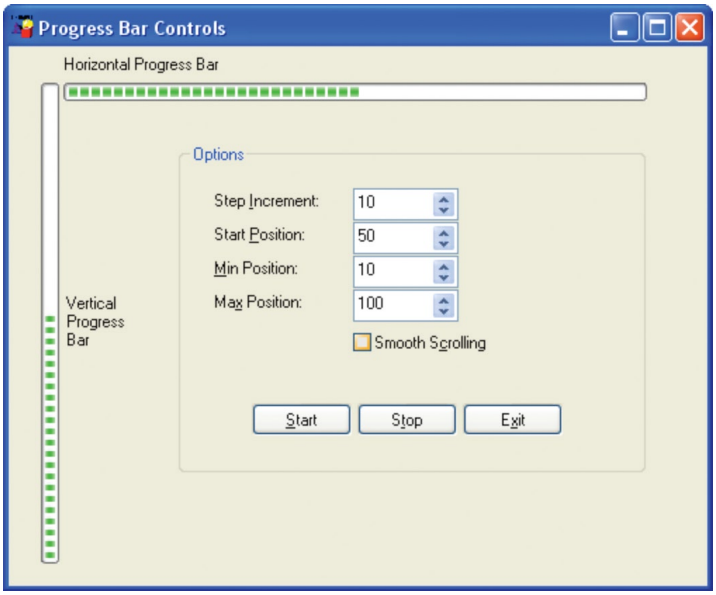

Figure 11: The Progress Bar Control Example from the PowerBuilder Examples Application

Moving beyond replacement of individual controls, entire frameworks have evolved in the PowerBuilder shareware market that have a much more modern look and feel, as well as employing more modern coding techniques frequently found in Java or .NET applications.

The Kodigo project (**www.Kodigopower.com**) is one example of a modern development framework for PowerBuilder. The Kodigo project is an open source PowerBuilder framework that is evolving into a developer suite for C/S and N-Tier development, complete with tools for unit testing, code generation and documentation.

The Kodigo project framework includes abstract classes for rapid application construction, application services, control containers to abstract Win32 and .NET Win Form development, DataWindow control classes, theme and layout manager support, resizable panels and tab containers, and low level classes to hook to the OS. Figure 12 shows a sample Kodigo project-developed window.

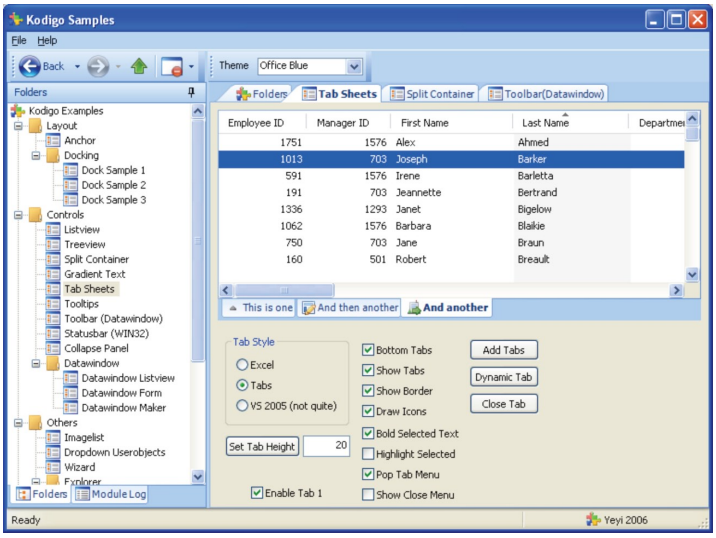

Figure 12: A sample Kodigo Project Window

Another intriguing new framework for PowerBuilder development is Brad Wery's PowerDock framework and related GUI components. Brad's website is **www.PowerToTheBuilder.com** which was mentioned previously. His framework includes a dock-based IDE-style interface, shortcut bars, gradient and theme support, XP-style listbars, modern dockable tab controls, modern looking button list bars, and toolbar strips, and other controls.

Most of the window controls are not WYSIWYG, however, and to overcome this Brad has created a PB UI designer. The designer allows the developer to customize the GUI controls and writes PowerScript that can then be pasted into the developer's application to achieve the desired results.

In addition to the examples at the www.PowerToTheBuilder.com site, an example of the look and feel of a PowerDock-based application can be seen on WerySoft's home page at **www.Werysoft.com**. The PowerDock framework uses a .DLL that allows for these modern-looking GUI controls to be deployed in Win32 and .NET Win Form environments, but does not support .NET Web Forms as of this writing due to use of the Handle() function in PowerBuilder. Figure 13 shows an example of a PowerDock-based application, the PowerDock Sample Application.

