Subject: [solved] Widget events - any other than WhenAction ?
Posted by xrysf03 on Fri, 15 Nov 2019 19:56:16 GMT
View Forum Message <> Reply to Message

Dear everyone,

once again I seem to be struggling with an old borlandese delphian habit of my childhood...
Do I understand correctly that the Edit widgets (or any U++ widget, for that matter?) have just a single "hookable event", called the WhenAction ? From Delphi I seem to recall various events, namely OnEnter and OnExit. Why I long for them? I need to take some action when the user is finished editing the value in an Edit box. EditDouble actually. I do not want to call that same action after every keystroke in the EditDouble box, which is how WhenAction works.
I'm expecting an answer along the lines of "the are no other actions, and it's by design, that's just how UPP/CtrlCore is..."

Any comments welcome

Frank Rysanek

Subject: Re: Widget events - any other than WhenAction ?
Posted by Oblivion on Fri, 15 Nov 2019 20:17:28 GMT
View Forum Message <> Reply to Message

Hello Frank,

Quote:
Do I understand correctly that the Edit widgets (or any U++ widget, for that matter?) have just a single "hookable event", called the WhenAction ?

No, not really. WhenAction is the common event for all (well, almost all) widgets. Edit widgets, for example, have their own events too.

EditDouble (or other Editxxx variants) has:

Callback1<Bar&> WhenBar

This callback represents the context menu of EditField. The default is StdBar.

Callback WhenEnter

This callback is invoked if user presses Enter key while in EditField. If not empty, EditField also consumes Enter key (so that it is not passed up in Ctrl hierarchy). Default is empty.

Callback1<WString&> WhenPasteFilter

This callback is invoked when Paste operation is performed and can be used to alter the text to be pasted. Default is no change to the text.

Callback1<Vector<Highlight>&> WhenHighlight

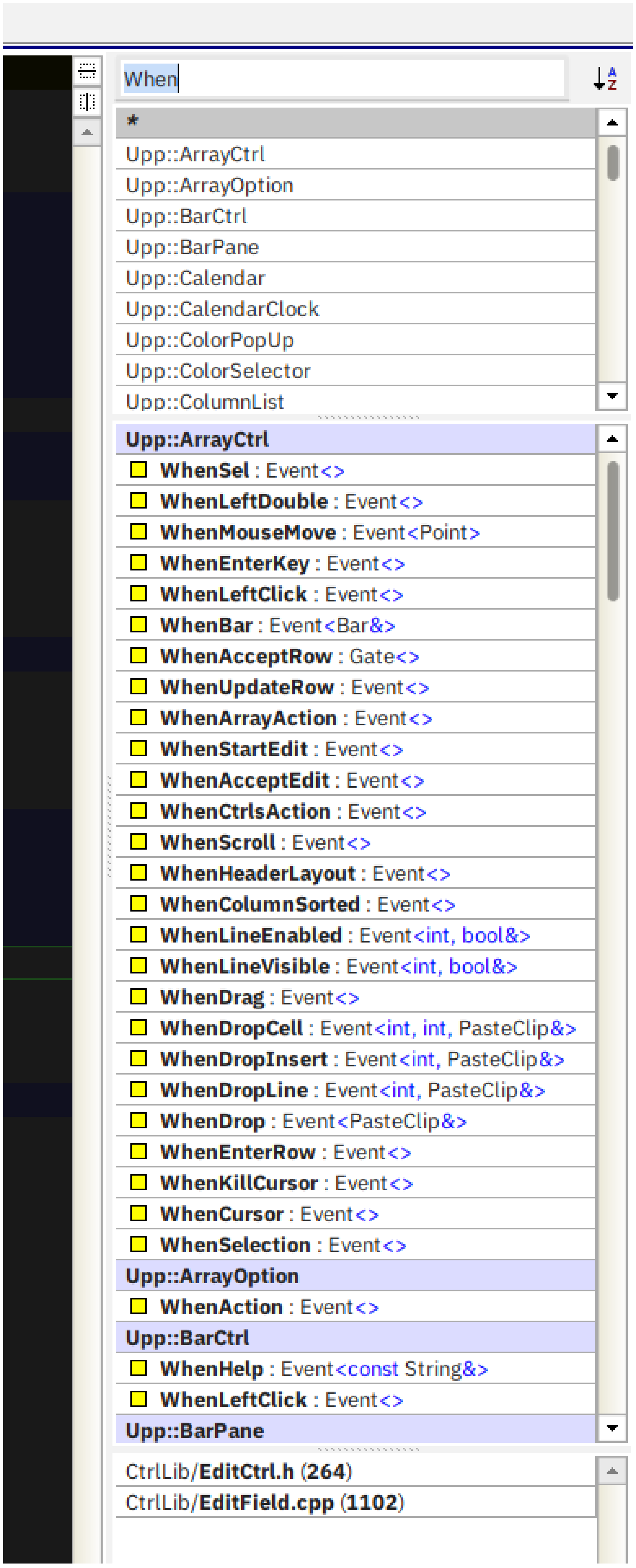
Called by default implementation of HighlightText. Provides a chance to change the text color and background for individual characters.

