Subject: DOCU: Ctrl design concepts
Posted by kohait00 on Mon, 30 Aug 2010 08:26:05 GMT
View Forum Message <> Reply to Message

hey guys,

i'm trying to summ up things that a have learned so far cencerning development of own Ctrl's, where to pay attention and on what to take care. help this one to go to manual / documentation..i think it's rather important, since own widgets are a quite common demand and everyone tries to find its own way and pace with it. but outlining some design concepts (from the makers, not from me would ease the task drastically, and example code sometimes is hard to understand from the beginning. but knowing some things before (and beeing able to recognize them in actual code later) helps a lot. so here is some of the things.. please add the rest or suggest on what more need to go there

////////

Ctrl Design Concepts

When developing own Ctrl's one often is 'reinventing' the wheel, because there are common patterns to do in your Ctrl that Ultimate++ is probably already providing in Ctrl base class. Thus, knowing the base class and some of its key design concepts can make your life esier and the development of your Ctrl faster, while focusing more on the problem than the methods. As always, the best reference for Ctrl is its source code, which is quite large, thats why I try to sumerize some of the usefull things you can already use. In any case, look at the virtual functions in Ctrl to see what is meant to be implemented or used by you. I won't cover LogPos related things here, it is covered in another docu.

Generally, a Ctrl in Upp is helper to visualize some kind of data. The data, though, is not static, and can be changed from GUI (point and click) perspective or from API perspective, using manipulating functions. The difference is, that GUI interaction should modify the internal data (or state) of Ctrl AND notify application somehow about change, but modifying it using API should NOT generate change notifications. This is a very important design rule that keeps you away from a lot of head ache from recursive invokations when modifying Ctrl's in API.

Ctrl Tree

Ultimate++ uses a linked list for all the child Ctrl's that have been Add()ed to it, partaking of its drawing space. The Ctrl does NOT own its children, but simply references them (Ptr<Ctrl>). They should be owned by your appliction, somewhere in a U++ container, i.e. Array<Label> or they are already made members of your app when using Layout files. If a Ctrl is added to another, it is ensured to be properly removed from its previous parent, thus a Ctrl cant be part of 2 trees.

GetData / SetData

Most Ctrl's you will ever create will only need one single value to visualize or represent. This is true for EditFields, Buttons, Labels, etc. To be able to Get / Set this single value into/from the Ctrl,

Upp uses it's own 'polymorphic' Value class (see another docu), which enables the Ctrl's to receive and handle intrinsic datayptes internally through one single interface, reliefing you from the convertion pain. That's why exsists GetData / SetData pair. it is the main door into your Ctrl. Even more complex Ctrl's like TreCtrl use it to provide the currently selected index. Think of your Ctrl, which information it could provide as general through this interface. it makes implicit usage easy, also in terms of notification (see next)

WhenAction Callback

To notify upper layers of some changes, your Ctrl can use internally (or the user externally) the Action() function, which will call WhenCallback. and provide the feedback This is the Callback that can be set using '<<= THISBACK()' approach, so using it for your own Ctrl is preferable, since it leads to Upp conform short syntaxes. Be carefull to only call Action() inside your code upon graphical user interaction. When modifiing your Ctrl from API, it should generate no Action(). More or diverse notifications can be provided in your controls using other global Callbacks (or even Callback1<yourtype> or more), if needed. Use the WhenSomething name convention to reflect Event behaviour.

Updated(), SetModify(), ResetModify(), ClearModify(), IsModified()

Often, the control needs to process or calculate other things based on the change of some data inside the control (like maybe some results, cached values or the like, NOT graphical helper data, this is done using Layout() which is invoked when resizing or opening the Ctrl). Use the Updated() virtual function to realize this, because it can be triggered from 'outside' using the Update() function. It also SetModified()'s your Ctrl, so you can check for it. Often, when data is changed, Ctrl needs to be updated somehow calculating its things and then the user needs to be notified. UpdateAction() does this in one step, calling both. If graphical data needs change as well, UpdateActionRefresh() is the chain to go, which will invoke an additional Paint(). ClearModify() acts recursively on all children too.

Accept / Reject

lorem ipsum

Layout

lerem ipsum

////////////