PowerBuilder Journal

10 PowerBuilder Pitfalls

First the good news: PowerBuilder is a great tool; in fact you can't accidentally do much wrong. This strength is based on a number of reasons. The following is list of why I think PowerBuilder is so great, but you might like to add one or two more items to it:

- PowerScript is simple; it's easy to learn because of its clearly laid out grammar.
- PowerScript is a strongly typed language; many problems simply don't arise because of the compiler telling you about an error.
- PowerScript is easy to read. PowerBuilder itself takes care of the tedious task of correct indentation. Thus PowerBuilder programs look pretty much the same in terms of "layout" in all PowerBuilder shops around the world.
- There is no pointer system. You cannot access addresses of objects and do nasty things with them.
- PowerBuilder does all the low-level memory management for you. There's no need to allocate and dealloacate memory for strings or arrays, as it's done for you. When using objects, garbage collection cleans up after you.
- No dangling pointers. All references to a single object become invalid whenever the the object is destroyed and can be checked by using IsValid().
- Values of simple data types are initialized - numeric values with zero, Booleans with FALSE, etc.

Those elements are part of the strength of PowerBuilder (plus the DataWindow, of course). However, there are a few pitfalls in PowerBuilder programming that seem to frequently occur. This article will show you some of those issues. I have collected them over time in my function as technical lead where I do a lot of code review and "programmers first-level support" as I repeatedly come across the same problems. For some of the problems, I'll propose a workaround to make your programs safer. Let's start.

1. Find With Wrong Parameters

What's wrong with the following piece of code?

```
l_max = ds.RowCount()
l_index = ds.Find('x>5', 1, l_max)
DO WHILE l_index > 0
   of_WorkOn(l_index)
   l_index = ds.Find('x>5', l_index + 1, l_max)
LOOP
```

Answer: It hides a potential endless loop - Find() can search backwards! If the value in column x is greater than 5 in the last row of the DataStore, l_index will eventually become l_max. The Find() will read:

```
l_index = ds.Find('x>5', l_max + 1, l_max)
```

PowerBuilder recognizes that l_max + 1 is greater than l_max and will start to search backwards. Of course, it will find the row l_max that fits the expression. The perfect endless loop is done.

HOW TO AVOID PITFALL 1:
This behavior is documented and thus won't change. What can we do about it? There are two workarounds:

- Always code the Find finishing not at RowCount(), but at RowCount() + 1.
- Implement a function Find in your DataWindow or DataStore ancestor class that only takes two arguments: the expression and the starting row. Within the function use the solution stated above.

2. Incorrect Boolean DataWindow Expressions

This pitfall is really a major problem if programmers are not aware of it, so be warned. PowerBuilder supports two language grammars: PowerScript and DataWindow Expression syntax. Many functions known from PowerScript are available within DataWindow expressions as well, so it's easy for us to code in each of them. But beware, there are some subtle and very important differences.

Whats's wrong with the following expression?

```
NOT IsNull(x) AND x > 100
```

Nothing really. At least if you use it within PowerScript, PowerScript will evaluate it to

```
(NOT IsNull(x)) AND (x > 100)
```

But take care: if you use the same expression within a DataWindow (for Find, Filter, etc.), PowerBuilder will evaluate it to:

```
NOT (IsNull(x) AND (x > 100))
```

therefore evaluating it to FALSE, because a value cannot be NULL and greater than 100 at the same time.

What is the reason for that?

DataWindow expressions have a different operator precedence for logical operators compared to PowerScript (and just about any other computer language I know). Usually NOT has precedence over AND, and AND has precedence over OR. But within DataWindow expressions, AND and OR have precedence over NOT. AND and OR will be evaluated in the order of how they occur.

This is very weird, but still is expected and documented behavior. Simply start PowerBuilder help and search for "operators:precedence", and you'll find the two diverging styles of logical operator precedence within PowerBuilder.

HOW TO AVOID PITFILL 2:
The only way you can get around this problem is to use brackets to tell PB explicitly what you mean. For the example above you'll need to write:

```
(NOT IsNull(x)) AND (x > 100)
```

3.Missing Message Boxes

Sometimes we use MessageBoxes for "quick and dirty" debugging. Once in a while a programmer tells me that PowerBuilder is not showing message boxes anymore. Take a look at the next code snippet; what could be wrong here?

```
s_data = ds.GetItemString(1, 'data')
MessageBox('data is', s_data)
```

The reason that PowerBuilder doesn't show the MessageBox is simple: s_data is null! And PowerBuilder does with MessageBox just about the same thing it does with all other functions that are being called with nulls as arguments: it does not execute the function but returns null.

HOW TO AVOID PITFALL 3:

The workaround for that is simple, but involves some work on your side: you need to code your own version of function MessageBox (for instance as a global function). You can't override PowerBuilder built-in functions without losing the ability to call them, so you need to give the function a new name, say MsgBox. The problem is that there is a wealth of overloaded versions of MessageBox functions, so you need to implement quite a few. Within each of the calls, check the arguments for nulls.