Api docs page usually list them, but there might be some missing. IF you are using TheIDE, just type "When" into the navigator and it'll list all the events that are part of the classes in your project:

Best regards,
Oblivion

File Attachments

1) TheIde-Navigator.png, downloaded 976 times


When

*

Upp::ArrayCtrl
Upp::ArrayOption
Upp::BarCtrl
Upp::BarPane
Upp::Calendar
Upp::CalendarClock
Upp::ColorPopUp
Upp::ColorSelector
Upp::ColumnList

Upp::ArrayCtrl
- **WhenSel** : Event<>
- **WhenLeftDouble** : Event<>
- **WhenMouseMove** : Event<Point>
- **WhenEnterKey** : Event<>
- **WhenLeftClick** : Event<>
- **WhenBar** : Event<Bar&>
- **WhenAcceptRow** : Gate<>
- **WhenUpdateRow** : Event<>
- **WhenArrayAction** : Event<>
- **WhenStartEdit** : Event<>
- **WhenAcceptEdit** : Event<>
- **WhenCtrlsAction** : Event<>
- **WhenScroll** : Event<>
- **WhenHeaderLayout** : Event<>
- **WhenColumnSorted** : Event<>
- **WhenLineEnabled** : Event<int, bool&>
- **WhenLineVisible** : Event<int, bool&>
- **WhenDrag** : Event<>
- **WhenDropCell** : Event<int, int, PasteClip&>
- **WhenDropInsert** : Event<int, PasteClip&>
- **WhenDropLine** : Event<int, PasteClip&>
- **WhenDrop** : Event<PasteClip&>
- **WhenEnterRow** : Event<>
- **WhenKillCursor** : Event<>
- **WhenCursor** : Event<>
- **WhenSelection** : Event<>

Upp::ArrayOption
- **WhenAction** : Event<>

Upp::BarCtrl
- **WhenHelp** : Event<const String&>
- **WhenLeftClick** : Event<>

Upp::BarPane

CtrlLib/**EditCtrl.h** (**264**)
CtrlLib/**EditField.cpp** (**1102**)

Subject: Re: Widget events - any other than WhenAction ?
Posted by xrysf03 on Sat, 16 Nov 2019 07:27:13 GMT
View Forum Message <> Reply to Message

@Oblivion okay, thanks for your patient explanation. I've learned a new trick

At face value, I can see that there are no ready "When callbacks" for a widget getting or losing focus.

Taking one step back, and using the excellent "API search" function, I have noticed the following methods, defined in Ctrl.cpp:

```
void Ctrl::GotFocus()                {}
void Ctrl::LostFocus()               {}
```

They're actually documented here. Means to me that this is not... say... a Win32-specific feature, or a Borland VCL feature, but actually there is rudimentary systemic support for this in the U++ GUI layer.

