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.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Wed, 16 Mar 2011 16:26:47 GMT
View Forum Message <> Reply to Message

i'm currently facing the same problem, to easily create controls in a tree, without keeping them around somwhere else. the use case is to have the Ctrl's edit and manipulate some central Values of the app, where more than one, and different Ctrls can 'point' to the same bit of Value information and the user can create and release those controls. i suppose you have something similar.

AFAIK, the thing of owning Ctrls would make some more changes nessecary. and i dont know if mirek is willing to take the risk of braking some API handling, and actually a rule of thumb, which you outlined.

maybe think of some additional mechanism..

anyway, this question is a design aspect decision.
maybe it is to be considered..

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 16:29:02 GMT
View Forum Message <> Reply to Message

AND

```
// when a child is removed, it should be delete'd if its @param owned
```

```
// flag is true. Sometimes we may want to keep it alive and transfer the
// ownership to some other Ctrl, in this case, we should call the following
// function with @param detachOnly set to true
//
void    RemoveChild0(Ctrl *q, bool detachOnly=false);
```

And corresponding changes in the CtrlChild.cpp file. I have attached these files. The changes have been commented, so it should stand out. I am on a library computer, session time is limited.

File Attachments

1) CtrlCore.h, downloaded 665 times

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 16:30:08 GMT
View Forum Message <> Reply to Message

And CtrlChild.cpp

File Attachments

1) CtrlChild.cpp, downloaded 744 times

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 16:39:22 GMT
View Forum Message <> Reply to Message

kohait00:

I have implemented the requested changes. Just replace existing files with my attached version (make backup before doing that) and you should be able to use it immediately.

Here is some sample how you can use it.

```
struct MyButton : Moveable<MyButton>, Button
{
   // note: alwasy default to false becasue
   // existing U++ user are used to this behaviour
   //
   MyButton(bool toBeOwned=false)
   {
      SetOwned(toBeOwned);
   }
```

```
    virtual ~MyButton()
    {
        DUMP("Yes, I am properly destructed!!!");
    }
};

class App : public TopWindow
{
public:
    App()
    {
        Button * p=new MyButton(true);
        Button * p2=new MyButton();

        p->SetLabel("Button 1").TopSizeZ(2,20).LeftSize(2, 60);
        p2->SetLabel("Button 2").TopSizeZ(40,20).LeftSize(2, 60);
        Add(*p); // p will be destructed because we SetOwned in
                 // its constructor;
        // or (*this) << p;

        Add(*p2, 1);
    }
};
```

Please try it and let me know there is any problem. I am not very sure if I changed anything in Ctrl.cpp.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Wed, 16 Mar 2011 17:01:32 GMT
View Forum Message <> Reply to Message

havent tried it yet, but more things that are involved, and need to be checked is at least this:

adding an owned control, which is part of a tree already, to another tree needs to traslate the ownership to the new tree..

i'll try it soon..
but first, i want to hear from mirek if this is worth it at all

---

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 17:10:18 GMT

AFAIK, that has already been taken care of by AddChild0 or something like that.

```
void  Ctrl::AddChild(Ctrl *q, Ctrl *p, int flag)
{
 GuiLock __;
 ASSERT(q);

 LLOG("Add " << UPP::Name(q) << " to: " << Name());
 if(p == q) return;
 bool updaterect = true;
 bool owned;
 switch(flag)
 {
  default: // note, all value of flag other than 0&1 in this branch
   owned=q->owned;
   break;
  case 0:
  case 1:
   q->owned=flag==1;
 }

 ///if(dynamic_cast<q

 if(q->parent) {
  ASSERT(!q->inframe);


//************************
// following couple of lines takes care of the issues you raised.
//************************
  if(q->parent == this) {
   RemoveChild0(q, true); // detach only
   updaterect = false;
  }
  else
   q->parent->RemoveChild(q, true); // detach only }
 q->parent = this;
 if(p) {
  ASSERT(p->parent == this);
  q->prev = p;
  q->next = p->next;
  if(p == lastchild)
   lastchild = q;
  else
   p->next->prev = q;
  p->next = q;
```

```
 }
 else
  if(firstchild) {
   q->prev = NULL;
   q->next = firstchild;
   firstchild->prev = q;
   firstchild = q;
  }
  else {
   ASSERT(lastchild == NULL);
   firstchild = lastchild = q;
   q->prev = q->next = NULL;
  }
 q->CancelModeDeep();
 if(updaterect)
  q->UpdateRect();
 ChildAdded(q);
 q->ParentChange();
 if(updaterect && GetTopCtrl()->IsOpen())
  q->StateH(OPEN);
 if(dynamic_cast<DHCtrl *>(q))
  SyncDHCtrl();
}
```

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 17:29:02 GMT
View Forum Message <> Reply to Message

That's why I praise the design and implementation of U++.

Even though I don't know U++ really well, I can be pretty much sure my solution works or will work with minor refinement.

It can be proved by reasoning:

1. Every child of a Ctrl is itself a Ctrl2. However deep the derivation chain, the virtual destructor of Ctrl will alway be called and there will be only one version of the destructor which I touched.
3. In the destructor of Ctrl, it repeatedly remove each of its children by calling RemoveChild;
4. U++ design guaranteed each Ctrl will have but 1 or 0 parent.
So however ownership is changed during a Ctrl's life time, it will have 1 parent if it was ever assigned a parent. When its parent is destructed, and its owned flag is set to true, it's guaranteed to be destruct by our implementation of RemoveChild family functions.

Subject: Re: Proposed change to U++ to allow owning children.

Posted by Lance on Wed, 16 Mar 2011 17:35:51 GMT
View Forum Message <> Reply to Message

I just successfully built TheIDE with the revised library. So it should not affect exsiting code.

Only questions left, will there be some code in the U++ libary that will make use of the bit I assigned to the new memebe, i.e. owned, magically change it, or will some Ctrl derived class decides to remove its children by itself and without make use of Ctrl::RemoveChild family of functions?

If answer to these questions are no, they my way should work (maybe with some more modifications).

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 17:43:26 GMT
View Forum Message <> Reply to Message

Yet another file that's changed (by insert a single line)

Ctrl.cpp

File Attachments
1) Ctrl.cpp, downloaded 499 times

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 16 Mar 2011 18:24:45 GMT
View Forum Message <> Reply to Message

Here is the line that's been changed in Ctrl.cpp in case you don't want don't the whole file.

```
Ctrl::Ctrl() {
 GuiLock __;
 LLOG("Ctrl::Ctrl");
 destroying = false;
 owned = false; // <---This line initialized owned to make
                // which is necessary!
 parent = prev = next = firstchild = lastchild = NULL;
```

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 01:13:11 GMT
View Forum Message <> Reply to Message

Not related to this topic.