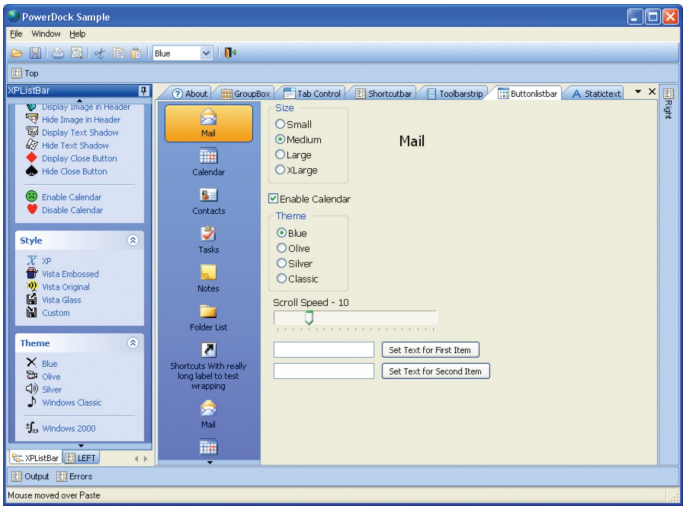

Figure 13: The PowerDock Sample Application

Improving on the Use of MDI

If reading the tea leaves at Microsoft is any clue, the traditional MDI has fallen out of favor in Redmond. Potential successors to the MDI interface include the Tabbed Document Interface (TDI) and the IDE-Style Interface. A TDI interface allows multiple documents to be contained within a single window, using tabs to navigate between them. It is an interface style most commonly associated with web browsers, web applications, text editors and preference panes.

An IDE-Style interface has a shell window with child windows residing under a parent window, with the exception of modal windows. Elements of IDE-Style interfaces include dockable and collapsible child windows, panes, a tabbed document interface for panes and sub-panes, splitters to resize the panes and sub-panes, and mechanisms to remember the users' preferences.

An IDE-style interface is distinguished from the Multiple Document Interface (MDI), because all child windows in an IDE-style interface are enhanced with added functionality not ordinarily available in MDI applications. Because of this, the IDE-Style interface can be considered a functional superset of MDI. Both Kodigo and PowerDock are prime examples of the IDE-style interface. Other IDE-style examples include Outlook, Eclipse, Visual Studio .NET, and PowerBuilder 12 .NET. Figure 14 shows the upcoming PowerBuilder 12 .NET sporting an IDE-Style interface.

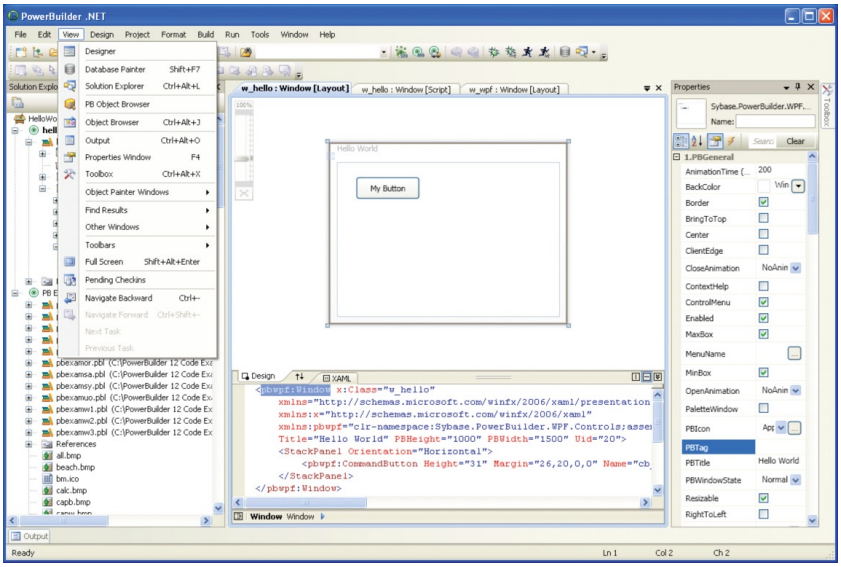

Figure 14: PowerBuilder 12 .NET as an Example of an IDE-Style Interface

Beyond MDI – The Windows Presentation Foundation

Microsoft assures us there is life after MDI, at least for Win32 applications. Microsoft has promised continued support of MDI in Windows Forms so existing Windows Forms applications will continue to have Microsoft's support for the foreseeable future. Programmers can continue to be productive working in a technology they are familiar with.