And obviously I'm wondering... to get the desired "callback on an Edit losing focus", I'd need to inherit from say EditDouble and overload its LostFocus() - but next, I'd have to embed that in a layout, so first I'd need to create a visual component, along the lines of MyEditDouble.

I haven't advanced that far yet, maybe I should, but at the same time I'm wondering... how much trouble would it be, instead, to create systemwide WhenGotFocus/WhenLostFocus actions by adding some stuff to Ctrl::GotFocus()/LostFocus()   This is not necessarily a feature request, let me say that I'm just wondering aloud at this stage... Or, would it be against some "design virtues" of U++ to add those? Would it break existing code? Make existing code run slower?

Subject: Re: Widget events - any other than WhenAction ?
Posted by xrysf03 on Sat, 16 Nov 2019 08:24:55 GMT
View Forum Message <> Reply to Message

Thinking again, those two events are probably no definitive salvation to my envisaged "GUI input cross-checking logic". I was wondering if I could run an extra thread to do it, but in the end I should probably just cross-check (and auto-complete) the inputs in the next explicit step in the "user workflow". For the most part it is a non-issue.

Subject: Re: Widget events - any other than WhenAction ?
Posted by Oblivion on Sat, 16 Nov 2019 08:46:35 GMT
View Forum Message <> Reply to Message

Hello Frank,

Quote:
They're actually documented here. Means to me that this is not... say... a Win32-specific feature, or a Borland VCL feature, but actually there is rudimentary systemic support for this in the U++ GUI layer.

They aren't win32 specific. Platform-specific stuff have a notice stating that on their api docs. They work on Linux and MacOS too.

Quote:
I haven't advanced that far yet, maybe I should, but at the same time I'm wondering... how much trouble would it be, instead, to create systemwide WhenGotFocus/WhenLostFocus actions by adding some stuff to Ctrl::GotFocus()/LostFocus()

Well, I personally don't see any problem, but Mirek (the main developer of U++) should answer this question anyway, as he knows the rationale behind this decision.

Quote:
And obviously I'm wondering... to get the desired "callback on an Edit losing focus", I'd need to inherit from say EditDouble and overload its LostFocus() - but next, I'd have to embed that in a layout, so first I'd need to create a visual component, along the lines of MyEditDouble

Not necessarily. Any Ctrl-derived widget can be used in the layout editor. But only the most used components have a complete representation. Ony if you need full visual representation you'll need to write a usc (layout entry) file. For example, you can add a rudimentary visual representation of your widget. But by default it will just show a blank rectangle area with a tag (name of the class) attached on it. You can use it to set its layout on window. This is a two steps process:

1 Declare your derived class in header file, just above the layouts file.
2 Add it to your layout using the layout editor.

I've attached an example U++ code to the message.

File Attachments

1) LayoutTest.zip, downloaded 338 times

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

Subject: Re: Widget events - any other than WhenAction ?
Posted by Oblivion on Sat, 16 Nov 2019 22:16:16 GMT
View Forum Message <> Reply to Message

Hello Frank, I'm glad you find a solution to your problem.

I'd like to point out some things that may also help others, and save you from taking unnecessary steps:

1) You can edit layout files in TheIde. A layout file is nothing more than a C++ file which relies on method chaining and templates, i.e it is not "really" a macro file (of course it has macros in it but that's more for the sake of readability. Those macros expand to C++ templates.) When you open the Layout editor in TheIde, go to Edit menu and select "Edit as text" item. You can use same manu (this time the menu item will be "Edit using designer") to return to visual editing.

