Functions and Events

When to use which?

WRITTEN BY
ROLAND MÜHLBERGER

Sometimes it simply doesn't matter whether you use a function or an event, just pick one and stick to it

This article will shed some light on the differences between the two ways of implementing scripts in PowerBuilder: functions and events. I will discuss the issues in detail, giving background information on each of them. Then you'll be able to decide for yourself when to use which.

In the old days of PowerBuilder, functions and events were a different kind of beast. But as PowerBuilder evolved, functions and events became more and more similar. Now many programmers aren't sure what the differences are, and when to use which.

In the Sybase PowerBuilder newsgroups (see news://forums.sybase.com) you'll find a wealth of information, but I didn't find a thorough and up-to-date posting on the issue. So in this article I will tell you the similarities and the differences of functions and events. By the way, I used PowerBuilder 10 for my comparison.

Prelude: Correct Terms

In this article, the following terms from the object-oriented domain will be used:

Method: used as a general term for function or event.

Overloading: implementing a method in a class that already has a method with such a name, the difference lies in the argument types. No inheritance is involved. Overloading in PowerBuilder is useful for quite a few things:

- Mimicking optional arguments to functions: Other languages can define default values for arguments in the definition of the function itself. If you omit those arguments, the system will use the default values. Because PowerBuilder doesn't (yet!) have this functionality, we can mimic it by the using overloading. Simply implement a function with its required parameters. After that, implement another function with at least one less argument (beginning on the right-hand side, of course). Then, in the new functions, call the first function with the missing arguments set to default values.
- Support for different datatypes: you want to define a function that can either take an integer or a string as input. You can give them the same name, just use integer as the argument type for the first, and string for the argument type of the second function. At runtime, PowerBuilder will find the correct function for you.

A good example for overloading can be found in the class systemfunctions: MessageBox. You will find both of the issues above implemented there, optional arguments (leave out all but the first two arguments) as well as different data types (either use a string or an integer as the second argument). Picture 1 gives you a snapshot of the PowerBuilder browser.

Overriding: implementing a method in a derived class using an identical set of arguments (names as well as datatypes) found in the base class. It needs at least one base and a derived class to be implemented. Okay, this is nothing new to you. You do it every day when programming in PowerBuilder. Simply doubleclick on an event in the event list or a function in the function list and, voila, – the overriding is done. Picture 2 will show you the difference between overloading and overriding.

PowerBuilder also has its own terms:

Extending: implementing an event in such a way that the code of the base class is executed before the code of the current class. In deep class hierarchies this means that the code of the root class is executed first, then the code of the class derived from it, then the code of the class derived from that class…until you reach your own code, which executes last. You can switch this behaviour by toggling the "Extend Ancestor Script" in an event's context menu. Unfortunately there's no way to see whether an event is extended or not just by looking at the event script painter. You need to rightclick to the script and check whether "Extend Ancestor Script" is checked or not. (For those of you who are ISUG members there's an enhancement request to show that information in the script painter. Please go to @@@todo to vote for it.) Extending used to have the drawback that one couldn't access the return value of the base event. But PowerBuilder introduced a metavariable, ancestorReturnValue. You don't need to declare it, it's already available if you extend an event. AncestorReturnValue always has the datatype returned from the event.

Similarities

As I said before, functions and events are very similar (this is one reason why it's hard to decide when to use which). These things do have functions and events in common:

- Functions and Events can be POSTED by using the keyword POST
- The function or event of a direct base class can be called by using super::
- The function or event of an arbitrary super class can by called by using Classname::
- According to the PowerBuilder online help there's no performance difference between functions and events
- Functions and events can be overridden easily by doubleclicking the function/event list.
- Functions and events can be implemented using the same script painter. I mention this because up to PB 6, you used a different approach for each. Ancestor code can be seen by using the rightmost dropdown in the script painter

Differences

So now for the interesting part: what are

AUTHOR BIO
Roland Mühlberger works as a PowerBuilder class librarian and software engineer for the Austrian company ecosys. He also runs his owns business (ROMU Software) as an independent consultant. His special interest, besides mountain climbing, is programming tools. He's the author of SmartPaste, a tool for documenting PB source code.

the differences? Some of them are rather academic, but can be very important, especially to people writing class libraries.

ACCESS LEVELS

You can only define the access level for functions: it can be either "private." which means you can only call it from within the same class, or "protected," which means you can call it from within the same or inherited classes, or "public," which means you can also call the function from outside the object.

Events are implicitly public, there's no way to make them private or protected.

OVERLOADING

Only functions can be overloaded. Just look into any system class. PowerBuilder uses this ability in most of them. Events can't be overloaded. If you try, PowerBuilder will say "Duplicate event name."

OVERRIDING

Both functions and events can be overridden. But whereas new events are implicitly extended, functions aren't. Here you need to call the base class function explicitly.

DYNAMIC CALLS

When you use DYNAMIC to call a method, functions and events behave differently. Functions will produce a runtime error; events will fail silently. You will even have a chance to find out whether the event really existed, because PowerBuilder will set returning any value to NULL if it doesn't find it.

Using exception handling you can catch the runtime error that PowerBuilder throws ("dynamic function not found") when a function called dynamically isn't found.

TRIGGEREVENT

Another kind of dynamic call is using the TriggerEvent. You can build an event name during runtime and use the name (a string variable) to call an event.

This is an interesting feature in PB and the only thing close to "function variables" that PowerBuilder provides. The good thing about it is that the event doesn't need to exist at runtime.

The downside is you can't provide any parameters to the called event or get the result (apart from using any global means for the exchange, for instance, the message object). All the arguments you defined will be NULL using this syntax and the returned value is simply lost in cyberspace. There's no such possibility with functions.

