



# Migrating Existing PowerBuilder Apps to the Web

By Ramesh Babu Ve and B.G. Balaji

## *Moving your eCommerce applications to the web*

In today's IT environments, eCommerce transaction processing needs to be spread across the enterprise to facilitate exchange of information between customers, suppliers, and users. Most organizations use their own or third-party software to capture data from all these internal and external resources in a timely manner. To achieve this, however, enterprise data must be available to the customer or salesperson to place the order directly from their location.

Now, the question is, "How do we provide data access to customers or sales people who are on the road or otherwise mobile?" Creating an intranet or Internet-based application is one answer, but this takes time and is expensive. The best and fastest solution is often to migrate your current PowerBuilder application to the web.

Such a migration provides its own set of challenges. One of the greatest of these is that in some environments, business rules have been activated only in certain divisions of the organization; while in others, rules are consolidated and centralized, to be accessed from multiple points. You will need to assess and understand your two-tier application thoroughly beforehand to ensure a successful migration to three tiers.

## **The Migration Process**

Let us take a typical two-tier PowerBuilder application migration as an example. The first step is to analyze your existing application. Carefully study the list of objects implemented in it. Most applications use datawindow objects for data update and retrieve, and these can be easily taken to the web with some limitations. Like datawindow objects, the nonvisual user objects (NVOs) also can be taken to the web without a problem.

The next step is to find where the logic is implemented. In typical PB applications, the logic is implemented in datawindow events, button events, window events, functions, and NVOs. You should identify implementation of the business logic, data access logic, and screen level validations from these objects.

Third, determine what functionality needs to be moved to the server side and to the client side. All database access logic should be moved to the server side, since the browser client will not support a database connection. All business-specific logic, such as selecting a product rate based on the quantity entered, for example, should be moved to the server side. These business rules can be accessed from the web and PowerBuilder application simultaneously. All resource-intensive



*Ramesh Babu Ve and B.G. Balaji are PowerBuilder consultants with Ewak Creative Compusoft in Chennai, India. they can be reached at [ve.ramesh.babu@ewaksoft.com](mailto:ve.ramesh.babu@ewaksoft.com) and [bgb@ewaksoft.com](mailto:bgb@ewaksoft.com)*

processes, such as the processing of large data, should be moved to the server side, as should media interactive processes such as email and report processing.

Segregate the validations as *business validations* and *front-end validations*. Validations that change with the business are business validations, and should stay on the server side. For example, a product discount may change based on the market value, competition, etc. The client-side validations, however, never change. For example, the validation for *quantity should always more than 0* should be performed on the client side.

Finally, you must determine how to move the logic. Migrate all the business logic and data access logic to the server side as component methods, and move all front-end client side validation logic as either ASP or JSP scripts.

### Creating Server Side Components

You can create a new server-side component using the wizard available from the PowerBuilder, or convert the existing NVOs with little modifications. Once you have done so, move the business logic as the component methods. When creating the methods in the server component, do not merge all the functionality in a single component. For example, it is best to create separate components for customers and products. Stuff all the methods related to the customer (i.e., methods for retrieving, updating, and validating customer data) in the customer component while placing product-related methods in the product component. This will help you to manage the components more effectively.

Also, consider keeping separate components for web and business-related functionalities. If you keep both web and business rule-related methods in a single component, the client application has access to the web methods that it does not need. For example, keep a separate component (you can call it the web component) to generate HTML from the customer entry datawindow and apply the changes from the browser client methods.

### Creating an Ancestor Object for the Server-Side Component

As you can see, it is common to place more functionality in the server-side components. For example, the database access logic is common for all server-side components. To avoid scripting this logic in each component, however, create an ancestor object and place this logic in it. Also, you can add scripts in the ancestor object for generating the HTML, reading default values from the ini file, declaring the default instance variables, error logging, etc.

### Moving Existing NVOs as Server-Side Components

You can easily move your existing NVOs as server-side components. Create new “Activate” and “Deactivate” events and map to default events *pbm\_component\_activate* and *pbm\_component\_deactivate* respectively. Instead of creating these new events in each individual NVO, create them in the ancestor object.

### Moving Existing Datawindow Objects onto the Web

As mentioned earlier, you can usually use the existing datawindow objects for migration except for cross-tab, graph presentation-type datawindows which are not supported. Freeform, tabular and grid-type datawindows can be migrated easily to the web, but you will need to set additional properties in the datawindow painter. Graphic objects such as line, rectangle, etc. will *not* be supported on the web. You must replace these line objects with text objects to display the line in the generated HTML.

You can do this by creating a text object and adding the text `<HR WIDTH=100% align=left size=1 color=black noshade>` and **Select 'Value is HTML'** property for the text object to display the line in the HTML page.

### Moving Scripts into Server-Side Components

#### Datatypes

Before moving scripts into the server-side component, you need to consider the PowerBuilder data types, variables, etc. Not all PowerBuilder datatypes are supported in the server-side components. For example, objects such as datastore, transaction, and window cannot be passed as parameters for server-side components, but they can be used internally within the component (i.e., a datastore instance). Likewise, the datatype 'ANY' is not supported in the server-side component methods.

#### Datawindow Objects

All datawindow reference in the script should be changed to datastore. Use datastore to retrieve and update the data in the component script.

