

# C++ menu character string standard library string Header Annotations

Posted by uty - 2008/05/08 08:14

[http://msdn.microsoft.com/en-us/library/aa383701\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa383701(VS.85).aspx) Header Annotations Header annotations describe how a function uses its parameters and return value. These annotations have been added to many of the Windows header files to help you ensure that you are calling the Windows API correctly. If you enable code analysis, the Visual Studio 2005 compiler will produce level 6000 warnings if you are not calling these functions per the usage described through the annotations. You can also add these annotations in your own code to ensure that it is being called correctly. To enable code analysis in Visual Studio, go to the Project menu, click Properties, click Code Analysis, and enable the feature. For more information, see the documentation for Visual Studio 2005. These annotations are defined in Specstrings.h. They are built on primitives that are part of the Standard Annotation Language (SAL) and implemented using `_declspec( SAL_* )`. There are two classes of annotations: buffer annotations and advanced annotations. Buffer Annotations Buffer annotations describe how functions use their pointers and can be used to detect buffer overruns. Each parameter may use zero or one buffer annotation. A buffer annotation is constructed with a leading underscore and the components described in the following sections. Buffer size Description (size) Specifies the total size of the buffer. Use with `_bcount` and `_ecount`; do not use with `_part`. This value is the accessible space; it may be less than the allocated space. (size,length) Specifies the total size and initialized length of the buffer. Use with `_bcount_part` and `_ecount_part`. The total size may be less than the allocated space. Buffer size units Description `_bcount` The buffer size is in bytes. `_ecount` The buffer size is in elements. Direction Description `_in` The function reads from the buffer. The caller provides the buffer and initializes it. `_inout` The function both reads from and writes to buffer. The caller provides the buffer and initializes it. If used with `_deref`, the buffer may be reallocated by the function. `_out` The function writes to the buffer. If used on the return value or with `_deref`, the function provides the buffer and initializes it. Otherwise, the caller provides the buffer and the function initializes it. Indirection Description `_deref` Dereference the parameter to obtain the buffer pointer. This parameter may not be NULL. `_deref_opt` Dereference the parameter to obtain the buffer pointer. This parameter can be NULL. Initialization Description `_full` The function initializes the entire buffer. Use only with output buffers. `_part` The function initializes part of the buffer, and explicitly indicates how much. Use only with output buffers. Required or optional buffer Description `_opt` This parameter can be NULL. The following example shows the annotations for the `GetModuleFileName` function. The `hModule` parameter is an optional input parameter. The `lpFilename` parameter is an output parameter; its size in characters is specified by the `nSize` parameter and its length includes the null-terminating character. The `nSize` parameter is an input parameter. Copy CodeWORD WINAPI GetModuleFileName( `_in_opt` HMODULE hModule, `_out_ecount_part`(nSize, return + 1) LPTSTR lpFilename, `_in` DWORD nSize ); The following are the annotations defined in Specstrings.h. Use the information in the tables above to interpret their meaning. `__bcount`(size) `__bcount_opt`(size) `__deref_bcount`(size) `__deref_bcount_opt`(size) `__deref_ecount`(size) `__deref_ecount_opt`(size) `__deref_in` `__deref_in_bcount`(size) `__deref_in_bcount_opt`(size) `__deref_in_ecount`(size) `__deref_in_ecount_opt`(size) `__deref_in_opt` `__deref_inout` `__deref_inout_bcount`(size) `__deref_inout_bcount_full`(size) `__deref_inout_bcount_full_opt`(size) `__deref_inout_bcount_opt`(size) `__deref_inout_bcount_part`(size,length) `__deref_inout_bcount_part_opt`(size,length) `__deref_inout_ecount`(size) `__deref_inout_ecount_full`(size) `__deref_inout_ecount_full_opt`(size) `__deref_inout_ecount_opt`(size) `__deref_inout_ecount_part`(size,length) `__deref_inout_ecount_part_opt`(size,length) `__deref_inout_opt` `__deref_opt_bcount`(size) `__deref_opt_bcount_opt`(size) `__deref_opt_ecount`(size) `__deref_opt_ecount_opt`(size) `__deref_opt_in` `__deref_opt_in_bcount`(size) `__deref_opt_in_bcount_opt`(size) `__deref_opt_in_ecount`(size) `__deref_opt_in_ecount_opt`(size) `__deref_opt_in_opt` `__deref_opt_inout` `__deref_opt_inout_bcount`(size) `__deref_opt_inout_bcount_full`(size) `__deref_opt_inout_bcount_full_opt`(size) `__deref_opt_inout_bcount_opt`(size) `__deref_opt_inout_bcount_part`(size,length) `__deref_opt_inout_bcount_part_opt`(size,length) `__deref_opt_inout_ecount`(size) `__deref_opt_inout_ecount_full`(size) `__deref_opt_inout_ecount_full_opt`(size) `__deref_opt_inout_ecount_opt`(size) `__deref_opt_inout_ecount_part`(size,length) `__deref_opt_inout_ecount_part_opt`(size,length) `__deref_opt_inout_opt` `__deref_opt_out` `__deref_opt_out_bcount`(size) `__deref_opt_out_bcount_full`(size) `__deref_opt_out_bcount_full_opt`(size) `__deref_opt_out_bcount_opt`(size) `__deref_opt_out_bcount_part`(size,length) `__deref_opt_out_bcount_part_opt`(size,length) `__deref_opt_out_ecount`(size) `__deref_opt_out_ecount_full`(size) `__deref_opt_out_ecount_full_opt`(size) `__deref_opt_out_ecount_opt`(size) `__deref_opt_out_ecount_part`(size,length) `__deref_opt_out_ecount_part_opt`(size,length) `__deref_opt_out_opt` `__deref_out` `__deref_out_bcount`(size) `__deref_out_bcount_full`(size) `__deref_out_bcount_full_opt`(size) `__deref_out_bcount_opt`(size) `__deref_out_bcount_part`(size,length) `__deref_out_bcount_part_opt`(size,length) `__deref_out_ecount`(size) `__deref_out_ecount_full`(size) `__deref_out_ecount_full_opt`(size) `__deref_out_ecount_opt`(size) `__deref_out_ecount_part`(size,length) `__deref_out_ecount_part_opt`(size,length) `__deref_out_opt` `__ecount`(size) `__ecount_opt`(size) `__in` `__in_bcount`(size) `__in_bcount_opt`(size) `__in_ecount`(size) `__in_ecount_opt`(size) `__in_opt` `__inout` `__inout_bcount`(size) `__inout_bcount_full`(size) `__inout_bcount_full_opt`(size) `__inout_bcount_opt`(size) `__inout_bcount_part`(size,length) `__inout_bcount_part_opt`(size,length) `__inout_ecount`(size) `__inout_ecount_full`(size) `__inout_ecount_full_opt`(size) `__inout_ecount_opt`(size) `__inout_ecount_part`(size,length) `__inout_ecount_part_opt`(size,length) `__inout_opt` `__out` `__out_bcount`(size) `__out_bcount_full`(size) `__out_bcount_full_opt`(size) `__out_bcount_opt`(size) `__out_bcount_part`(size,length) `__out_bcount_part_opt`(size,length) `__out_ecount`(size) `__out_ecount_full`(size) `__out_ecount_full_opt`(size) `__out_ecount_opt`(size) `__out_ecount_part`(size,length) `__out_ecount_part_opt`(size,length) `__out_opt` Advanced Annotations Advanced annotations provide additional information about the parameter or return value. Each parameter

