Subject: hierarchical tree data structure & binding to TreeCtrl
Posted by solareclectic on Thu, 01 Nov 2007 07:35:28 GMT
View Forum Message <> Reply to Message

I've a few questions/ideas I'd like to bounce of those more experienced with U++...

I've more than a few applications to write that all want to have a hierarchical tree-like data structure as their primary structure. I see that TreeCtrl has the Key/Value (which, handily, take a Value) and all the requisite add/remove/count/find child/parent functions that one should expect in a Tree. At first glance, I thought I'd use whatever TreeCtrl inherits from- expecting to find a "Tree", to which Tree"Ctrl" then added all the GUI aspects; but, alas, I find it all in TreeCtrl.

Now, it just so happens that these tree data structures are, in fact, meant to be directly manipulated by the user; and it would be the simplest matter to just use the TreeCtrl directly as the data structure and add/remove my various other objects into the Key/Value "Values," using .Is() to determine the class of object that had been stuck in to a Node previously, and ValueTo<MyObject> to get them out and call their functions.

Perhaps there is an easier way, still, to call functions of their common base class with out doing the ValueTo<MyClass> bit?
I had tried making my common base class "Rich," but wasn't entirely successful, and wasn't sure it was actually necessarily any easier.

Now, to the real question... I'm looking for an obvious way to keep that Model/View/Controller separation that we've all been compelled to accept as gospel. I find no compelling reason (for my applications) or examples of TreeCtrl keeping the MVC separation with an datastructure internal to the application (the filesystem is an external datasource providing hierarchy, an other sample populate it with assorted labels - but no "linking" to internal datastructure). Can I just use TreeCtrl as my datastructure, as it provides all the functionality that I require, including the GUI stuff? Or if MVC separation must be maintained, is there a harm in using two TreeCtrls - one for the View, and one for the Model that just never gets shown?

And, a bonus question... (though for different widgets)
Since I'm taking shots a MVC separation anyway, I'm finding a similar situation with U++ nifty EditField derived widgets, which have all of the range/notNull checking, etc. that I want on to protect my Model object fields from getting set incorrectly, from the GUI or otherwise. If I want to keep, say, an Int datamember in one of my objects AND I want the user to manipulate it, why wouldn't I declare that datamember as EditSpin, instead of Int? Then when I want to present it to the user for editing, I simply have it draw itself?
Are there major performance/memory issues with this behavior?

Now I do, honestly, understand and respect the value of MVC separation, but since U++ has so conveniently packaged obviously useful behaviors together, I thought I should ask you all to see if I'm just making things harder on myself and overlooking the obvious "U++ Way" of doing things.

Thanks for any insight any of you can provide regarding best use of U++. Regardless of this answer, U++ has restored my interest in programming for FUN again.