Microsoft is positioning WPF as complementary to, not a replacement for, Win32 and MFC code. WPF and Windows Forms can both be used in a composite application since each is capable of hosting user interface elements defined by the other. The two platforms have different strengths and can be used to complement each other. Windows Forms, for example, provides backwards compatibility, performance, and many useful LOB elements, while WPF uniquely offers things such as superior 3D graphics and animation.

The Windows Presentation Foundation (WPF) is a graphical OS subsystem for rendering user interfaces in windowing applications. Initially released as part of .NET Framework 3.0, it is designed to remove dependencies on the aging GDI subsystem. WPF is built on DirectX, which provides hardware acceleration and enables modern UX features like transparency, gradients and transforms. WPF also provides a consistent programming model for building applications and provides a clear separation between the user interface and the business logic.

WPF was created to allow developers to easily build rich applications that previously were difficult or impossible to build in Windows Forms, and applications that require a range of other technologies which are often hard to integrate. For example, WPF is suitable for applications that combine 2D & 3D graphics with highly-styled form elements, video, rich media and interactive visualization across a wide range of platform hardware.

WPF is Microsoft's UX road to the future. As a component of the Microsoft .NET Framework 3.5, the classes that support WPF are included in the .NET System.Windows namespace and can be utilized by any .NET language. All elements of WPF may be coded in a .NET language (C#, VB.NET). Whatever parts of this functionality it uses, the basic structure of every WPF application is much the same.

WPF applications can provide a traditional dialog-driven interface or a navigational interface. A dialog-driven interface consists of the common UX elements that every Windows user is familiar with. A navigational interface acts much like a browser, in that rather than opening a new window for a dialog it loads a new page. Interfaces are thus implemented as a group of pages, each consisting of a user interface defined in XAML together with logic expressed in a programming language.

A core aspect of WPF is its declarative markup language, called Extensible Application Markup Language, or XAML. XAML is a Microsoft-developed standard based on XML. XAML's specific advantage is that it is a declarative programming language. In declarative languages, the developer or designer describes the behavior and integration of components without the use of procedural programming. The XAML code can ultimately be compiled into a managed assembly, called a .BAML, similar to the way all .NET languages are compiled.

The introduction of XAML allows application designers to contribute to the application development cycle. Using XAML to develop user interfaces also specifically addresses the separation of model and view (the MVC pattern), which is considered a good architectural principle. In XAML, elements and attributes map to classes and properties in the underlying APIs.

A key element of WPF XAML is that a WPF application UX uses panels for layout not sheets like MDI. Each panel can contain one nested element, including other panels and controls such as buttons and text boxes. Different kinds of panels provide various layout options. For example, a DockPanel allows its child elements to be positioned along the edges of the panel, a Grid allows positioning its children precisely on a grid, and a Canvas lets a developer position its children anywhere within the panel's boundaries. Figure 15 shows the same Sample PowerBuilder 12 .NET application window as Figures 3 and 4, and Figure 16 and 17 show the underlying XAML of the WPF window class.