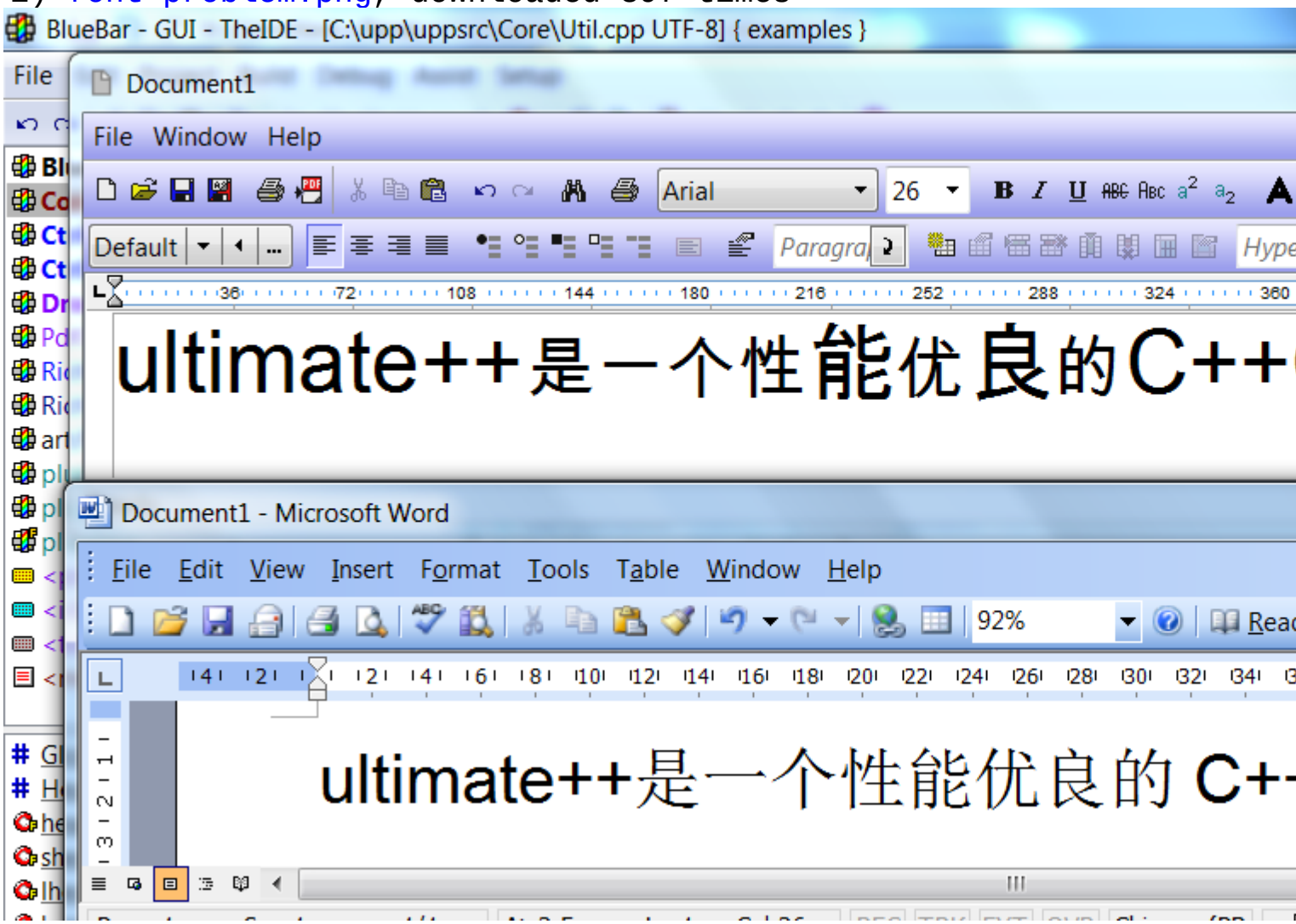
One little thing that worries me, U++'s support for Chinese Fonts need more work, or may be it's because I don't know how to add more fonts yet. See attached for a sample.

The sample text is:

I guess U++ misinterpreted certain parameters in the non-western font information. Neither GTK nor QT has the same problem.

File Attachments

1) font problem.png, downloaded 897 times



Subject: Re: Proposed change to U++ to allow owning children.

Posted by kohait00 on Fri, 18 Mar 2011 10:01:35 GMT
View Forum Message <> Reply to Message

the flag handling should be better IMHO.

flag = 2 is quite bad, leaving things untouched...
it should determine itself, when the control is owned by someone else, to take over the responsibility

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 13:39:38 GMT
View Forum Message <> Reply to Message

Hi Kohait00:

Internally, the owned flag has but 2 possible values: true or false. When a Ctrl is removed from its parent by RemoveChild family of functions, there are 3 scenarios:
1. it's owned and should be automatically deleted by the parent control thereby it's destructor is called and the memory in the heap to house the Ctrl is also freed;

2. it's not owned. Basically this is the behaviour before the owned flag is introduced. So business as usual. Exisiting code should not feel any difference. They don't know anything about the owned flag, don't rely on it and will not be bothered by it.

3. it's owned and should be leave alone. Techinically, it should be named DetachChild. But again, we don't want to any unexpected impact to users who don't know the existence of this newly installed capability. So I add an additionaly parameter detachOnly[to the RemoveChild family of functions, with carefully chosen default value so as no to break existing code. As you can see, in the AddChild function implementation I highlightly in a previous post, RemoveChild is called with detachOnly set to true so that it's not destroyed as our realy intention is just to change a dad for it.

Also, it could be useful to library user. User programs can detach a child and take appropriate actions. The only thing to remember is once a child is detached, the one who detach it has its ownership. It can either free it, or entrust it to someone else. The principle of "Everything belongs somewhere" is consistently confirmed.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 14:21:01 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Fri, 18 March 2011 11:01the flag handling should be better IMHO.

flag = 2 is quite bad, leaving things untouched...
it should determine itself, when the control is owned by someone else, to take over the responsibility

Sorry I misinterpreted you. Your point is valid.

Yes it's possible to revise flag from tribool to bool for all the AddChild*(and Add) functions. The implementation of the function will need to be changed accordingly.

Another option is to promote SetOwned(bool owned=false) from protected to public, and keep AddChild prototypes untouched.

```
...
public:
...
    // query owned flag
    bool IsOwned()const{ return owned; }
    Ctrl& Owned(bool b=false){ owned=b; return *this; }

...
};
```

And the way it's used will be changed to something like this

```
class App: public TopWindow{
...

App()
{
  Button * p  = new Button();
  Button * p2 = new Button();

  this->Add(p.Owned());

  this->Add(p2); // Error, p2 is not owned.

  // fixed p2;
  p2.Owned();

};
```

Anyway, the detail can be polished and fixed to satisfy majority of users as long as there is no big holes in the design.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 14:40:52 GMT
View Forum Message <> Reply to Message

Kohait00, Thank for the input.

