Subject: Re: Core chat...
Posted by mdelfede on Fri, 26 Oct 2007 10:13:44 GMT
View Forum Message <> Reply to Message

luzr wrote on Fri, 26 October 2007 09:36
Sure. But then, what now:

ctrls[10].Create<Button>()

(To make me more clear, "COPY" of the "COPY ON WRITE" is impossible...)

uhmmmm... no, you were not clear enough !
Looking on Array::Create<>() is defined as

```
 template<class TT> TT& Create()     { TT *q = new TT; Add(q); return *q; }
```

and no Create<>() is defined for Ctrl class. So I suppose that your ctrls is an array of array of controls... maybe. Your examples becomes difficult to understand...
So, you're taking the 11-th element of your array, wich is an array of controls, and add to it a new button, ok.
Let's start from the beginnin (I'm thinking as I write, sorry!):
with ctrls[10] you get a reference to 11-th array (element must exist, but I guess it's not important).
You add then a new button to it, ok. That should be equivalent to :

```
Array<Array<Ctrl>> a;
Array<Ctrl> b;
b.Create<Button>();
a.At(10) = b;  // just to be sure that 11-th element exists...
```

which in your pick behaviours leaves a owning b contents and b in picked state.
I still don't see caveats in my Copy-On-Write behaviour, besides of the impossibility of using [] operator because missing lvalue-rvalue signature difference.
If I do the same with my class, I have :

```
Array<Array<Ctrl>> a; // empty array of arrays of Ctrl
Array<Ctrl> b; // empty array of Ctrl
b.Create<Button>(); // no problem, a button is added to b
a.At(10) = b; // a[10] adds a reference to b object... no problem
```

then, I do have 2 possibilities :

```
b[0] = myCtrl; // don't look for a while at [] problem...
```

b becomes a DIFFERENT array as a[10], and a[10] points to an object that has a single reference
Or... b exits from its scope, leaving object at a[10] again with a single reference.
You can join all that on a single line, just replacing this damn' [] operator with some member function OR don't care about havng copy-on-write even on reads if array has more references.

Let's look at the second case (keeping []) :

Array<Array<Ctrl>> ctrls;
ctrls[10].Create<Button>()

the 11-th element of ctrls gets a button added, I don't see caveats. Where am I wrong ????

BTW, the *real* caveat of copy-on-write behaviour is the missing difference of signature between lvalues anr rvalues, which gives you 2 choices :
1- Make(if possible) operator[] returning a read-only element, and use some other function to set element value; that is the most efficient, which leads to true copy-on-write when array as more than 1 reference on it.
2- Let [] do the jobs, so you can have also copy-on-read if array has more references, which is a waste of time.
As I said before, C++ standard as *many* caveats and missing stuffs...