Subject: Re: Core chat...
Posted by mirek on Wed, 24 Oct 2007 20:05:56 GMT
View Forum Message <> Reply to Message

mdelfede wrote on Wed, 24 October 2007 14:24luzr wrote on Wed, 24 October 2007 18:23
Well, I guess there is always instrinsict conflict between performance, convenience and safety.

But, in modern compilers, doing a 3 line construct or an equivalent 1 liner At() should be the same in terms of performance.

Nobody argues that...

Quote:
BTW, I really see few practical usage of At();

Well, in that case, just do not use it:)

Anyway, it is used about 100 times in the uppsrc. There are many cases when it is the right thing to do - if you do not want to duplicate the code.

Quote:
normally (about 100% of cases) you know in advance your needed array size

Wrong, in 99% you do not know the GetCount of resulting Vector or Array. Or you do not WANT to know it - I mean, sure, it is e.g. possible to count all elements in the file first, then create the array of the right size, the reopen the file and load them in. But load them in into dynamic array is much more simple...

Quote:
, and, if you don't, I don't really see the point of increasing (and such copying the whole stuff) the array on a 1 element basis; se that example :

```
Array<int> a(1);
for(int i = 2, i < 1000; i++)
  a.At(i) = i;
```

of course that works, but that means, in the worst case of a badly written Array code, 998 realloc() calls, with 998 buffer copy, memory releases and allocations, ecc ecc.

Well but that is completely different issue altogether... But be sure that Vector code is not badly written

Quote:

In a better and more realistic case, if the array is grown in chunks of 10 elements, you'd still have to do it 98 times. That's no efficiency neither good code.

Growing by static chunks is very stupid method. You always need exponential growth - this is the same for NTL and STL. In that case, the total number of copying stuff is amortized constant - both for STL and NTL (but for NTL, unlike STL, the copy of Vector element is performed by raw binary move, which can be much faster).

Quote:
Where the CheckSize(minSize, increment) is a member function that checks the actual/required size of array and, if too small, make it to grow of a specified size (here 1000).

Works well speed wise only for small array sizes - but for small ones, it wastes memory. What you really need is exponential growth.

Quote:
That'd be an huge preformance gain with a small line of code added.

Current At method is more optimal.

Quote:
Here I'd let the At() the sole purpose of element accessor and put somewhere else the array resize. I don't see nothing bad on a.At().At().At(), as they don't change array sizes.

Why should you duplicate operator[] with a method?

BTW, if you want to study easy to undestand practical examples of using At, look at ArrayCtrl::SetDisplay or Switch::Set.

Mirek