Subject: Re: I don't get some aspects of STL ... [pointless rant]
Posted by cbpporter on Thu, 19 Mar 2009 08:33:56 GMT
View Forum Message <> Reply to Message

I'm sorry to disappoint you if you're looking for some enlightening comments that will help you appreciate STL better. Coming from NTL it will allays suck.

I never used STL for extended periods of time, and it's ugly design is probably the reason for it. At first I used Delphi for serious development, then MFC which had it's own containers. Then U++. And STL here and there in between.

But my latest effort in a small utility tool needs to use STL and I'm quite hatting it. I took me over an hour to load a file in memory. There is no LoadFile. I had to use a file stream, go to the end, take the position, compute the size, go to the start then allocate the buffer and read. I found that there is a flag you can use in the constructor so that the file cursor is positioned at the end of the file, so now I have take position, go to start, read. A lot shorter. The result of my efforts was an aptly named LoadFile function . Something like this should be part of the standard library. Loading files line by line is OK, but when creating parsers, loading the full file is often a must.

Also, other useful conversion functions seem to be missing. Try converting and int to a string, or vice-versa. You have basically two options. A stringstream and <<, or c string with a local buffer of fixed size and itoa. One would think that such basic conversion would be part of a general library, or at least string + int would work, seeing as strings don't convert to char*.

Iterators are also incredibly ugly and verbose. and the worst part is the lack of uniformity. I can navigate a vector or string by index, but that's about it. Delete for example needs and iterator. Even on vector. Deleting the fifth element is a simple as v.erase(v.begin() + 4).

Other gripes: wstring is again somewhere between Unicode 1 and 1.1 and wchar_t is 16 bit or 32 bits depending on platform. This wouldn't be a problem if STL would be Unicode aware, but it's not.

And please, include boost::format already. I hate cout<< because setting output options is again ugly and verbose. cout.width(10); cout << 100 << endl; cout.width(10); cout.flags (ios::right | ios::hex | ios::showbase); cout.width (20); cout << 100;
Yes... right... ahem...

But will all this, STL is really great. Looking both at the history of C/C++ and also current OSS projects, STL is a blessing. Every single C and even C++ project I have seen defines it's own int_t, int32_t, int32, int_32, __int32 or other variants of basic types. Also, prefixing it with the name of the library or the first letter is very popular: gint, fint, glint, axiom_int. If you want your programs to be portable and sizes of types are so different, than the standard should have a type that is guaranteed to accommodate some restrictions. You need a 32 bit integer? The standard says you need _____int_std____C__or_C___ESCP_ESCP_32_bits. There! Problem solved. This way not every single header will define these types. Actually, there is stdint.h, which offers a little more friendly names for types like int32_t and int_least32_t.

But redundant typedefs are only the tip of the iceberg. The problem is that every single C lib

reinvents basic containers. Try building an application by combining open source libraries. In a project I ended up with 3 C string types and two C++ string types. Ironically the two C++ types were normal std::string, and normal std::string which support NULL values for DB interactions.

So basically, for every C++ library that doesn't invent it's own vector, STL gets on huge award and my gratitude. IMO it is a lot better if everyone uses the same ugly library, rather then them using separate libraries, which will always overlap in functionality and are often ugly to.