The second option I proposed is way better. That way:
1. AddChild*/Add, RemoveChild* etc's prototype don't need to be changed.
2. RemoveChild's implementation do not need to be changed.
3. Only Ctrl default constructor and RemoveChild* need to be changed slightly.

I will implement it and upload changed version.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 15:22:08 GMT
View Forum Message <> Reply to Message

Now we have a much cleaner solution thanks to Kohait00's input.

Brief list of changes to the current libary code:

In CtrlCore.h

Added to Ctrl class
    bool owned : 1;
And 2 public members

```
 bool   IsOwned()const{ return owned; }
 Ctrl&   Owned(bool v=true){ owned=v; return *this; }
```

In Ctrl.cpp, initialized owned flag to false

```
Ctrl::Ctrl() {
 GuiLock __;
 LLOG("Ctrl::Ctrl");
 destroying = false;
 owned = false;  // <-----This line is added
 parent = prev = next = firstchild = lastchild = NULL;
 top = NULL;
 exitcode = 0;
 frame.Add().frame = &NullFrame();
 enabled = visible = wantfocus = initfocus = true;
 editable = true;
```

```
// GLX = false;
#ifdef PLATFORM_WIN32
 activex = false;
 isdhctrl = false;
#endif
 backpaint = IsCompositedGui() ? FULLBACKPAINT : TRANSPARENTBACKPAINT;
 inframe = false;
 ignoremouse = transparent = false;
 caretcx = caretcy = caretx = carety = 0;
 SetRect(Rect(0, 0, 0, 0));
 inloop = popup = isopen = false;
 modify = false;
 unicode = false;
 popupgrab = false;
 fullrefresh = false;
 akv = false;
 hasdhctrl = false;
}
```

And in CtrlChild.cpp

```
// @param: q , the child to be added
//         q,  an existing child to precede p
void  Ctrl::AddChild(Ctrl *q, Ctrl *p)
{
 GuiLock __;
 ASSERT(q);

 LLOG("Add " << UPP::Name(q) << " to: " << Name());
 if(p == q) return;
 bool updaterect = true;

 // remember and change
 bool owned=q->owned;
 q->Owned(false); // that way it's guarenteed not to be
    // accidently delete'd when possibly changing parents


 if(q->parent) {
  ASSERT(!q->inframe);
  if(q->parent == this) {
   RemoveChild0(q);
   updaterect = false;
  }
  else
   q->parent->RemoveChild(q);
```

```
 }
 q->parent = this;
 if(p) {
  ASSERT(p->parent == this);
  q->prev = p;
  q->next = p->next;
  if(p == lastchild)
   lastchild = q;
  else
   p->next->prev = q;
  p->next = q;
 }
 else
  if(firstchild) {
   q->prev = NULL;
   q->next = firstchild;
   firstchild->prev = q;
   firstchild = q;
  }
  else {
   ASSERT(lastchild == NULL);
   firstchild = lastchild = q;
   q->prev = q->next = NULL;
  }

 // succesfully added as children of *this, now
 // it's perfect time to restore saved owned flag
 q->Owned(owned);

 q->CancelModeDeep();
 if(updaterect)
  q->UpdateRect();
 ChildAdded(q);
 q->ParentChange();
 if(updaterect && GetTopCtrl()->IsOpen())
  q->StateH(OPEN);
 if(dynamic_cast<DHCtrl *>(q))
  SyncDHCtrl();
}

.....

void  Ctrl::RemoveChild0(Ctrl *q)
{
 GuiLock __;
 ChildRemoved(q);
 q->DoRemove();
 q->parent = NULL;
```

```
 if(q == firstchild)
  firstchild = firstchild->next;
 if(q == lastchild)
  lastchild = lastchild->prev;
 if(q->prev)
  q->prev->next = q->next;
 if(q->next)
  q->next->prev = q->prev;
 q->next = q->prev = NULL;

 if(dynamic_cast<DHCtrl *>(q))
  SyncDHCtrl();
 // code added to allowed owned child****
 if(q->owned)
  delete q;
 // end code added by Lance
}

void  Ctrl::RemoveChild(Ctrl *q)
{
 GuiLock __;
 if(q->parent != this) return;
 q->RefreshFrame();

 bool owned=q->IsOwned();
 q->Owned(false); // we still need it to be alive

 RemoveChild0(q);
 q->ParentChange();
 if(GetTopCtrl()->IsOpen())
  q->StateH(CLOSE);

 if( owned )
 {
  delete q; // this is why the new'd-only requirement.
 }
 // no need to restore q's owned flag, it's either destroyed or
 // its owned flag is correctly set
}
```

File Attachments

1) CtrlCore.rar, downloaded 490 times

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 15:34:42 GMT

And a sample program using this newly added capability.

```
#include <CtrlLib/CtrlLib.h>

// http://java.sun.com/docs/books/tutorial/uiswing/start/swingTour.html

using namespace Upp;

struct MyButton : Moveable<MyButton>, Button
{
public:
 // should always default to false to confirm with
 // existing U++ users habit
 //
 MyButton(bool toBeOwned=false)
 {
  if(toBeOwned)
   Owned();
 }
 virtual ~MyButton()
 {
  String msg=String().Cat()<<"From within MyButton's destructor, with label "<<GetLabel();

  DUMP(msg);
 }
};

struct ButtonApp : TopWindow {
 int    count;
 MyButton button;

 Label  label;

 void RefreshLabel()
 {
  label = Format("Number of button clicks %d", count);
 }
 void Click()
 {
  ++count;
  RefreshLabel();
 }

 typedef ButtonApp CLASSNAME;

 ButtonApp()
```

```
 {
  count = 0;
  //button=*new MyButton();
  Button * p = new MyButton(true);
  Button * p2 = new MyButton();

  button <<= THISBACK(Click);
  button.SetLabel("&I'm an Ultimate++ button!");
  Add(button.VCenterPos(20).HCenterPos(200));
  Add(label.BottomPos(0, 20).HCenterPos(200));
  label.SetAlign(ALIGN_CENTER);
  Sizeable().Zoomable();
  RefreshLabel();

  Add(p->SetLabel("&Another button").TopPosZ(10, 50).LeftPosZ(30, 200)); // becuase MyButton
is contructed with proper flag
  Add(p2->SetLabel("Heap Button 2").TopPosZ(10, 50).LeftPosZ(250,200).Owned());
 }
};

GUI_APP_MAIN
{
 ButtonApp().Run();
}
```

With output:

```
msg = From within MyButton's destructor, with label I'm an Ultimate++ button!
msg = From within MyButton's destructor, with label Another button
msg = From within MyButton's destructor, with label Heap Button 2
```

Subject: Re: Proposed change to U++ to allow owning children.
Posted by mirek on Fri, 18 Mar 2011 18:32:51 GMT
View Forum Message <> Reply to Message

Lance wrote on Wed, 16 March 2011 12:18Newbie 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.

