

Migrating Legacy Client/Server PowerBuilder Apps to Web-Enabled PowerBuilder Apps

A step-by-step process

BY VAZI OKHANDIAR & SACHIN AGARWAL

In today's competitive IT environment, organizations are reducing development and maintenance costs, and improving the accessibility of their legacy applications. With the growth in technology and demand for n-tier architecture-based applications, corporations now have an opportunity to transform their "legacy" client/server PowerBuilder applications to Web-enabled PowerBuilder applications by migrating to PowerBuilder 11.0. PowerBuilder 11.0 provides the functionality to transform PowerBuilder client/server apps to Web-enabled .NET PowerBuilder applications with minor redevelopment, instead of rewriting the PowerBuilder application in J2EE or a .NET Framework, which can be an expensive proposition. The initiative to migrate legacy PowerBuilder applications to Web-enable PowerBuilder 11.0 applications, allows corporations to maintain pervasive, efficient, and cost-effective methods of accessing the data and functionalities that were provided in large and complex PowerBuilder applications.

The process of migrating a mission-critical PowerBuilder application can be complex, risky, expensive, and time-consuming, if the migration and deployment planning isn't done properly. If planned correctly, the migration of a legacy PowerBuilder application to a Web-enabled PowerBuilder 11 application can be achieved for a fraction of the cost, as compared to the cost of rewriting that application in .NET or J2EE.

We recently migrated several legacy applications, which were developed in PowerBuilder 8.0,

9.0, and 10.0 to .NET PowerBuilder 11.0. In this article, we'll focus on the best practices and the steps we took during the migration of an application from a PowerBuilder 8 (or higher) to a Web-enabled N-tier .NET PowerBuilder 11 application.

The Migration Process

The first step in migrating a legacy application to a Web-enabled application is to understand the architecture of the legacy application, since client/server and Web-enabled applications are significantly different in their architecture.

- A client/server PowerBuilder application is based on a two-tier architecture that can have a thin client or a fat client. If the application is based on thin client architecture, then the business logic resides on the server side. In that case, the business logic may be embedded in the stored procedure. If the application, however, is a fat client, then the business logic is embedded in the user interface.
- A Web-enabled PowerBuilder application is an n-tier architecture where business logic resides in the middle tier (see Figure 1). Typically, in a Web-enabled application, a separate layer is maintained for the user interface, business layer, and database layer to reduce maintenance costs and increase availability and accessibility. As a result, once the application is migrated to a .NET environment, the business logic needs to be separated from the user interface layer and

moved to the middle tier, and tested to ensure that the functionalities have remained intact.

The next step is to analyze the existing legacy PowerBuilder application and understand the requirements for Web-enabling it. The analysis phase includes meeting the stakeholders, understanding their application and the business and technical requirements that are to be met by Web-enabling the application. The following are some of the items to be considered during the requirements analysis phase:

- Analysis of the current legacy application, including the DataWindow objects used for updating and retrieving data.
- Differentiation between the non-visual user objects (NVOs) that could be easily migrated to .NET, and those that would require rewriting.
- Review the relevant application documents and baseline source code. Usually the business logic in a legacy PowerBuilder application is embedded in the events (DataWindow, button, and window) functions and NVOs. To migrate the application to the Web, it's important to differentiate between the information for the business, database access, and user interface layers.
- Elaborate the details of the application's current functionalities that need to be migrated to the application's server side and client side. It's important to maintain the data validation on the client side, whereas, the business logic has to be moved to the middle tier.

In our recent migration project, to transform a legacy application to a Web-enabled application, we used a two-phase approach. The first phase was to upgrade the legacy application to a client/server PowerBuilder 11 application. The second phase was to migrate the client/server PowerBuilder 11 application to a Web-enabled N-tier PowerBuilder 11 application.

Phase 1 – Migrate the PowerBuilder 8 Application to PowerBuilder 11 Application

The following are the steps used while converting the client/server PowerBuilder 8 application to the client/server PowerBuilder 11 application:

Step 1: We started the migration process by making a backup of the existing system. It's very important to make a backup of the existing Pow-

erBuilder code (PBLs).

During the migration process, PowerBuilder 11 replaces the old code with new code. As a result, a user can't fix the bugs in the legacy application if the old code has been replaced with the new code.

