Subject: Possible improvements to U++ callbacks
Posted by cas_ on Thu, 18 Sep 2008 17:34:59 GMT
View Forum Message <> Reply to Message

I know there are probably more urgent things to deal with (like switching to some modern graphics backend, documentation improvements and perhaps more decent debugging support on Linux), but anyway I've decided to share some thoughts.

U++ supports callbacks through Callback* family of templates. They are simple, effective and sufficient in most situations. However, there is very little support for connecting multiple functions to single callback object - only chaining is supported, as far as I know. Unfortunately, this approach makes it hard to disconnect a previously connected callback. What if you delete an object, which is pointed by a callback placed in some chain? Some existing signal/slot libraries (like libsigc++) give you even possibility to handle such situations automatically (in sigc++ it's enough to derive your class from sigc::trackable).

What do you think about improving U++ callbacks in this way? Would you find it useful? It should be possible to extend current implementation without losing backward compatibility.

Subject: Re: Possible improvements to U++ callbacks
Posted by mirek on Fri, 19 Sep 2008 06:16:11 GMT
View Forum Message <> Reply to Message

cas_ wrote on Thu, 18 September 2008 13:34
What if you delete an object, which is pointed by a callback placed in some chain? Some existing signal/slot libraries (like libsigc++) give you even possibility to handle such situations automatically (in sigc++ it's enough to derive your class from sigc::trackable).

Actually, to some degree, this is possible in U++ too - see pteback family.

Quote:
What do you think about improving U++ callbacks in this way? Would you find it useful? It should be possible to extend current implementation without losing backward compatibility.

Well, I think this is a good idea, however I see very limited benefit. In fact, even the chaining is sort of redundant.

In reality, I do not remember a usecase where I would have been sorry of not having what you suggest. And simplicity is a virtue of its own

Mirek

Subject: Re: Possible improvements to U++ callbacks
Posted by jlfranks on Mon, 23 Mar 2009 20:43:36 GMT
View Forum Message <> Reply to Message

Your missing Use Case:

We are using U++ callbacks as multi-cast delegates in a
publish-subscribe event message scheme.

Up until now, the subscribers were static, i.e., setup when
objects were instanced at application start-up.

We are starting on Modbus mapping of data <--> modbus
registers using multiple tree controls and callbacks
to do the heavy lifting of data I/O. This mapping is
dynamic at run-time and can be changed by the operator.

This means that the delegate must have the capability of
removing one-of-n callback functions (Subscriber) from
the callback list.

I'm not sure how to do that with PTEBACK().

Can you provide me with more insight on this?

--jlf

Subject: Re: Possible improvements to U++ callbacks
Posted by mirek on Tue, 24 Mar 2009 11:10:10 GMT
View Forum Message <> Reply to Message

jlfranks wrote on Mon, 23 March 2009 16:43Your missing Use Case:

We are using U++ callbacks as multi-cast delegates in a
publish-subscribe event message scheme.

Up until now, the subscribers were static, i.e., setup when
objects were instanced at application start-up.

We are starting on Modbus mapping of data <--> modbus
registers using multiple tree controls and callbacks
to do the heavy lifting of data I/O. This mapping is
dynamic at run-time and can be changed by the operator.

This means that the delegate must have the capability of
removing one-of-n callback functions (Subscriber) from

the callback list.

I'm not sure how to do that with PTEBACK().

Can you provide me with more insight on this?

--jlf

Well, PTEBACK probably cannot really solve this issue, because it would leave Callback record intact, only made it "inactive" after destruction of pointeee.

I think that in order to correctly solve this issue, you would have to use some sort Vector<Callback> and unsubscribe command that does remove from this.

There are many possible approaches to the problem..

Mirek