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

Subject: Re: Proposed change to U++ to allow owning children.
Posted by mirek on Fri, 18 Mar 2011 18:35:12 GMT
View Forum Message <> Reply to Message

Lance wrote on Thu, 17 March 2011 21:13Not related to this topic.

One little thing that worries me, U++'s support for Chinese Fonts need more work, or may be it's because I don't know how to add more fonts yet. See attached for a sample.

The sample text is:

I guess U++ misinterpreted certain parameters in the non-western font information. Neither GTK nor QT has the same problem.

I guess you should repost this in separate thread and also provide information about the host-OS.

Mirek

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);
```

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

Lance wrote on Fri, 18 March 2011 18:02
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.

Totally untrue. You can put anything derived from Ctrl there without any compromises or efforts.

See

http://www.ultimatepp.org/reference$DynamicDlg$en-us.html

(and perhaps learn about U++ containers and Array flavor).

Mirek

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 22:34:35 GMT
View Forum Message <> Reply to Message

Thank you for the quick reply. The example really amazed me. I guess I mixed Array with Vector. I didn't look into the code, but I guess Array introduced another layer so that like link list/map, etc, it doesn't require objects be stored in adjacent memory locations.

In the link you give to me, only Labels are stored in Array<Label>. But I don't dispute you. I will do a test to put Ctrl objects of different size to an Array<Ctrl>.

If that's the case, the additional costs to maintain dynamic object is not that hefty. But it still didn't solve the problem I raised regarding use cases where frequent insertion and deletion is needed.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by mirek on Fri, 18 Mar 2011 22:40:42 GMT
View Forum Message <> Reply to Message

Lance wrote on Fri, 18 March 2011 18:34
If that's the case, the additional costs to maintain dynamic object is not that hefty. But it still didn't solve the problem I raised regarding use cases where frequent insertion and deletion is needed.

If you are concerned about Array::Remove/Insert costs, I believe that can easily work up to 10000 elements without issue.

If you get over that, you are dealing with some very special situation anyway and "owned" flag is not going to help you very much I believe.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Fri, 18 Mar 2011 23:02:37 GMT
View Forum Message <> Reply to Message

Thank you very much! I have no doubt with U++'s speed. That's one big aspect why it attracts me.

Very informative.

Since you are here, how do you like my way of revised Ctrl::Ctrl implementation?

```
Ctrl::Ctrl() {
 GuiLock __;
 LLOG("Ctrl::Ctrl");

    // a smarter way to implement this function
    // as we can see, most member variable to initialized
    // to 0, we can save a couple of cpu cycles by simply
    // zero out the part of object that are of POD type

    // Note Non-POD member variable frame, info, pos has been move to
    // follow POD members, with pos being the first non-pod member var.
    typedef int32 unit; // 4 should be deduced for flexibility
    unsigned size=((char*)&this->pos -(char*)this)/sizeof(unit);
    for(unsigned i=0; i<size; ++i)
     reinterpret_cast<unit*>(this)[i]=0;


 //destroying = false;
 //owned = false;
 //parent = prev = next = firstchild = lastchild = NULL;
 //top = NULL;
 //exitcode = 0;
 frame.Add().frame = &NullFrame();
 enabled = visible = wantfocus = initfocus = true;
 editable = true;
// GLX = false;
#ifdef PLATFORM_WIN32
 //activex = false;
 //isdhctrl = false;
#endif
 backpaint = IsCompositedGui() ? FULLBACKPAINT : TRANSPARENTBACKPAINT;
 //inframe = false;
 //ignoremouse = transparent = false;
 //caretcx = caretcy = caretx = carety = 0;
 //SetRect(Rect(0, 0, 0, 0));
 //inloop = popup = isopen = false;
 //modify = false;
 //unicode = false;
```

```
 //popupgrab = false;
 //fullrefresh = false;
 //akv = false;
 //hasdhctrl = false;
}
```

Subject: Re: Proposed change to U++ to allow owning children.
Posted by mirek on Fri, 18 Mar 2011 23:37:08 GMT
View Forum Message <> Reply to Message

Lance wrote on Fri, 18 March 2011 19:02Thank you very much! I have no doubt with U++'s speed. That's one big aspect why it attracts me.

Very informative.

Since you are here, how do you like my way of revised Ctrl::Ctrl implementation?

Slightly negative speed impact, saved about 100 bytes of .exe, dangerous w.r.t. code maintainance...

Mirek

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Sat, 19 Mar 2011 00:30:00 GMT
View Forum Message <> Reply to Message

Thank you for giving it a try.

I was surprised to know there was actually a speed penalty. Maybe that's because of the for loop. Some really smarter compiler will tranlsate the for loop to movsb, movsw, movsdw, etc, then the for loop will no longer pose a speed penalty.

I was thinking set bit fields will be slower. Maybe just simple put the bitfields in a union and the whole flags field to 0.

Yes, the code will be harder to read and maintain. If somebody accidently add a non-pod data member into the area that's suppose to be POD members, he/she may be surprised.

A similar occasion.
[code]

Then here in the Draw.h

```
class Font : AssignValueTypeNo<Font, FONT_V, Moveable<Font> >{
 union {
  int64 data;
  struct {
   word  face;
   word  flags;
   int16 height;
   int16 width;
  } v;
 };
...

 Font()  { data=0; } // that's the whole point why
                     // a union is introduced IMHO
```

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Sat, 19 Mar 2011 00:35:29 GMT
View Forum Message <> Reply to Message

Sorry I am wasting your time. It's not worthy going further.

I still hold the belief that allowing a child to be owned by its parent is not a bad idea. But as the penalty for not allowing it is so small , there is no chance I can persuade you. So I give up.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by mirek on Sat, 19 Mar 2011 09:59:38 GMT
View Forum Message <> Reply to Message

Lance wrote on Fri, 18 March 2011 20:30Thank you for giving it a try.

I was surprised to know there was actually a speed penalty. Maybe that's because of the for loop. Some really smarter compiler will tranlsate the for loop to movsb, movsw, movsdw, etc, then the for loop will no longer pose a speed penalty.

Nope, they are slightly slower than series of assignments.

Actually, smart compiler might unroll that loop into series of assignments anyway.

Quote:
I was thinking set bit fields will be slower. Maybe just simple put the bitfields in a union and the whole flags field to 0.

Now that would be smarter. Still, compiler are good, I would not be surprised if they replaced those assignements anyway.

Anyway, we are speaking here about negligible impacts on result (this area of code is definitely irrelevant optimization-wise).

But my primary complaint is that the loop is poor choice for quality of code, too error-prone.

Quote:
Then here in the Draw.h

```
class Font : AssignValueTypeNo<Font, FONT_V, Moveable<Font> >{
 union {
  int64 data;
  struct {
   word  face;
   word  flags;
   int16 height;
   int16 width;
  } v;
 };
...

