Subject: Re: Core chat...
Posted by mdelfede on Thu, 25 Oct 2007 21:26:17 GMT
View Forum Message <> Reply to Message

luzr wrote on Thu, 25 October 2007 21:33

You can try. However, long time ago, such class template was part of U++. But there was no use for it. "pick" is confusing at first, but quite powerful concept.

I do find "pick" quite clear. You get the top performance at at the expense of some loss of easy to use stuff.
With deepcopy behaviour, you have the opposite.... large performance loss with the gain of ease to use.
I think using refcount objects is a compromise between both, you have the same easy to use as deepcopy behaviour at the expense of a *small* performance loss in respect to your pick mechanics, that I guess is mostly due to a double indirection accessing elements, when you don't need a deep copy.
In your class :

```
array<int> a, b;
a.At(1000) = 1;
b = a;
a[10] = 2;  <==ERROR
b[10] = 2;  <==OK
```

in my class :

```
array<int> a, b;
a.At(1000) = 1;
b = a;      <== b and a share same memory area
a[10] = 2;  <== here an automatic deep copy, ok
b[10] = 2;  <== here no deep copy, just array access, ok
```

in usual deep copy behaviour :

```
array<int> a, b;
a.At(1000) = 1;
b = a;      <== deep copy
a[10] = 2;  <==OK
b[10] = 2;  <==OK
```

in 95% of cases, my array behaves exactly as yours, but... in the rest 5%, you DO have to specify WithDeepCopyOption, in mine that's done automatically, with a 'small' performance penalty.
What I'm really curious about is how 'small' that is....

BTW, returning an array from a function, my class don't need a deep copy, as yours, as the origin

array is released when function ends, leaving only the result reference to array.

array<int> a;
a = MyFunc(x);

array a gets 2 references to it for a while, just when MyFunc returns a value. Then, temporary from MyFunc get destroyed, leaving a with a single reference. Any subsequent access to a[] don't need a deep copy.

Max