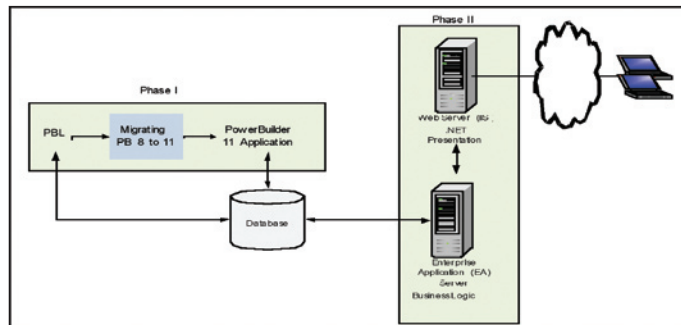


Figure 1

Library	Object Name	Object Type	Context Where Found	Line Where Found	Search Value	Suggested Replacement
clnlogica.protocolologica.document_of_core	Menu	of_security subroutine	public subroutine of _security (menu as_item, string as_security) throws ex_alframe_security		throws	New Reserved Synt.
		of_security subroutine	public subroutine of _security (menu as_item, string as_security) throws ex_alframe_security if isvalid(as_item) then		throws	New Reserved Synt.
		of_security subroutine	public subroutine of _security (string as_username, string as_security) throws ex_alframe_security		throws	New Reserved Synt.
		of_security subroutine	public subroutine of _security (string as_username, string as_security) throws ex_alframe_security; menu as_item		throws	New Reserved Synt.
		of_security subroutine	throw create ex_alframe_security		throw	New Reserved Synt.
		of_security subroutine	throw create ex_alframe_security		throw	New Reserved Synt.

Figure 2

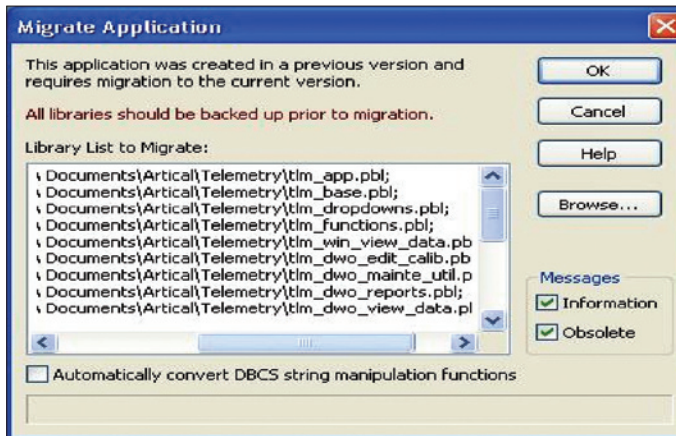


Figure 3

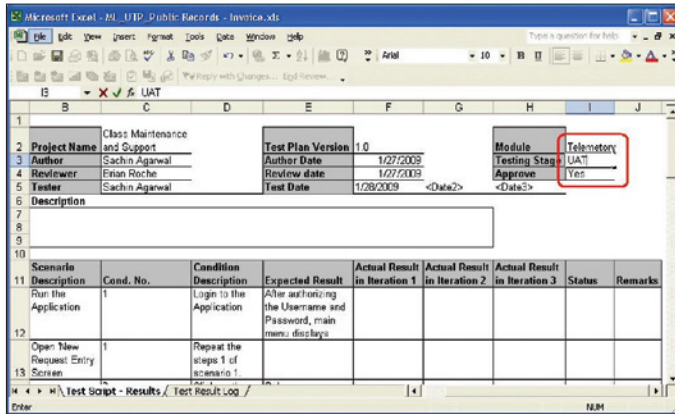


Figure 4



Figure 4

Step 2: Develop the unit and integrated test cases for the legacy application so that once it's migrated to PowerBuilder 11, the migrated application can be tested for accuracy. First test the application using the developed test cases in the PowerBuilder 8 environment. Then store the results. Once the application is migrated, test the PowerBuilder 11 application again to compare the results.

