Subject: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 16:18:41 GMT
View Forum Message <> Reply to Message

Newbie proposes changes to the library? Well, this is another proof of how well U++ is designed and implemented.

But will this break the "Everything belongs somewhere" principle? I don't think so. Owned Ctrls will be taken care of by their parents who own them. So they belong to their parents. As it's clearly defined and easily determineable, the principle is actually perfectly confirmed.

1. Advantage of allowing parent Ctrl to own certain child.
a. sometimes it's more natural to allocate Ctrl's on the heap;
b. it can relieve the programmer(library user) from keeping track of uninterested objects only to properly destroy them afterwards;
c. if used with discretion, it can reduce the memory footprint of generated program. I will give some examples if you don't believe me.

2. Will the proposed change break any existing code?
No. If not impossible, it's very very unprobable that the changes will affect any existing codes that was not aware of it

3. How signicant are the changes in the current libary codes to allow for children ownship?
It's minimal. I cannot handle it if it's too big as my knowledge with U++ is still very limited. About 6-10 function has been changed, another flag (1 bit) is added which will not increase memory requirement of Ctrl objects.

Here is a list of the changes (may not be complete)
A. In Ctrl.h
A.1

```
#ifdef PLATFORM_X11
 bool        ignoretakefocus:1;
#endif
 bool   owned:1; // <--This line

 static  Ptr<Ctrl> eventCtrl;
```

AND

```
 // proposed changed, open door for libary developer
 // but still concealed from library user
 //
 bool   IsOwned()const{ return owned; }
 void   SetOwned(bool v=true){ owned=v; }
```

AND

```
// flag: 0 - not to be owned, eg for Ctrl alloc on stack or otherwise maintained
//       1 - yes, own the child, will be responsible for its destruction
//       2 or other values - the owned flag has been properly set, just use it.
void            AddChild(Ctrl *child,int flag=2);
void            AddChild(Ctrl *child, Ctrl *insafter,int flag=2);
void            AddChildBefore(Ctrl *child, Ctrl *insbefore, int flag=2);
// seel RemoveChild0 for explanation of detachOnly parameter
//
void           RemoveChild(Ctrl *child, bool detachOnly=false);
```

AND

```
// flag 2 means to leave the owned flag untouched( already properly set)
void    Add(Ctrl& ctrl, int flag=2)  { AddChild(&ctrl, flag); }
```

To be continued. I am running out of my time.