POSSIBILITIES FOR EMBEDDED OBJECTS

When you embed an object in another (for instance, putting a control on a window or adding an object to the nonvisual object list), thus implicitly creating an embedded class, you can only edit its events. You can override/extend them or even add events. PowerBuilder gives you no way to override or add functions to those embedded classes. This is really a GUI deficiency because the tests I ran showed that PowerBuilder's syntax would allow add/override functions for embedded classes.

A major drawback is that you can't view the ancestor code for functions in embedded classes. But, as I said, this is only a matter of the script painter not letting us. The PowerScript parser can work with functions in embedded classes. Who knows, maybe we'll get some way to view and implement functions for embedded classes as well.

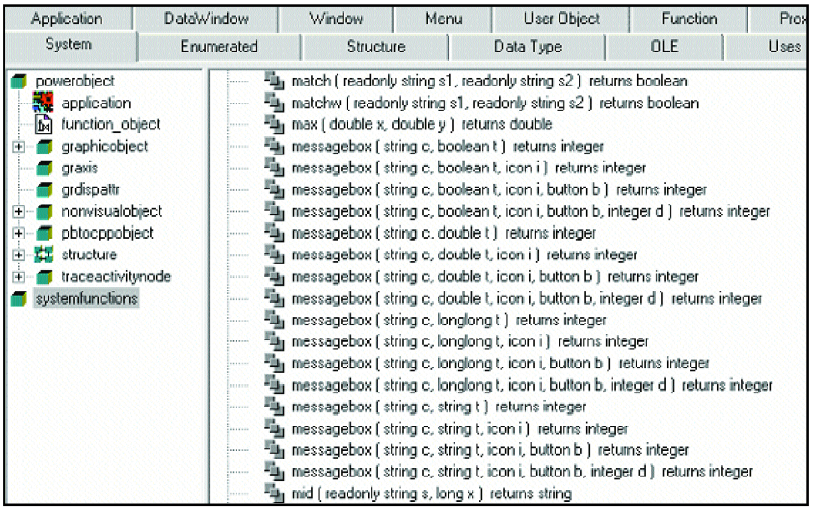

FIGURE 1 |

EXCEPTION HANDLING

Built-in events (like open, clicked, etc.) can't throw a checked exception (an object of a class derived from the system class Exception). You can only use unchecked exceptions (all objects of a class derived from RunTimeError). The reason is that checked exceptions must be defined in the event interface, and the event interface for built-in events is already defined by PowerBuilder. Checked exceptions need to be defined as part of the method interface (and therefore force the caller to deal with them) whereas an unchecked exception doesn't have need that.

THE NEED FOR RETURN STATEMENT

In built-in events, there's no syntactical need to issue a RETURN statement. PowerBuilder will simply return 0 in that case (which means "continue" in the Windows API sense. Returning 1 would mean "stop processing, I've already handled the call.")

User-defined events and functions need to implement at least one RETURN statement when a return type is declared.

MESSAGING SYSTEM

Events can have message IDs. By selecting a message ID, you tell the system to call this event whenever the message with the selected message ID occurs in your application. PowerBuilder already predefines many events (the clicked event on CommandButtons), but you can map any event of the event ID dropdown to your own event (for example, pbm_other).

This system is only available for events and not functions. Actually, this was the main difference between functions and events in the early stages of PowerBuilder, because there was no way to create an event without having an event ID.

CROSS-APPLICATION METHOD CALLS

If you have to send a message to a PowerBuilder program from outside, the only way to do it is to use the operating system's own messaging system. There's a range of user-defined message IDs that you can use (starting at 1024). In PowerBuilder you map those messages to events with custom event IDs. Pbm_custom01 maps to 1024, pbm_custom02

maps to 1025, and so on. You are limited in the arguments you can use. There are only two predefined arguments – lParam and wParam.

Calling events from outside works for all objects that have a Windows handle (windows, windows controls, ...) because the calling application needs to know this handle as well to make the call. There's no way to call a function from outside an application.

Conclusion

Now that we have ticked off the similarities and differences, it's time to decide. In fact, there's no definitive decision to be made between functions and events. It lies in how you do your programming. But I can give you some advice to help you make your decision:

- When programming a class library or any class that you think will be used or derived from quite a lot, try to be "academic" and think about access levels. Don't make the methods public that shouldn't be called from outside. Then it's clear that you will use functions for them instead of events.
- Use the "template functions" and "hotspot events" system. That means that code that shouldn't be overridden (the template methods) sits in functions, and code that should be overridden (the hot spots) sits in events. The reason is that you can define the access level for the functions (the ones that are called from outside), and when you inherit, you usually only need to implement events, making the list of spots to override shorter.
- If you use exception handling with checked exceptions, you are limited to using user-defined events and functions. Quite a big discussion has been going on about checked and unchecked exceptions in the newsgroups of other programming languages. I personally dislike the checked ones because they introduce implementation details to the interface of the class. Whenever I change the implementation and I'm forced to throw other exceptions, I need to change the function interface as well. So all programmers using my class are forced to change their code as well, which is bad. So for me, this issue doesn't exist. I can throw exceptions in events as well as functions, because I simply use the ones inherited from RunTimeError instead of Exception.
- Use functions for overloading.
- Use events to give a programmer a chance to override the logic in embedded classes. This is particularly true for base classes of controls (your DataWindow ancestor, for example).

You won't use functions or events exclusively. In fact, there are lots of times when it simply doesn't matter whether you use a function or an event to implement your source. In those cases, you should decide and stick to one or the other. But sometimes it's good to think ahead and choose wisely. This list of differences will help you do that.

office@romu.com