Step 3: Set up a development environment with the appropriate database connections and programming tools. It includes the following:

- Set up two development environments; one for testing and bug fixing the legacy application and one for the PowerBuilder 11 development environment for migration, testing, and User Acceptance Testing (UAT).
- Then set up the PowerBuilder 11 database server connectivity using Open Database Connectivity (ODBC) or Object Link Embedding Database (OLEDB) using PowerBuilder's Database Painter. The Data Source Name (DSN) and database connectivity from PowerBuilder 11 to the source database is to be established prior to

proceeding further.

Step 4: Before starting the actual application migration process, we used PowerBuilder's Migration Assistant Tool, provided by Sybase, to help identify potential problems that may arise during the actual migration process. The Migration Assistant Tool scans the legacy libraries (PBLs) and highlights all the obsolete functions and events used in the legacy application.

PowerBuilder makes it very easy to use the Migration Assistant Tool. Just start PowerBuilder 11 and select New Option in the File menu. This opens up a new window. In the new window, select the Migration Assistant Tool in the Tools Tab, which brings up the Search for Syntax of Type window. To get a list of all the PowerBuilder functionalities used in the legacy application that are no longer supported by PowerBuilder from versions 8, 9, 10, 10.5, select the option PowerScript New, Obsolete or Removed Syntax in The Search for Syntax of Type window. Next, select the libraries needed to migrate from the legacy application to PowerBuilder 11 and click on the Finish button. This starts generating a list of all the obsolete functionalities used in the legacy application and are no longer supported in PowerBuilder 11.

After the Migration Assistant process is complete, the tool generates a Migration Report with a list of the obsolete syntax found in the application. The report lists the exact location, the object name, the function/event name, and the line number in the code where the obsolete syntax was found in the library (PBL), similar to the one shown in Figure 2.

Step 5. The next step is to share the report generated by the Migration Assistant with the stakeholders and identify the fixes for each of the issues listed. We found it useful to log the bugs identified by the Migration Assistant tool with the relevant information, such as who modified the code, when the modifications were added, and the reason in case it was necessary to undo our changes.

Step 6. Once all the issues listed in the Migration Report are resolved in the legacy application, the next step is to actually migrate the application to PowerBuilder 11 using the following steps:

- **Step 6.1: Start PowerBuilder 11 and select the**

Open option under the File menu. In the Targets – To Be Migrated window, select the option Target To Be Migrated and click OK.

- **Step 6.2: In the Migrate Application Window, select all the libraries for the legacy application that need to be migrated** and check the boxes next to the “Information and Obsolete” so the results are displayed after the migration is done.
- **Step 6.3: After the migration process is completed, PowerBuilder will list all the warnings and errors generated during the migration process.**
- **Step 6.4: Based on the errors reported by the migration process, fix the errors in the legacy application** and restart the migration process again. Repeat the migration process (Step 6.1 to Step 6.4) until there are no more errors reported.

Note: If an application uses the PFC framework then all the PFC libraries need to be migrated to the latest version. The PFC application workspace alone can be migrated, or the old PFC libraries can be replaced with the new libraries. The latest versions of the PFC libraries are at <https://powerbuilder.codexchange.sybase.com>.

Step 7: Once the application is migrated to a client/server PowerBuilder 11 application, use the test cases to test it. We did both unit level testing and integrated level testing.

Note: As a best practice it's a good idea to maintain a log of the issues faced during the migration process and the resolutions used to resolve any issues. If there are any roadblocks faced during the migration process, work with the stakeholder, provide a possible work around, and put together a plan to overcome the blockages.

Based on our experience in PowerBuilder legacy application migrations, we found that 80% of the functionalities in the legacy application could be migrated using PowerBuilder 11, without facing any major roadblocks. The other 20% of the functionality can be migrated efficiently and effectively with proper planning.

At this stage, we have a fully tested and functional client/server PowerBuilder 11 application. The next step is to move to Phase 2 and convert the client/server PowerBuilder 11 app to a Web-enabled .NET PowerBuilder 11 application once all the anomalies are addressed.

Note: Before moving to Phase 2, it's a good

practice to make the sure the application is well tested by the end users and that the stakeholders have signed off on Phase 1 acceptance.

