Subject: Re: Core chat...
Posted by mdelfede on Fri, 26 Oct 2007 11:03:54 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 26 October 2007 12:42
Sorry. My mistake. Forget about [10]. Only

ctrls.Create<Button>();

or (to be more clear):

Array<Ctrl> a, b;
a.Create<Label>();
a.Create<EditField>();

b = a;

b.Create<Label>(); // Now what?!

Array<Ctrl> a, b; // two empty arrays, reference to nothing
a.Create<Label>(); // adds a Label to a, a becomes an array with a single reference at underlying data
a.Create<EditField>(); // adds an EditField to a, a is grown by 1 element, as it had a single reference to data, nothing strange happens.
b = a; // now a AND b have both reference to the same underlying array
b.Create<Label>(); // underlying data is copied ....

I see your point, here you *did* want a single array of controls, and you *did* want to destroy a when copying to b. You could have easily write :

Array<Ctrl> a;
a.Create<Label>();
a.Create<EditField>();

Array<Ctrl>&b = a;

b.Create<Label>();

That leaves both a and b pointing to SAME array. You then could say that when a is destroyed, b points to nothing, that's true, but I can hardly imagine such a case. For example :

Array<Ctrl> MyFunc()
{

```
  Array<Ctrl> a;
  a.Create<Label>();
  a.Create<EditField>();

  Array<Ctrl>&b = a;

  b.Create<Label>();

  return b;
}

Array<Ctrl> c = MyFunc();
```

still works... just before a is destroyed (and thus b points to nothing...), c adds a reference to its contents, then it remains the sole owner of it when function ends.
Let me say that in my case it's clear what you pretends to do, in yours you must know pick behaviour, and still you have the possibility to write an erroneus a.Create<SomeControl>() AFTER a is picked. In my case, you can AND you get what you want.
You may still say that if you do a.Create<SomeControl>() you get an error, ok... and I can tell you that if you forget the & you have double controls on screen!

Ciao

Max