2) Callback macros are pre-C+11 stuff, kept for compatibility, and they are deprecated. You can use THISFN macro instead. (It does not care about the number of arguments passed to the callback, i.e there are no THISFN1, THISFN2, etc. macros to deal with. However, THISFN macro is simply a lamdba -an in-place function- wrapper. Therefore you can directly use --and I highly recommend using-- C++11 lambda functions, as they are very flexible (they can even capture local variables etc.)

E.g.

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

struct App : public TopWindow {
 EditString edit;
 App()
 {
  SetRect(0, 0, 400, 200);
  Sizeable().Zoomable().Add(edit.HSizePos().VCenterPos());
  edit.WhenEnter = [=] { PromptOK((String) ~edit); };
 }
};

GUI_APP_MAIN
{
 App().Run();
}
```

3) MyEditDouble is "not" in the layout editor, because it doesn't have a .usc (visual representation) file at the moment. However, if you right click on the layout editor area, you'll see a menu item called "User Class" in the context menu. IF you select that item it will add a blank box, and let you set the class type (MyEditDouble) and its instance name on the left pane.

I hope this helps. I suggest you check reference examples in the reference section. They really help a lot. IME, U++ is very flexible, once you're familiar with its style.

Best regards,
Oblivion

Subject: Re: Widget events - any other than WhenAction ?
Posted by peterh on Fri, 02 Sep 2022 08:21:15 GMT
View Forum Message <> Reply to Message

Oblivion wrote on Sat, 16 November 2019 23:16
3) MyEditDouble is "not" in the layout editor, because it doesn't have a .usc (visual representation) file at the moment. However, if you right click on the layout editor area, you'll see a menu item called "User Class" in the context menu. IF you select that item it will add a blank box, and let you set the class type (MyEditDouble) and its instance name on the left pane.

I found this, more by accident and trial and error and studying the sources:

An *.usc file can inherit from CtrlLib.usc.
If you add an *.usc file, eg. Ctrllib.usc to your project eg.

```
ctrl MyEditDouble {
 >EditDouble;
}
```

Then the derived control can be seen and edited in the layout editor, it inherits the usc file from its parent.
I find this very useful, it should be in the tutorial or/and documentation.
This makes it much easier for newbies.

Subject: Re: Widget events - any other than WhenAction ?
Posted by peterh on Sun, 04 Sep 2022 19:28:05 GMT
View Forum Message <> Reply to Message

There is another - more elegant - solution:

Code your derived Ctrl as a template:
(this is experimental stuff I made for learning purposes, it works as expected, but does nothing meaningful)

```
template <typename TCtrl>
struct MyCtrl : public TCtrl {
 int count = 0;
 MyCtrl()//:TCtrl() // Default constructor wird automatisch aufgerufen, deshalb ist ...:MyCtrl:TCtrl()
(hier) nicht notwendig, schadet aber auch nicht.
 {
  WhenEnter << []{Cout() << " Enter ";};
 }