 Font()  { data=0; } // that's the whole point why
                     // a union is introduced IMHO
```

Data is used to hash Font value and to compare it too.

Anyway, in any case there are 4 stable fields in union. Such situation is easily manageable.

Ctrl fields are much more numerous and change over time.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Sun, 20 Mar 2011 09:32:56 GMT
View Forum Message <> Reply to Message

i am dealing with the same issue, and will, if fortunate, provide a no-need-to-touch-upp solution soon..using a container interface and a template approach, so you can safely handle the stuff with Array<Ctrl>

just a hint, i myself where trying to convince mirek of things several times . his experience is hard to beat. upp demands/suggests to give up some long existing (and maybe errorprone) programming habbits. the benefit (especially when using upp) is clear code, simple model approach and a high degree of code readability (and thus maintainability). take your time to get familiar with upp and its way to handle things. you quite soon gonna love it. it saves you a lot of hassle (just to mention a few: no pointer hassle, memleaks almost never (everything belongs somewhere) and a sophisticated serialisation/marshalling mechanism. upp is well thought out, and the things mirek is defending are part of a long run design process which prooved to be reliable.

cheers

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Sun, 20 Mar 2011 13:07:29 GMT
View Forum Message <> Reply to Message

Thanks, kohait00! As mirek has demonstrated, it's not hard.

The only issue is if it's worthy. Paying the cost of an additional container (I didn't look into Array implementation, from its interface, it's probably an Vector of pointers pointing to individually allocated objects. The user would not feel a difference. But still there are memory cost & speed cost paid.) so that Upp libary users can do without manual new/delete, that's something I cannot appreciate at my current level or mindset.

It's really a philosophical/religious difference. Many ones believe pointers are evil, Java go as far as do away with it completely, but that doesn't make the small bunch us to leave C++ and embrace it.

C++ also encourages using its library and smart pointers to minimize the chances to use new or have to keep track of new'ed objects. But it certainly doesn't go as far as declaring new/delete a privilege of library developers only.

Anyway, this is really unimportant. As I have said, we stand no chance to convince mirek, who have achieved so much with the sticking to his philosophy, plus the cost is but minimal.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Sun, 20 Mar 2011 13:11:50 GMT
View Forum Message <> Reply to Message

BTW, I like the IDE facility (probably by overloading global new/delete) to detect memory leak in debug mode. It's very convenient and considerate.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Sun, 20 Mar 2011 13:28:08 GMT
View Forum Message <> Reply to Message

It just come to me that we can change the interface slightly to make it unnecessary/impossible for end users to use new directly.

Add a template member function to the Ctrl class

```
class Ctrl...
{
...
    bool owned : 1;
....
protected:
   bool IsOwned()const{ return owned; }
   Ctrl& Owned(bool v){ owned=v; return *this; }

public:
   template <typename ChildType>
   ChildType& AddOwned()
   {
      ChildType* p=new ChildType();
      p->Owned();
      (void)Add(p);
      return *p;
   }


   template <typename ChildType, typename T>
   ChildType& AddOwned(T& t)
   {
      ChildType* p=new ChildType(t);
      p->Owned();
      (void)Add(p);
      return *p;
   }

   // return ChildType so that user can further set its
   // properties.

   // The IsOwned and Owned functions will be demoted to protected
   // so that they are available only to libary developers
   //
   // whether a child is owned is a one time decision.
   // an owned child will not be able to be reverted to unowned
   // by end user (without derive from the Ctrl class or its
   // derivatives), but he/she is free to change parents for it
```

```
    //
    // Unless the end user uses some hackish practice,
    // library developers can be assured the owned flag
    // is reliable and predictable.
};
```

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Fri, 25 Mar 2011 12:26:11 GMT
View Forum Message <> Reply to Message

still interested in that topic?

i think having owned Ctrls just like that wont do much, since you have to know/reference the controls somehow/somewhere, you will end up traversing the child link tree, and doing dynamic_casts allover.

ultimate is heavily based on compile time connection of the application parts, means, if you have a Ctrl and place it somewehere, then you probably will hook up some static (not runtime specific) functionality to it, within your code.

a heap creation of Ctrls implys some sort of dynamic application behavior. for what the controls need to be fairly straight forward in handling, i.e. only Get/SetData interface unsing, only WhenAction Callback using, etc.. anything else is too specific.

i basicly have the same problem. that's why some time ago, i started the CtrlProp package in bazaar. a uniform api to query/control some properties of Controls, without actually knowing its type.

another project i am currently takling is the generation / modification of Controls in its position, a control factory so to say, which would be extended using *decorator* design pattern.
meanwhile see the CtrlPos package. which is part of it.

maybe thats sth you can use as well. see also FormEditor, which is another nice work, from someone else.

as soon as the ctrl factory is in a fitting shape i'll provide it in bazaar.

cheers

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Sun, 27 Mar 2011 13:33:57 GMT
View Forum Message <> Reply to Message

kohait00:

Sounds interesting. Please let me know when you do.

Thanks,
Lance

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Sun, 17 Apr 2011 15:25:02 GMT
View Forum Message <> Reply to Message

in my journey exposing core elements of upp to python i ran into this issue again. it'd make things really easy, if a control can own another.

in python, the idea of using only a known set of interfaces when dealing with Ctrl is crucial, to be able to make dynamic guis, where Value is the core data element (which is pretty much what a Python object is). a python exported Ctrl is instanced by python interpreter and is able to manage the instance itself or transfer the owning to another facility. creating ctrl instances in python is crucial, when decorating ctrl functionality with python means is desired..

but maybe exporting upp ctrls to python will also benefit from the stuff i described before (the additional dont-touch-upp layer).

anyway, maybe this should be reconsidered.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Tue, 19 Apr 2011 04:00:56 GMT
View Forum Message <> Reply to Message

We'll probably not going to see owned children in Upp any time soon. But this problem HAS been elegantly solved: Interface similar to Array::Create<ChildType>() to avoid end user having to use new operator; virtually no additional cost except a couple of cpu cycles; no existing code will be broken.

Please let me know if I am wrong

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Tue, 19 Apr 2011 07:14:10 GMT
View Forum Message <> Reply to Message

where can i see this interface? you dont mean mine

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Tue, 19 Apr 2011 12:09:20 GMT
View Forum Message <> Reply to Message

In post 31696. I change the interface so that using new is no longer required and no longer permitted. When user wants a child to be owned(thus destroyed) by its parent, he/she uses the following way:

```
parent.AddChild<Label>().SetLabel("Hello world!")
            .PosRight(...).PosTop(...);
Button& b=parent.AdddChild<Button>.SetLabel("Click Me!");
b<<THISBACK(ButtonClicked);
```

Non-owned child controls are still added in the old way.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Tue, 19 Apr 2011 16:05:37 GMT
View Forum Message <> Reply to Message

now got you..

though it's nice the way you planned it, the way upp handles ownership in containers i.e. is a bit different.

there, the implicit rule of thumb is (asuming Array is used)
that a pointer denotes a freshly created element (see Add(T* x) function in Array) for which the container needs to take ownership. moreover, there is the Detach() function, returning the pointer indicating that ownership has been stopped and the user needs to take care of that element again.

sth like this should also be expected from the ownership interface for Ctrl (if anytime to come).

IMHO ownership is not usefull in a C++ only / static environment (where no controls are created and thus nothing needs to be deleted). they are defined in Layout and done.

for dynamic environment OTOH, where controlls with a reduced set of api (GetData,SetData,WhenAction Callback) are created and destroyed in a deliberate manner, ownership would improove codeability.. (especially thinking about scripting layouts, like my current intent: Python).

thus maybe mirek can help with some ideas. but it needs to be thought out well, since it touches the upp philosophy..

BTW: as soon as the last little fix from CtrlLib is online, i'll post here a current environment, for dynamic control handling, like promised.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 20 Apr 2011 00:44:11 GMT
View Forum Message <> Reply to Message

Adding Detach is a piece of cake.

Something like this:

```
Ctrl& Ctrl::Detach()
{
    if(owned)
    {
       owned=false;
       parent->RemoveChild(this);
    }
    return *this;
}
```

I intentionally disallow all subsequent changing owned state capabilities so that it appears less dangerous/error-prone. If nobody can detach it, its destruction by its parent is guaranteed. Changing parent will not break the mechanism. Detach and leave alone will open the door for memory leak.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Wed, 20 Apr 2011 06:51:25 GMT
View Forum Message <> Reply to Message

Lance wrote on Wed, 20 April 2011 02:44Adding Detach and leave alone will open the door for memory leak.

Detach is dangerous, but it's there and sometimes needed. the coder has to have the possibility to handle stuff in fair custom ways as well. the containers do have it too.

could you provide the files with all the latest changes as one? i'm bit lazy to collect it.. i'd like to review it maybe can get some idea..thanks in advance

cheers

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 20 Apr 2011 17:24:08 GMT
View Forum Message <> Reply to Message

Will do.

I am working on something I called RecordSet/Dynamic Struct, maybe a reinvention of wheel. But it keeps really busy.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 20 Apr 2011 19:17:05 GMT
View Forum Message <> Reply to Message

I didn't backup my original work. Basically I have to redo what has been done.

Overwrite the Upp libary version of Ctrl.cpp CtrlCore.h CtrlChild.cpp (or whatever names) with attached version. Backup before proceed.

BTW, those files are in $(UPPMAIN)/uppsrc/CtrlCore directory.

Attached is also a test program. How do I know it works? No news is good news: had there been memory leak, our honest TheIDE will not forget to prompt us.

File Attachments
1) CtrlSupportOwnedChildBackup.rar, downloaded 465 times

Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 20 Apr 2011 19:23:36 GMT
View Forum Message <> Reply to Message

Out put of sample program:

And the main content of the sample program:

```
using namespace Upp;

class MyLabel : public Label
{
public:
 MyLabel(String& s){ SetLabel(s); }
};
```

```
struct ButtonApp : TopWindow {

 ButtonApp() : count(0)
 {
  Sizeable().Zoomable();
  Click();
 }

 void Click()
 {
  int x, y;
  Ctrl * p;
  x=count%max_row_button*(button_width+5)+2;
  y=count/max_row_button*(button_height+5)+2;

  if( count & 1)
   p=&NewChild<MyLabel>(String().Cat()<<"["<<count<<"] A Label");
  else
  {
   p=&NewChild<Button>()
    .SetLabel(String().Cat()<<"["<<count<<"] Click Me");
   *(Button*)p<<=THISBACK(Click);
  }

  p->LeftPos(x,button_width).TopPos(y,button_height);

  ++count;
 }

 typedef ButtonApp CLASSNAME;

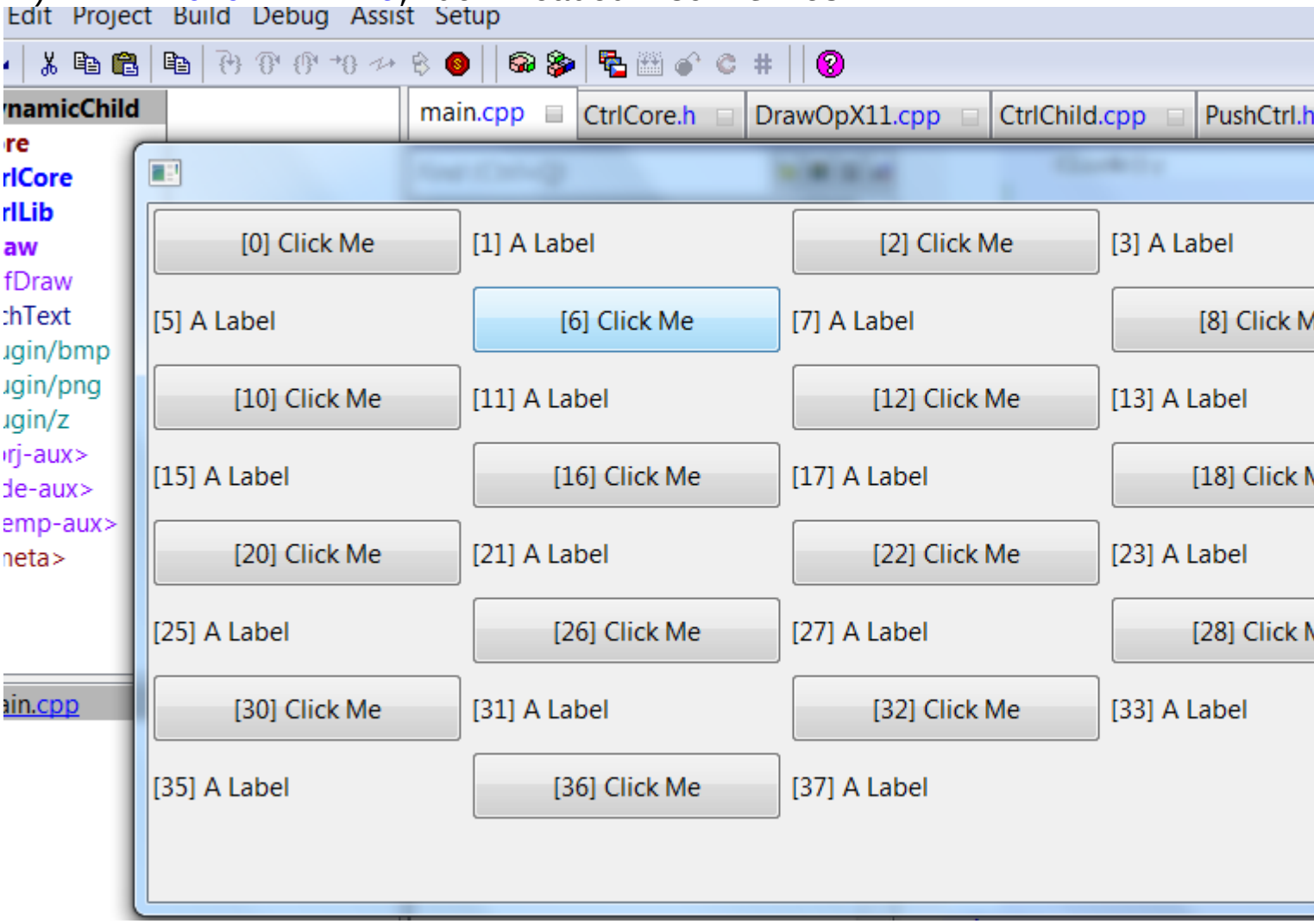
private:
 const static int button_width=180;
 const static int button_height=40;
 const static int max_row_button=5;

 int    count;
};