4. Not Fully Regenerated Source Code

Sometimes people in the Sybase newsgroups complain that PowerBuilder is slow and not stable. More often than not, the reason for that is the source code is not thoroughly regenerated. This has been especially true for the early versions of PB 7, where a change within the source of a function changed the order of the function list in the source code, which in turn led to incorrect functions being called if you did not regenerate all depending (i.e., calling) classes.

HOW TO AVOID PITFALL 4:
The workaround for this is simple: let computers do the dirty work for you. Do a nightly regeneration of your sources either with OrcaScript or any of the third-party tools that focus on that kind of task.

5. Passing Objects per Reference

This is not really a pitfall but rather an aesthetic issue: many developers apparently still believe that they need to pass objects "by reference" in order to be able to change their instance variables. This is wrong, PowerBuilder does not pass the object but only the object reference per reference. The only time you'll need this is when you change the object reference in the called function (for example, because you create or re-create it).

HOW TO AVOID PITFALL 5:
That's an easy one: simply use pass by value for objects.

6. New DataWindow Columns Are Not Updatable by Default

A PowerBuilder classic - I guess just about every programmer tripped over this behavior at least once in his or her career. Whenever you add a column to a table, you need to add it to the DataWindow result set in order to retrieve it from the database. That's the simple part. But, the newly added columns are not updatable by default. PowerBuilder used to give those columns a tab order of 0, so you couldn't edit the column. However, with more recent builds, newly added columns get the highest tab order, so you are able to edit the columns right away. The bad part: those columns won't be saved until you manually add them to the list using the "update properties" dialog.

HOW TO AVOID PITFALL 6:
Always remember to edit the update properties when you add a column.

7. Using ClipBoard in a DataWindow

Is the function ClipBoard not working for you? It might be because you use it within a DataWindow.

Sometimes the clipboard is used to store interim values. Using ClipBoard(s_data) within DataWindow source code is not successful, because the function ClipBoard is implemented within the class DataWindow with different semantics and thus hides the global function ClipBoard. The result is that the data you want to put in the clipboard simply isn't there.

HOW TO AVOID PITFALL 7:
When trying to access the clipboard in DataWindow sources, explicitly use the global operator :: and

therefore write ::ClipBoard(s_data).

8. SetItemStatus Needs Intermediate Step

This is also a PowerBuilder classic but is already well known: there are some item statuses that can't be set directly; you need a second SetItemStatus to achieve the goal. One example: changing the column status from "new" to "notModified" simply doesn't work. You need to set it to "dataModified" first, then you can change it to "notModified". Some others settings are allowed, but don't work as expected; changing from "newModified" to "notModified" will change the state to "'new".

HOW TO AVOID PITFALL 8:
Whenever you implement SetItemStatus, check with the PB online help for the "SetItemStatus method" if the combination is to be successful. Do an intermediate step if necessary.

9. No Runtime Error for Incorrect SetItems or SetFilters

If you issue an incorrect GetItemX on a DataWindow, PowerBuilder will raise an error if you used a wrong row, a wrong column name, or a wrong data type for the column. Unfortunately, there will be no runtime errors for incorrect SetItem commands. You need to check the return code of the function yourself. The same holds true for SetFilter. I believe these two functions to be some of the most important ones for PB scripts as they can influence the flow of execution and the stored data.

HOW TO AVOID PITFALL 9:
In your ancestor classes for DataStore and DataWindow, override all SetItems and SetFilters and raise an exception if there is an error. You will make your programs much more robust.

10. No Data After Retrieve

Retrieving data in a datastore doesn't always mean that you have the read rows available in the primary buffer. While any experienced PowerBuilder programmer will know this, it is a common source of problems for newbies. PowerBuilder applies the filter expression defined in the DataWindow painter after it has read the data. So it may happen that you read loads of data and the return value of Retrieve is zero! Why is that? The answer is simple: the return value of Retrieve is the number of rows in the primary buffer after the Retrieve, not the number of rows retrieved. The two numbers (rows retrieved versus rows in primary buffer) might be different in two cases:

1. There is a filter expression defined that filters some data. PowerBuilder applies the filter after having read the rows. You will find those rows in the filter buffer of your DataWindow/Datastore.
2. You returned 2 in RetrieveStart, thus not emptying the buffers before the read. If there had been rows before the Retrieve, the return value of Retrieve will be the sum of existing rows and newly read rows.

HOW TO AVOID PITFALL 10:
Just be aware of the fact. You might want to remove the built-in filter and do a SetFilter() and Filter in source code (which is much more readable anyway).

Summary

PowerBuilder has only a few gotchas you need to know in order to write reliable software. The 10 pitfalls mentioned above are my personal list of items to be aware of. Keeping those in mind when programming or - even better - taking care of them in your base classes will help you write a robust application.

Do you know of any other issues I didn't mention here? Please drop me a brief note - if I get enough

feedback, I'll compile those issues into another *PBDJ* article.

www.romu.com