**String integration**

After programming in MFC for a few decades, many new projects that I started used MFC merely because I was so accustomed to the [CString](https://docs.microsoft.com/en-us/cpp/atl-mfc-shared/using-cstring) class. Even to the point that I integrated MFC in those projects just to use the CString class. A few attempts to do away with MFC altogether failed. Not because [std::string](https://en.cppreference.com/w/cpp/string) is not a good string class, but because of the familiarity with all the methods of the former string class. Besides, CString knows of a few good tricks that you occasionally just need, like getting BSTR-ings, getting an environment variable, or all sorts of tricks like ‘FormatV’ where you would otherwise need an extra function. So I would half-way some algorithm be stuck trying to remember ‘how-to-do-this-in-this-string-class’ anyway. That would then interrupt my train of thoughts. This roundtrip through some string operation would often cause a distraction of the work at hand.

**One solution: study harder**

A solution would of course be to study harder on std::string. But knowledge is not just the one solution. Literally million of lines of code are hanging around. Not only in my private projects but also at the office. At work we need to support quite a lot of millions of lines of code with MFC’s CString in them. So I would always be betting on two horses and trying not to mix them both up.

**Another solution: elimination**

Another possibility would obviously be to eliminate one of those classes. Preferably the older one (MFC CString) would have to go. There is no reason why you cannot add std::string to a mixed source base with other classes in it. And I was already using the STL containers like ‘vector’, ‘set’, ‘map’ and ‘multimap’ everywhere in favor of the old MFC containers. But still: this would be a lot of work with no extra’s like extra functionality attached to it

**My solution: integration**

The road chosen here was to integrate the two of them. Derive a new class from std::string, so the new class has all the standard functionality of the STL standard string class. And then add all the MFC CString methods to that class. So we would have both of them at the same time.

*Now I hear you thinking: “YUK!” What a horrible thing to do!*

Yeah. I know! The end result is NOT a nice and comprehensive class. But wait…. The winning part is not in this end result. The winning part is when you are programming and do not want to stop doing what you are doing. All the things you would be required to remember are now wrapped inside the new string class.

Another winning property is that you can use it as a drop-in replacement.

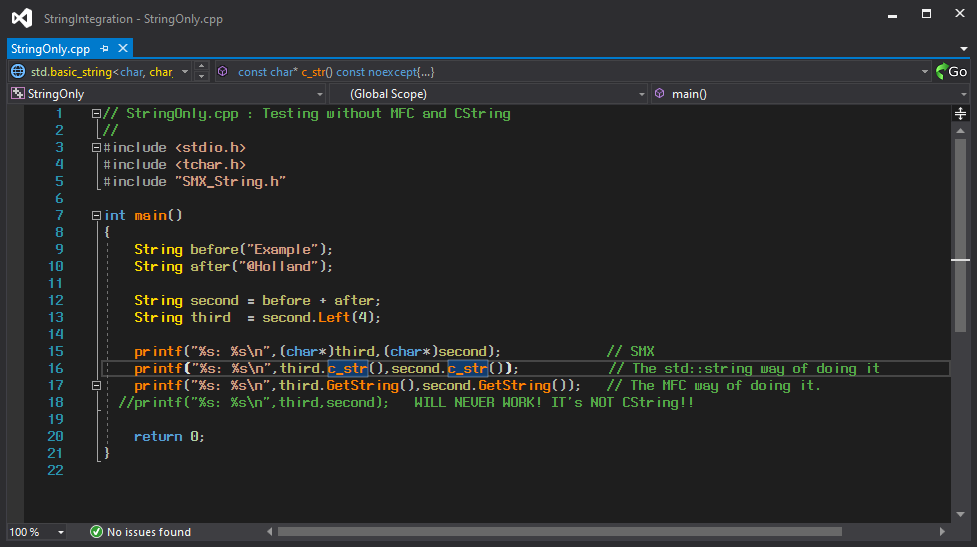
The new class is called “SMX\_String” which stands for “Standard MFC eXtended String”. The interface definition in the “SMX\_String.h” file is ended with a choice of name overrides. Here we choose a more handsome name to program in. The first case being just “String”, as defined by a typdef.

Just in case we have a project at hand with only a CString in it, we can rename the CString with a typedef to our new class. We could even rename a “std::string” with a macro (but that is not a nice solution).

|  |
| --- |
| // Now typedef this as a standard "String"  typedef SMX\_String String;  // Use this typedef for a plugin-replacement of a MFC CString-only project  // typedef SMX\_String CString;  // Use this in a std::string based project to override the standard string  // #define string SMX\_String |

**The end result: both sets of methods are accessible**

And there you have it. You can now have a project without MFC and a string class that can do both the tricks of std::string and MFC::CString in one package.



We can now do everything with a “String” that we want. In the example above you can see that you can do both “c\_str()” and “GetString()” and that a “Left(n)” is also working. No need to go over millions of lines of code to eliminate al those “Left()’s” 😊

**What’s not working**

And of course there are a few things that are definitely not working. That would need a lot of reworking on a std::string to get working at all. Here are those few things:

* The default (LPCTSTR) casts will never work. CString is using an elaborate scheme with a StringData object, so that a pointer to a class object coincides with the storage of the string data. Which in its turn makes it possible to dereference a memory pointer directly as in the fourth ‘printf’ example here above;
* Lock and Unlock. The std::string class do not have a locking scheme. You will need to do your locking outside of the string object;
* Refcounting. But we have autopointers in the standard library!
* Copy on write (COW). Clearly for single-threaded programs with small strings a fine thing to have. But COW has been banned from the standard library. See some of the arguments by Herb Sutter [here](http://www.gotw.ca/gotw/045.htm) and by Giovanni Dicanio [here](https://blogs.msmvps.com/gdicanio/2016/07/09/is-copy-on-write-really-a-pessimization-under-multithreading/)

**Some details that (might) bite you**

There are a few details that you should be aware of, or otherwise the \*\*will\*\* bite you!

* Take good care of the “Advanced project settings” in Visual Studio. Especially the “Use MFC” setting;
* Take good care of the standard precompiled header settings and any ‘stdafx.h’ that is still in use in your projects. I forgot about them in a good number of projects. Remove the “#include <afx.h>” and other settings or convert to the newer convention of using a “pch.h” file.

Well that’s about all. Have fun with the “SMX\_String”!