Phase 2: Migrating from a Client/Server PowerBuilder 11 Application to a Web-Enabled PowerBuilder 11 Application

For Phase 2 we used PowerBuilder 11 to deploy the client/server PowerBuilder 11 applications to browser-based ASP.NET 2.0 applications.

Setting Up the Web Server

Before starting the deployment of the application, we had to set up and configure a Web server. We used the following steps to set it up:

1. Installed the .NET frame work 2.0 Redistributables and the .NET frame work 2.0 SDK downloaded from the Microsoft Web site.
2. Configured Internet Information Services (IIS) on the Web server.
3. Configured ASP Defaults (the ASP.NET version depends on the SDK version) for Web sites
4. For printing the document in the Web application, we installed and configured:
 - a. The PS (PostScript) printer available on the system by renaming its profile to Sybase Data Window PS
 - b. Ghost script (<http://www.ghostscript.com/doc/GPL>)



5. Verified that the following DLLs were installed, to run the PowerBuilder application on the Web:

- a. Managed Code DLLs (Global Assembly Cache GAC)
- b. Unmanaged Code DLLs (Normal PB runtime files)

Note: These DLLs are installed and configured automatically during the PowerBuilder installation process.

6. Tested the IE Web controls to confirm that the following controls functioned properly on Web
 - a. TreeView Control
 - b. Tab Control
 - c. ToolBar Control

Creating a .NET Web Form Target

Once the Web server was configured, the next step was to create a .NET Web Forms Project and deploy the client/server PowerBuilder 11.0 application to a .NET application by using the .NET Web Forms Application Wizard that comes with the PowerBuilder 11.0. PowerBuilder 11.0 provided all the tools needed to deploy the client/server application to the Web.

To start the .NET Web Forms Application Wizard, we started PowerBuilder 11 and selected the New option in the File menu. In the New window, the target tab provided the .NET Web Forms Application wizard. We clicked on the wizard and then clicked the OK button to start the Web Forms conversion process.

Since we were migrating an existing application, we selected the “Use an existing library and application” option in the Create the Application Window. Next, we selected the library that needed to be migrated to a New WebForm application simply by specifying the target and the project name for our Web application. Then we provided the Web application name, the location for the resource files (.PBR files), images used in application, DLLs/API calls (if used in the application), and JavaScript (if used in application) to be deployed on the Web server.

After the wizard completed the deployment process, the following components were created on the Web server:

1. PBDs and JavaScripts for the selected PBLs that contained the DataWindows
2. Power Scripts that were converted to C# code

- and compiled to form the DLL (.NET assembly)
3. default.aspx (homepage)
4. Basic ASP pages required to run the application in the browser

At the end of the deployment process, the .NET Web Forms Application Wizard provided a list of all the features and functionalities that weren't supported by the Web-based application. We then had to develop the unsupported functionalities in .NET.

Note: Some of examples of unsupported features include the Property 'Title' of the DataWindow, Copy/Past feature, use of the Oval/Rectangle, and OLE controls. The PowerBuilder 11 Help Manual provides a complete list of unsupported functionalities.

Once the application was successfully deployed on the Web server, the application was available to the user through the browser.

Conclusion

Based on our experience in developing Web-enabling applications, we've found that using PowerBuilder 11 tools, such as the Migration Assistant Tool and the .NET Web Forms Application Wizard, made this a very easy, cost-effective, and efficient process to Web-enable client/server legacy PowerBuilder (8 and above) applications. Following this step-by-step process can help mitigate any potential issues or errors in the migration process. Having a specific project plan can make it a much easier process than expected, while achieving the desired outcome. ■

About the Authors

Vazi Okhandiar is the Vice President, Technology for mLogica. For over 19 years, she has been architecting, designing and managing PowerBuilder-enabled applications. Vazi has previously worked for EDS and Computer Science Corporations (CSC) and holds an MBA and BSEE from UC, Irvine and an MS in computer science from IIT, Chicago.
vazi.okhandiar@mlogica.com

Sachin Agarwal is a senior PowerBuilder Consultant with Sybase, Inc. He has over 8 years of experience in architecting, designing and implementing PowerBuilder Applications for clients such as mLogica, Air Quality Management District, Software Spectrum and Sybase IT. Sachin holds a BS and MS in Computer Applications.

Sachin.Agarwal@sybase.com