Subject: Re: Core chat...
Posted by mirek on Wed, 24 Oct 2007 16:23:03 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Wed, 24 October 2007 11:46luzr wrote on Tue, 23 October 2007 23:48
I am still not quite sure why people insist on such verbose syntactic sugar (it is really nothing else).

IME/IMO, there is usually much more properties to set than get and setting properties is much more convenient U++/C++ way...

Well, if you think so, you can have object-oriented code in plain C also, look in GTK code to have just an example.... or even in assembler.
That doesn't mean 'clean object programming' in my way of thinking.
Please understand that I'm not criticizing UPP, I still think it's one of the best coded toolkits.
What I do criticise is the lack of some very useful constructs in C++ that could make code much cleaner to write and to mantain.
Constructs that don't cause loss of performance, as properties.

Well, in any case, GTK code is much more verbose than C++. U++ "modifiers" are less verbose than properties.

Quote:
Wait a moment - this issue is quite rigirously documented. There is precisely defined which container methods invalidate references.

Yes, all is documented somewhere, you did know that but... still the bug was there. In my thinking, for example, a function that has the main purpose of accessing array element should *not* as a side effect invalidate references.
[/quote]

I agree. However, the main purpose of At is not accessing array element, but to create one if it does not exist yet.

Quote:
I know that write

a.SetSize(100);
a[99] = 10;

is one line longer than

a.At(99) = 10;

But that is not the correct equivalent.

Quote:
```
if(a.GetCount() < 100)
   a.SetCount(100)
a[99] = 10;
```

is the equivalent. OK, that would be something to live with, but consider composition:

```
Vector< Vector<int> > a;
a.At(10).At(20) = 10;
```

- that is the real reason to provide such utility method.

Quote:
You where complaining about shared ownership of objects (and you're right about it, if you use pointers), but in case of At() function you do have a case that is very similar, because you use [] operator that you think operates in an object that is invalidated by the At() function as a side effect.
Ok, it's documented, but then also realloc() is documented, but I've seen MANY times code like this :

```
int * a = malloc(100);
realloc(a, 200);
*a = 10;
```

and that's MUCH less subtle bug than yours with At().

Well, I guess there is always instrinsict conflict between performance, convenience and safety. "At", and in fact, the whole concept of "inplace creation" (instances get created in container, reference is returned) is nothing new, it is part of U++ design since the very beginning and proved to be very useful.

In any case, I believe I would make more bugs in "unrolling"

```
a.At(x).At(y) = ...
```

code than using such utility method.

Mirek