GUI_APP_MAIN
{
 ButtonApp().Run();
}
```

File Attachments

1) DYNAMIC CHILD.PNG, downloaded 1391 times



Subject: Re: Proposed change to U++ to allow owning children.
Posted by Lance on Wed, 20 Apr 2011 19:38:59 GMT
View Forum Message <> Reply to Message

Detach is implemented as protected. Intended for libary developer or person who knows how to handle it as it opens the door for possible leaks!

Also, there is one step I omitted. The owned flag should be unset in Ctrl copy constructor. If pick semantics is guaranteed, this will not be an issue. There is something we need to worry about only when there is prospect that the fix will be included into the library.

The reason being, a programmer may copy a dynamically created Ctrl to a stack allocated object. If this happen, the owned flag is wrongfully set and will lead to memory corruption. The move constructor (pick semantics) should always be preferred. The copy constructor should be "protected" so that it's available only to library developer or person know its potential danger.

Subject: Re: Proposed change to U++ to allow owning children.
Posted by kohait00 on Thu, 28 Apr 2011 14:32:57 GMT
View Forum Message <> Reply to Message

just as promised here comes my current test package, it's really a test package.

it can create on the fly new controls (which are meant to be hooked to a specific interface which is not yet finished). the gui can be saved and restored.

just a short description:
* go to live work tab
* click small rectangle at top left in view area, this will switch to edit mode
* click some control, it becomes selected and the properties are displayed and can be modified on the fly (live work)
* moving can be performed just as used from RectCtrl, if ever tried, use STRG, SHIFT as modifiers to restrict movements..
* while moving a control, hold ALT to move it *into* another, adding it as child, or move it outside its parent.. very fluid.

* the bottom frame of view area has got a Ctrl prototypes factory, it uses the same moving + hold ALT means to place the prototype onto the surface of another control or view area. the factory recreates the control in its own view area again.
* moving + hold ALT a control from somewhere into the factory again will leave it there pending for delete. the next control moved to the factory will toggle a remove of the previous. thus you can move a 'deleted' control back to surface if you change your mind.

* i'm extending the current Ctrl with an interface that manages the ownership of similar controls. it wont own Ctrl's that are not derived from that interface.

beeing already said its only my test environment. it's not complete. but it's what i plan to do.
basicly inspired from
jazzmutant's lemur control (if ever heared of it, thus a python console is to be included there as well, see BoostPyTest in bazaar).

let me know if that goes in your direction.
i admit, Ctrl having ownership means would greatly simplify this all.

File Attachments
1) LiveWorkTop.rar, downloaded 431 times

Subject: revised ownership change
Posted by kohait00 on Thu, 28 Apr 2011 16:28:42 GMT
View Forum Message <> Reply to Message

hi Lance

i've gone though the code and based on your work redid it.
current state is attached.

the scenarios are basicly theese:

1) normal scenario (current upp)
Ctrls are added to arbitrary context to be displayed, ownership context is maintained somewhere else. -> 2 different contexts, perfect control over whats happening. for complex environment.

2) ownership scenario (additional)
heap created controls can be added to other controls that will maintain their lifetime together with their visual context. means, when an owned Ctrl is removed from its owning parent, the parent is taking care of destruction. so both contexts, visual and ownership are bound. fits well for dynamic environment with reduced Ctrl API usage (GetData/SetData, WhenAction, etc)

features

* traditional upp way still default
* ownership adding (AddOwned, Detach) is only interface for dealing with ownership (similar to containers)
* auto transfer of ownership supported when an owned Ctrl is added to another parent.
* least intrusive code
* drawback: owned controls may not call Remove(), since their parent will try to delete them. ASSERT placed. arguable.. use Detach for that.. or specify implicit Detach with Remove?

changes to your version:

* removed template creators, not needed for ownership management.
* code cleanup and simplification
* redefined public interface for ownership (mainly: no public Owned(bool b true) method, veeery dangerous)
* some ASSERTS placed to ensure proper behaviour and early fault catch
* delete only in one place.. avoids some conrercases when context switching.

take a look..
maybe mirek will be really considering it. at least look at it

File Attachments

1) DynamicChild.rar, downloaded 453 times

Subject: Re: revised ownership change
Posted by Lance on Thu, 28 Apr 2011 19:45:59 GMT
View Forum Message <> Reply to Message

Good job.

Except the template creator should be kept. It's simulating the Array::Create<ChildType>() interface. Look at it this way, a Ctrl is a special type of container: it references some of its children while owning (responsible for their destruction) some others. Since allowing library user to change the owned flag is regarded as unsafe, and use of new/delete by library users is generally discouraged, it makes sense to have a similar Create/Detach interface like does Array.

The point of the template creator (NewChild, AddOwned, CreateOwned or whatever we may choose to call it) is to eliminate the necessity of asking end user to supply a new'ed object, instead of:

