

Re: CString to const char*

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.language/2006-09/msg00778.html>

- *From:* "Abdo Haji-Ali" <ahali@xx>
 - *Date:* Wed, 27 Sep 2006 15:43:59 +0200
-

"NickP" <a@xxxxx> wrote in message
news:eHvzYzi4GHA.4256@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

For some reason it will not convert to const char* automatically, if I try manually I just end up with a screwed up ASCII string still containing

0

bytes from the UNICODE equivalent.

Remember that there are two versions of CString: CStringA and CStringW. Which one is used depends on whether UNICODE is defined or not...

```
char *pBuffer = new char[iString.GetLength()];  
pBuffer = (char*)iString.GetBuffer(sizeof(pBuffer));
```

GetBuffer() would return "const wchar_t*" (since UNICODE is defined) not char*. Casting the result to char* won't convert the string from ASCII to unicode. See below

```
MessageBoxA(NULL, pBuffer, "", 0);
```

Strange thing is my CString class is within the WTL namespace, is there

more than one implementation of CString available?

What would be the best object to use for string handling, I thought this

class was okay but have read sources on the net that say std::string is better...

It's a matter of opinion. Personally I prefer using std::string because it's

Re: CString to const char*

more standard compatible

Preferably I'd like to just use CString and convert it nicely...

Thanks

loads in advance!

IIRC, CString has no function to convert Unicode characters to ASCII ones. However you can use mbtowc() or MultiByteToWideChar() to do the job

BTW, you can use TCHAR instead of char and use your normal functions (MessageBox instead of MessageBoxA) and avoid all the hassle...

—

Abdo Haji-Ali
Programmer
In|Framez