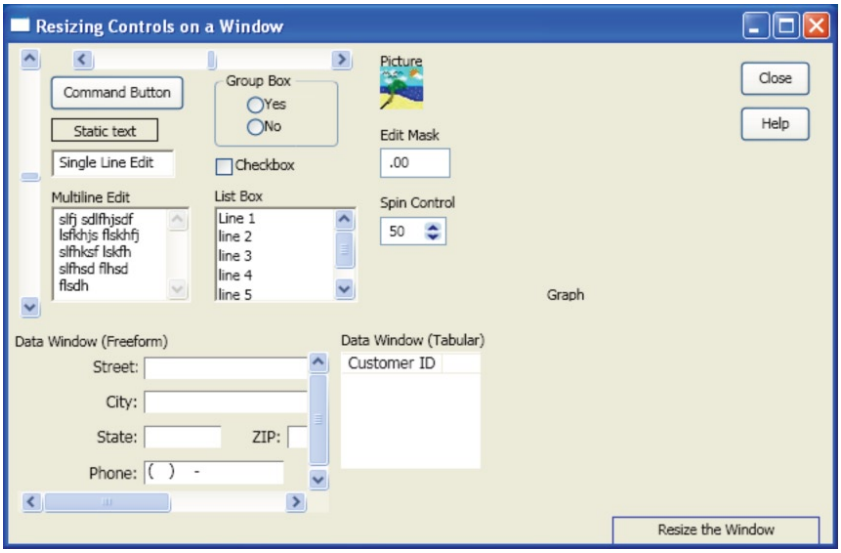

Figure 15: A PowerBuilder Example Application Window Ported to WPF and Compiled under PowerBuilder 12 .NET

```
<pbwpf:Window xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x=
    <Canvas Name="PBClientArea">
        <pbwpf:GroupBox Name="gb_1" X="722" Y="93" PBWidth="434" PBHeight="228" TabOrder
            <Canvas>...</Canvas>
        </pbwpf:GroupBox>
        <pbwpf:EditMask Name="em_1" X="1305" Y="535" PBWidth="232" PBHeight="87" TabOrde
        <pbwpf:CommandButton Name="cb_1" X="146" Y="100" PBWidth="473" PBHeight="109" Ta
        <pbwpf:StaticText Name="st_1" X="150" Y="234" PBWidth="377" PBHeight="71" Enable
        <pbwpf:SingleLineEdit Name="sle_1" X="150" Y="330" PBWidth="434" PBHeight="87" T
        <pbwpf:EditMask Name="em_2" X="1305" Y="324" PBWidth="246" PBHeight="93" TabOrde
        <pbwpf:MultiLineEdit Name="mle_1" X="153" Y="506" PBWidth="491" PBHeight="295" T
        <pbwpf:ListBox Name="lb_1" X="722" Y="506" PBWidth="502" PBHeight="295" TabOrder
            <pbwpf:ListBoxItem Item="Line 1"></pbwpf:ListBoxItem>
            <pbwpf:ListBoxItem Item="line 2"></pbwpf:ListBoxItem>
            <pbwpf:ListBoxItem Item="line 3"></pbwpf:ListBoxItem>
            <pbwpf:ListBoxItem Item="line 4"></pbwpf:ListBoxItem>
            <pbwpf:ListBoxItem Item="line 5"></pbwpf:ListBoxItem>
        </pbwpf:ListBox>
        <pbwpf:DataWindow Name="dw_1" X="1167" Y="951" PBWidth="491" PBHeight="359" TabO
        <pbwpf:Picture Name="p_1" X="1305" Y="74" PBWidth="150" PBHeight="132" PictureNa
        <pbwpf:Graph Name="gr_1" X="1892" Y="781" PBWidth="961" PBHeight="647" Enabled="
        <pbwpf:HScrollBar Name="hsb_1" X="225" Y="17" PBWidth="985" PBHeight="52" MinPos
        <pbwpf:VScrollBar Name="vsb_1" X="36" Y="17" PBWidth="57" PBHeight="833" MinPosi
        <pbwpf:DataWindow Name="dw_freeform" X="33" Y="954" PBWidth="1092" PBHeight="497
        <pbwpf:StaticText Name="st_7" X="153" Y="439" PBWidth="338" PBHeight="55" Enable
        <pbwpf:StaticText Name="st_8" X="1892" Y="740" PBWidth="246" PBHeight="55" Enabl
        <pbwpf:StaticText Name="st_11" X="25" Y="884" PBWidth="630" PBHeight="61" Enable
        <pbwpf:StaticText Name="st_9" X="2120" Y="1460" PBWidth="729" PBHeight="97" Enab
        <pbwpf:CheckBox Name="cbx_1" X="726" Y="346" PBWidth="424" PBHeight="71" Text="C
        <pbwpf:CommandButton Name="cb_exit" X="2568" Y="55" PBWidth="246" PBHeight="109"
        <pbwpf:CommandButton Name="cb_2" X="2568" Y="196" PBWidth="246" PBHeight="109" T
    </Canvas>
</pbwpf:Window>
```

Figure 16: The Underlying XAML of Figure 15, Displayed In a Color-Coded Text Editor