```
parent.AddOwned(new Button());
```

now use

```
parent.CreateOwned<Button>();
//
// similar to
// anArray.Create<Button>();
//
```

Yes, any access to owned flag should be protected. That's my true intention. Thanks for fixing it.

I still think that to expose Detach to public scope is not a good idea. It should be protected and only accessible through inheritance. Even though it works in common/reasonable senario:

1. a Ctrl is Detached from its parent:
   In any case, the Ctrl is removed from its parent's children list. If the Ctrl has owned flag set, a pointer to it will be returned so that user code will be responsible for its eventual destruction; if its a normal stack resident Ctrl. a NULL pointer will be returned to signal the user not to try to delete it afterwards.

2. the user delete's the detached Ctrl after finished using it. No problem, expected scenario.

3. the user decide to add the ctrl to another Ctrl as child by way of Add(), AddChild(), etc. Since the owned flag is correctly set, the new parent will be responsible for its destruction.

So it seems no hole is introduced. But here Add(), AddChild() becomes ambiguous to the user. The user may form an impression that Add()/AddChild() can be supplied with any new'ed Ctrls without the user to worry about their eventual destruction. To avoid this confusion, maybe it's best to simply hide it from library users.

Subject: Re: revised ownership change
Posted by kohait00 on Thu, 28 Apr 2011 19:57:25 GMT
View Forum Message <> Reply to Message

for Create<> i go with you..

for Detach i disagree. there is no point in offering Ownership ability to Ctrl without offering the complete interface to manupulate it. calling RemoveChild() would delete the child. so there is actually no way to safely get the owned ctrl back from there.

the question is
should Detach be handled explicitly or should a Remove() call on an owned child be seen as detaching it?

i think the later is quite logical, but it'd leave dangeling references in places Remove is used without the awareness of dealing with a floating reference.

Subject: Re: revised ownership change
Posted by Lance on Thu, 28 Apr 2011 21:18:09 GMT
View Forum Message <> Reply to Message

Personally I don't dislike Detach at all. It's hard to think of a situation where Detach causes problem more than Array::Detach does. The only problem is that it will make the code harder to understand for somebody who doesn't go deep enough on this topic.

And if a Child control can be Detached, could it be reattached to another parent? If the answer is not, than Detach would be perfectly fine. If the answer is yes, essentially you open the door for add a user newed object which Mirek objects.

So either:

1. Call parent.CreateOwned<ChildType>() to create the object on heap and add as parent's child;
2. When the child is expected to be killed, hide it, access all its properties, and call parent.Remove(..)

or

1. Call parent.CreateOwned<ChildType>() to create the object on heap and add as parent's child;
2. When the child is no longer needed, Detach it and remember the returned poiter.
3. Access its properties, and delete it manually.

The advantage of case 2 over case 1 is that it allow the parent be destructed while the child remain valid, think about the case where the parent itself is its parent's dynamically created child.

The disadvantage of case 2 over case 1 is that it requires the library user to use delete. But since

he/she has to delete the object returned by Array::Detach, it should not hurt that much.

If you agree with my above analysis, I go with case 2 too. Or do you have other recommendation?

Subject: Re: revised ownership change
Posted by Lance on Thu, 28 Apr 2011 21:31:33 GMT
View Forum Message <> Reply to Message

In either case 1 or 2, a CreatedOwned child can change parent like any other stack Ctrl without losing track. So this is not a concern. Or, change parent is perfectly defined behavior for either type of children.

If we go with Case 2, CreateOwned+Detach, a detached child should never be reattached to a parent [by calling AddChild(...) etc]. It's a final decision, you detach it, you destruct it. However, Derived class can reattach by way of protected member function.

This is a virtually zero cost, safe, and consistent solution as far as I can see.

Regarding the hole when copying/picking(yes, picking also requires reset the owned flag), we can tinker the Ctrl pick constructor a little bit to fix it.

Subject: Re: revised ownership change
Posted by Lance on Thu, 28 Apr 2011 21:54:08 GMT
View Forum Message <> Reply to Message

In the rare case where we are destructing its parent but could not make up our mind whether we are going to delete it or assign it to yet another parent (not decided at the moment), we can use a dirty trick to circumvent the no-reattachment limitations,

```
{
  // p is pointed to a dynamically created child

  ParentCtrl tmp;
  tmp.Add(p); // or should it be *p? I rely on TheIDE on this
              // now p is no longer part of its previous parent's
              // child tree.

  // Destruct its previous parent to free precious memory resource :)
  // and since *p belongs somewhere else, it will not be touched.

  // a thousand lines/function calls to make up our mind whether
  // we are going to destruct *p or add it to another parent.

  if( newdad !=NULL)
```

```
    newdad->Add(p);

  // we don't even need to detach and delete, as p will be
  // destructed with tmp at the end of the code block
  // if a newdad is not successfully found.
  // this is the expected behavior

}
```

Subject: Re: revised ownership change
Posted by kohait00 on Sun, 22 May 2011 19:58:07 GMT
View Forum Message <> Reply to Message

i think mirek wont let this happen, because there is really quite a lot of flaws in it ispecially the Remove() call of an owned Ctrl is a huge problem. since this would delete the Ctrl itself. if not, it would dangle. but there are a lot of handling expecting the Ctrl to continue to exist after calling Remove() on it.

so thats why i will procure in developping my LiveWork environment..

have you tried it?

Subject: Re: revised ownership change
Posted by Mindtraveller on Sun, 22 May 2011 22:28:54 GMT
View Forum Message <> Reply to Message

I think U++ owning system is far more efficient than ordinary approach. So I wouldn't recommend embed QT-style management of controls.

It is better learning better approaches than creating patches to allow old-style programming. Believe me, it is worth your efforts.

Some time ago I've made an article and had discussion about it here (it's in Russian)
http://habrahabr.ru/blogs/cpp/111259/

Subject: Re: revised ownership change
Posted by kohait00 on Mon, 23 May 2011 09:16:56 GMT
View Forum Message <> Reply to Message

i agree. will take the effort to read it, though living in germany i originate from kazachstan

the upp approach is worth gold (everything belongs somewhere) and some old habbits die hard.. that's why i decided to use a distinct interface class to manage ownership. this will make it esier

anyway.

i'm often tempted to drive upp towards the object base /polymorphism approach i.e. like in C# and forget about upp hard type checks.. which is the source of it's beauty and speed.

any polumorphism should be handled extra, in the specific containers.

Subject: Re: revised ownership change
Posted by Lance on Mon, 06 Jun 2011 03:05:38 GMT
View Forum Message <> Reply to Message

kohait00 wrote on Sun, 22 May 2011 21:58i think ispecially the Remove() call of an owned Ctrl is a huge problem. since this would delete the Ctrl itself. if not, it would dangle. but there are a lot of handling expecting the Ctrl to continue to exist after calling Remove() on it.

For example? Do you mean UPP library code relying on the continued existence of a control after it's been removed from its parent, or, user code, as a common practice, may access a dynamic control's properties after it's been removed (hence deleted) from its parent?

I agree the latter is an issue as it's against Upp programmer's habit.

I was really busy. I know there are lots of good stuff in the bazzar worth trying and learning, but I will have to wait.

Subject: Re: revised ownership change
Posted by Lance on Mon, 06 Jun 2011 03:11:07 GMT
View Forum Message <> Reply to Message

Mindtraveller wrote on Mon, 23 May 2011 00:28I think U++ owning system is far more efficient than ordinary approach. So I wouldn't recommend embed QT-style management of controls.

It is better learning better approaches than creating patches to allow old-style programming. Believe me, it is worth your efforts.

Some time ago I've made an article and had discussion about it here (it's in Russian)
http://habrahabr.ru/blogs/cpp/111259/

True. With my limited experience with U++, I never need to dynamically allocate a control. A common situation would be to create UI from , eg., XML, but in that case, a dedicated container like an hash map would be more efficient as it will be frequently required to access controls by their names/IDs.

My believe is that it's of virtual no cost but may of some use some time.