 virtual void Paint(Draw& w) override {
  //w.DrawRect(GetSize(), White());
  TCtrl::Paint(w);
  w.DrawText(2, 2, AsString(count));
 }
};
```

Then you can derive arbitrary Controls from it, and the compiler will bark if there is a problem.

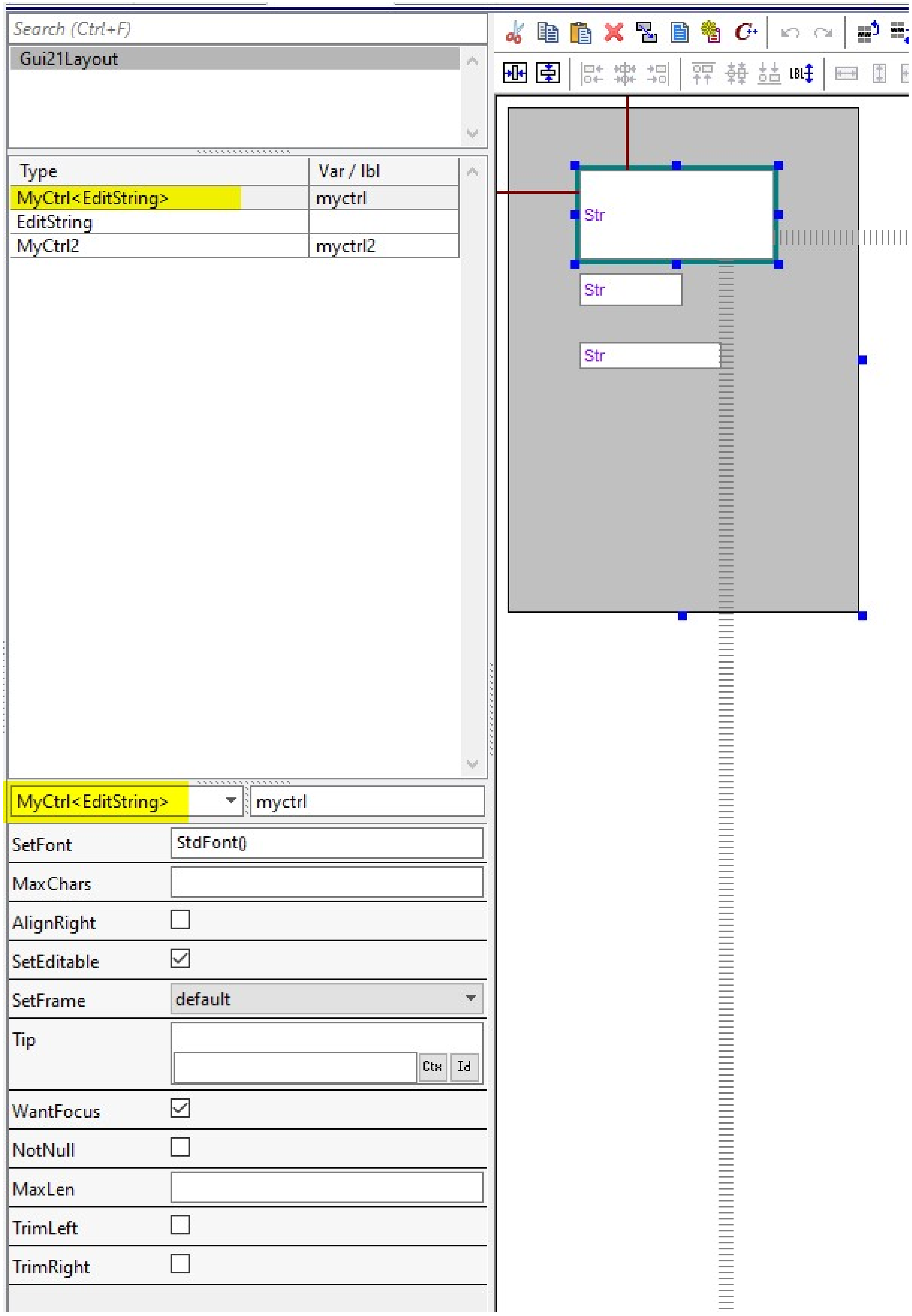
Then define your control in the layout editor like this:

I think this should be in the tutorial.
It might be "cold coffee" to regular users, but will get new users started.

An *.usc file is not required in this case.
In most cases I want to expand or modify the behavior of an existing control.
I think this is a very common problem that new users will encounter and possibly will give up on it.
Then this comes in handy.

File Attachments

1) Layout2.jpg, downloaded 581 times


Search (Ctrl+F)

Gui21Layout

Type	Var / lbl
MyCtrl<EditString>	myctrl
EditString	
MyCtrl2	myctrl2

MyCtrl<EditString> myctrl

SetFont	StdFont()
MaxChars	
AlignRight	☐
SetEditable	✓
SetFrame	default
Tip	
WantFocus	✓
NotNull	☐
MaxLen	
TrimLeft	☐
TrimRight	☐