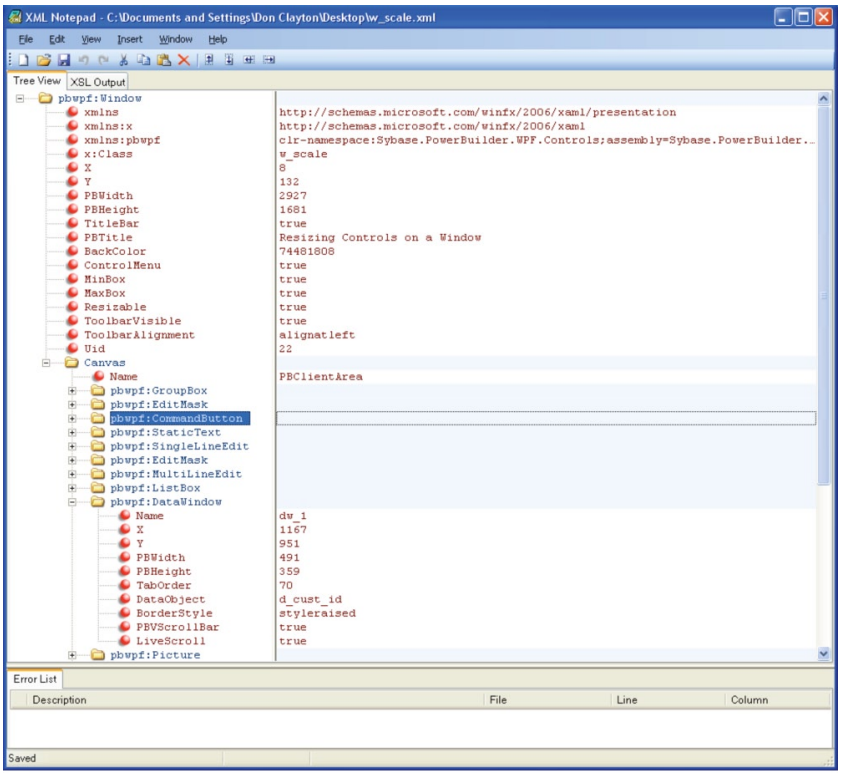

Figure 17: The Underlying XAML of Figure 15, Displayed In Microsoft XML Notepad

NEXT STEPS FOR POWERBUILDER DEVELOPERS

There are many forks in the proverbial road, and more coming up with PowerBuilder 12. So it is in an organization's best interest for PowerBuilder developers be prepared to add value to their organization. Developers should assess the company's use of the PowerBuilder application and determine the best steps forward, evaluating applications in light of current business practices and goals.

There are several different possible outcomes for how earlier development efforts have held up over time. The first consideration is whether the business has changed, and if so, how. The majority of companies that have considered various technology alternatives from Silverstream to Java to .NET to Flex have eventually reached the conclusion that it is usually more cost effective to enhance and modernize existing PowerBuilder applications that work than it is to start over with a new technology and a new learning curve

The second consideration is to review advances in technology. In addition to the many suggestions to update a PowerBuilder application presented here, there are other less obvious opportunities for application modernization, such as exploiting Web Services or developing a service-based architecture, moving processes or functions away from the client to an application server, creating .NET Smart Client applications to handle deployment headaches or moving parts of the application to the web.

Developers need to be able to understand the tradeoffs between PowerBuilder 12 .NET WPF applications and PowerBuilder 12 Classic, including .NET Win Forms and Web Forms applications. Once PowerBuilder Classic .PBLs have been converted to WPF there's no going back. (Of course, every developer will save a copy of their source code, and should the need ever arise to revitalize that Win32 application, it will be ready and waiting.) As mentioned earlier, interfaces like MDI are not supported, and it's not likely that a better metaphor than the tab metaphor that replaces MDI sheets in a .NET Web Forms application will emerge.

Developers need to understand WPF. Microsoft has clearly stated that this is the strategic direction of its user interface efforts, and separating WPF from the Windows OS and instead putting it in .NET 3.5 allows for the code to run on other platforms from smart phones to multi-touch surfaces. Developers also need to understand the role of XAML, XAML templates, which are similar to cascading style sheets, and BAML, the compiled managed code assembly.

CONCLUSION

Many techniques explored in this article are simple, low cost, and low risk. The migration to newer versions of PowerBuilder is required to take advantage of modern OS support, current database drivers, and to take advantage of the features discussed herein. Along the way, developers can do many simple things to modernize the look and feel of applications. Here are a few suggestions to help ensure that a developer's contributions are additive to the value chain of the organization.

Although it seems that presentation layer interfaces is a very dynamic topic right now, some things never go out of style. The partitioning of business logic and the separation of function from user interface, the layering of solutions by using abstraction and inheritance, and the adherence to current standards are all timeless. When developing with an eye to the future, be sure to stay committed to making your applications the best they can be.

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

www.sybase.com

SYBASE®