Subject: Re: Widget events - any other than WhenAction ?
Posted by xrysf03 on Sat, 16 Nov 2019 21:35:24 GMT
View Forum Message <> Reply to Message

Thanks a lot :)

So that was not at all complicated, after you have told me *how* :) You've done all the hard work for me... At first I read through all the files that you have included, only to find out that it really is very simple.

To add the class into my existing form's header file was a no-brainer. Copy and paste 6 lines from your example header file. And yes I pasted it just above #define LAYOUTFILE .

```
// Any derived-class to be used in the layout editor
// must be declared before the layout file.
struct MyEditDouble : EditDouble {
 Event<> WhenGotFocus;
 Event<> WhenLostFocus;
 void GotFocus()  override { WhenGotFocus();  };
 void LostFocus() override { WhenLostFocus(); };
};
```

I couldn't find MyEditDouble in the visual layout editor, neither as a new widget, nor if I just tried to change the class of my existing two edit boxes... so I ended up just editing the .lay file in VIM - just renamed the class of my two edit boxes. Reopened the layout in TheIDE, yes the two Edit boxes are now weird grey, but they keep their original dimensions. And, they can be referred to from main.cpp, and the two custom Event<>s appear in the autocompletion list. Good!
What I found a little hard was to find the right syntax of how to actually hook the event = how to register my own callback. All the docs mention just the default WhenAction and the <<=THISBACK() operator/macro. Sorry to say that but I was lucky with trial and error, and this is what works for me:
Edit1.WhenLostFocus << THISBACK(my_handler);
I.e. I need to use the "shift left" operator, point it at the particular Event by name, and the rest is as usual: THISBACK(handler_name).

After that, it does what I need :) Yippee!

Actually I had to look up "override" - didn't know what it was good for... I learned the basics of C++ just after Y2K :) So now I know...

Also, looking at your code, a few further pieces of the puzzle snapped in - or rather, I found answers to a couple questions that hadn't even occured to me that I should've asked in the first place :) Specifically: when dealing with events and callbacks, the uses and relationships for inherited member methods vs. the callback as a tool. They're different from each other and they probably need to be combined, to achieve what I was after.
So again for people who may come after me and stumbe upon this thread, let me rant on for a bit

about the way I understand it:

Those two member functions: GotFocus() and LostFocus(): those get inherited down the class hierarchy. From "class Ctrl" down to pretty much every visual widget class. All the heirs (inheritance children and grandchildren) of UPP::Ctrl can override those two methods. Note that each child and grandchild can override the method for its own use, and the author of that overridden method may ask himself, if he should also call the immediate ancestor class'es version of the method (if needed, this has to be done explicitly, as only constructors and destructors are cascaded (semi)automatically). And, all the instances of a particular child/grandchild widget class, will get the same version of the overridden method - inside the method, the "current" or "respective" widget instance is obviously known by the "this" pointer. So: these member methods follow standard inherintance, including its possible pitfalls and the "this" pointer.

Now what about the U++ GUI callbacks. Those are the things you register using the THISBACK macro. Note that a widget instance uses the callback to contact a particular instance of the "compositional parent" layout window (GUI form in Delphi slang). Note that callbacks get "primed"/hooked in the constructor of the "compositional parent" layout = in a member method of the layout class, in a context where "this" has a meaning, i.e. the code knows what the current instance of the layout object is. The callback is a link from a widget instance to its "compositional parent" layout instance. This means that, in contrast to the GetFocus() and SetFocus() methods, each instance of a particular widget class can have the same Event<> handled by a different callback, and delivered to a different instance of the "callback recipient" class.
This "instance to instance" relationship also means that there's probably quite a bit of template magic behind the Event<> thing and the THISBACK() "macro" :) Let me repeat the example:
Edit1.WhenLostFocus << THISBACK(my_handler);
The widget instance is fairly explicit - it's the object referred to by the Edit1 name (by value or by pointer). WhenLostFocus is the particular Event<> = itself an inheritable property in the widget class hierarchy. At the receiving end of the callback, my_handler() is a member method of the layout class. The recipient layout class name probably gets extracted from the method declaration/definition (is processed by the THISBACK macro behind the scenes) and the recipient layout instance is likely taken from "this" in the code scope where the THISBACK(my_handler) is called. I mean to say that the example one-liner using THISBACK() is anything but innocent :)

In contrast to member method inheritance and overriding, if the Event<>s got declared all the way up in UPP::Ctrl, and inherited across several layers of hierarchy, they could not be liberally "overriden" and used differently at different layers. The callback can only point to a single "recipient instance". Umm... I'm probably over-analyzing it... I mean to say that the GetFocus()/SetFocus() methods vs. the WhenSomething Event<>s are two different tools, serving different purposes, and can be conveniently combined together if needed - as instrumentally demonstrated by you, Oblivion :)

If I didn't have the Event<> and THISBACK syntactic candy, I'd probably have to point to the "compositional parent GUI layout instance" using a combination of an object instance pointer and a function pointer from the widget instance. It would mean a bit of extra custom coding.

Also... It probably makes good sense to declare the Event<>s on my own and call them from local overrides of GotFocus()/LostFocus() at my own level of abstraction. If I declared those Event<>s all the way up in "class Ctrl", it wouldn't make good sense to call them from

Ctrl::GotFocus()/Ctrl::LostFocus(), as these methods can get overridden by any "heirs", and the heirs would have to remember to call the upstream versions of GotFocus()/LostFocus()... it would be a mess. So if Mirek decided to include those two Event<>s in "class Ctrl", it would make good sense to ping those callbacks from the places that Ctrl::GotFocus()/Ctrl::LostFocus() get called *from* :) I haven't found any such places in the code using a simple "grep", but that just means I'm not capable enough, and I suspect that there can be multiple such places = it would need further thought. Also, as GotFocus()/LostFocus() already exist and are used (overridden), the question is, whether to ping Event<> WhenGotFocus / WhenLostFocus before or after GotFocus() and LostFocus() get called :)
Apologies if I let my imagination race too far ahead.
I just wanted to mention some design issues that I'd expect if someone decided to follow this path.

I don't think I need any further help with this, I'll flag the subject "solved".
Thanks for all your help and attention :)

Frank