The two-tier PB script:

```
dw_1.retrieve(li_id)

dw_1.getchild('product_id',ldwc)
ldwc.retrieve(100)
```

should be changed to:

```
ids.retrieve(li_id)

ids.getchild('product_id',ldwc)
ldwc.retrieve(100)
```

in the server-side component script.

### Handling Messages

In typical two-tier applications, messages are shown immediately to the user whenever there is a data error, process success, or failure. But in the web implementation, you cannot pop up the message box in the server-side script. Instead you must return the message back to the client and display the message in the browser. The error messages can be returned as method return or as in referenced argument. So, remove all the message box scripts from the server-side script. The two-tier script to display the error message `MessageBox('Error', 'Invalid data')` can be changed as follows:

As method return:

```
return ('Invalid data')
```

As referenced argument

```
as_msg = 'Invalid data'
return -1
```

### Variable Initialization

Add code in the server side component's "Constructor" event to handle one-time settings such as *assign the dataobject, creating transaction server context, initiate connection to error log service, initializing the instance variables*, etc. In the "Destructor" event, add code to destroy any object instances.

### Moving Command/Picture Buttons

In two-tier applications, the user always interacts with the system by command or picture buttons for data retrieve, update, processing, etc. The buttons can be included in ASP/JSP script, or you can add buttons in the datawindow objects and map actions like retrieve, update, append row, etc. When the buttons are included in the datawindow, the buttons are displayed in the browser from the generated HTML, which reduces the script in ASP/JSP

### Database Connections

In two-tier applications, the database is connected in the application open event and it remains live till the application is closed or terminated. In the two-tier environment, each user has a separate database connection and number of users is predicted. But in a web scenario, the number of users can expand in a very short time. You also cannot predict the number of users and you cannot increase the database connection easily, since it requires more resources and is very expensive. To resolve this dilemma, let us use the connection cache facility available in the Jaguar component server.

In this three-tier application, you should not keep the database connection live. Once you finished the data retrieval or update, release the connection immediately. To achieve this, code the database connection in the component "Activate" event and disconnect in the "Deactivate" event.

The following sample script shows the typical two-tier PB application open event to connect the database:

```
SQLCA.DBMS = "ODBC"
SQLCA.DBParm = "ConnectionString=DSN=EAS Demo DB
V4;UID=dba;PWD=sql"
CONNECT USING SQLCA;
```

For the web, this script should be changed as follows:

```
SQLCA.DBMS = "ODBC"
SQLCA.DBParm = "UseContextObject=Yes,CacheName='EASDemo4'"
CONNECT USING SQLCA;
```

In the application open event script, the database is connected using the DSN name, whereas in the server-side component, it uses the Jaguar connection cache name instead of the DSN name to avail the connection cache pooling.

### Transaction Management

In a two-tier application, the data update logic is called sequentially. For example, in a sales order system, the order header data is updated first, then detail data, and finally the item master. Whenever there is an update failure in detail or the item master, the whole update is rolled back to avoid a commit. This all takes place in a single transaction.

But if you migrate the update logic to different order and item components, then you cannot handle the transaction. For example, assume that you call the order component first and item component next for update. When the update fails in the item component, only the item update is rolled back.



But you cannot check the update status of the item in the order component, since the item component is out of scope once the method is finished. The data is committed if the update in the order component is successful, which results in a data integrity loss.

To overcome this, let Jaguar handle the transaction. Do not use `commit/rollback` commands; instead, you may use `setcomplete/setabort` commands to handle the transaction by Jaguar.

The following script is in the “Save” option of the two-tier PB application:

```
If dw_1.update () = 1 then
    Commit using sqlca;
Else
    Rollback using sqlca;
End if
```

This script should be then changed as in the server-side component scripts:

```
If ids.update () = 1 then
    SetComplete ();
else
    SetAbort ();
End if
```

### *Moving Data Across Components*

The data from datastore cannot be passed from one component to the others, since the PB component deployed as a CORBA object does not support the PB-specific datatypes. To pass data from one component to another, you can use datawindow synchronization methods. Convert the datastore as a blob and pass it to other component methods which you convert back to datastore. To convert the datastore as a blob use the syntax:

```
Ids.getfullstate(lblob)
```

To convert back as datastore, use the syntax:

```
Ids.setfullstate(lblob)
```

### *Moving Reports to the Web*

In two-tier environments, when the user select a report, the data is retrieved and displayed in the datawindow immediately, irrespective of the size of the report. For example, a 100-page item stock ledger for a financial period can easily be retrieved and displayed in the client window. But after being transformed into HTML, such a huge amount of data takes more time to display in the client browser. Also, navigating so much information in the web browser is quite cumbersome.

Therefore, you need to find alternative ways to show the report. One easy solution is to generate the report and keep it on the server, and then notify the user that the report is ready and let them download the file to his/her local machine. The same report can be downloaded by multiple users when required, thus avoiding the multiple retrieve of such huge data from the database.

### **Conclusion**

Finally, remember that you can minimize migration time using any of several available third-party tools to check your existing objects for your feasibility study, as well as commercially available frameworks. However, no tools are available that you can feed the two-tier application into and get the migrated application as output! Instead, good analysis of the best approach will make your migration a success. ■