Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 22:02:15 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 18 March 2011 19:32
Sorry, but you would have hard time to convince me to go this way.

I guess the 'U++ legit' solution to this problem is to use Array.

```
struct Parent : ParentCtrl {
   Array<Ctrl> child;

   template<class T> T& Create() { T& x = child.Create<T>(); Add(x); return x; }
};
```

I believe that the moment you encourage using manual heap usage, the whole idea collapses. You start adding flags to what is owned and what is not everywhere and end in regular C/C++ heap mess, moments later wishing that you had garbage collector to deal with it...

Mirek

I knew it's going to be a tough job But there is a distinction between allowing and encouraging. Like smoking and drinking are still allowed in most countries but that doesn't mean their governments are encouraging the practices.

I have no doubt that the problem can be solved elegantly in the current U++ framework without introducing anything extra. TheIDE, which has a graphical designer allowing it to create children at users' requests is a good proof of such capability.

That being said, the solution you proposed has certain drawbacks. That container can only host Ctrl itself, or with some effort, we can make it to host objects of class derived from Ctrl without introducing any member variables. Beyond that, you need to provide a different container for almost every type of Ctrl derivatives that it will ever host. This fact alone will make Array<Ctrl> impractical.

Array<Ctrl*> is the next best alternative. Unfortunately Array<Ctrl*> will not delete the pointers it hold upon its destruction. Either we enclose it in a class to ensure pointed Ctrls are deleted or, we can store some smart pointer objects instead.

What if the situation requires many child be removed and added, ie., in a really dynamic situation? We probably need to use Index.

What do we gain from introducing an Array<Ctrl*> members or its Index version to keep track of dynamically created children at the cost of more memory space and CPU cycles? Virtually nothing.

Compare the two cases:

1. Owned children allowed. Simply Add(CtrlObject.Owned()); and RemoveChild(CtrlObject);
Programmer understand there is a contract between him/her and the libaray, when he/she set the object as Owned, he/she surrenders the ownership to the containing object, just like he/she will sometimes ask a smart pointer object to take care of new'd objects

```
// when add
this->Add((new MySpeicalCtrl())->SizePosz(...).Owned());

// if the child is to destroy with *this, nothing
// further needs to be done,
// however, if user interactions require it to be removed
//
Ctrl * p = find_the_child_some_how();
this->RemoveChild(p);

// or more flexible, the program decide to take back ownership
Ctrl * p = find_the_child_some_how();
p->Owned(false);
this->RemoveChild(p);

// now p become freestanding, and it's the programmer's
// responsibility to delete it when no longer needed,
// or, find a new dad for it.


MySpecialCtrlContainer.Remove(actrl);
```
[/code]
2. Current status with no owning children support.

```
// when add
MySpecialCtrl * p=MySpecialCtrlContainer.Create();

// Setting up p properties
this->Add(*p);

// when remove
this->Remove(actrl);
MySpecialCtrlContainer.Remove(actrl);
```