---

or return value may use zero or one advanced annotation. Annotation Description `__blocksOn(resource)` The function blocks on the specified resource. `__callback` The function can be used as a function pointer. `__checkReturn` Callers must check the return value. `__format_string` The parameter is a string that contains printf-style % markers. `__in_awcount(expr,size)` If the expression is true at exit, the size of the input buffer is specified in bytes. If the expression is false, the size is specified in elements. `__nullnullterminated` The buffer may be accessed up to and including the first sequence of two null characters or pointers. `__nullterminated` The buffer may be accessed up to and including the first null character or pointer. `__out_awcount(expr,size)` If the expression is true at exit, the size of the output buffer is specified in bytes. If the expression is false, the size is specified in elements. `__override` Specifies C#-style override behavior for virtual methods. `__reserved` The parameter is reserved for future use and must be zero or NULL. `__success(expr)` If the expression is true at exit, the caller can rely on all guarantees specified by other annotations. If the expression is false, the caller cannot rely on the guarantees. This annotation is automatically added to functions that return an HRESULT value. `__typefix ctype` Treat the parameter as the specified type rather than its declared type. The following examples show the buffer and advanced annotations for the `DeleteTimerQueueTimer`, `FreeEnvironmentStrings`, and `UnhandledExceptionFilter` functions. Copy Code `__checkReturn BOOL WINAPI DeleteTimerQueueTimer( __in_opt HANDLE TimerQueue, __in HANDLE Timer, __in_opt HANDLE CompletionEvent );` `__callback BOOL WINAPI FreeEnvironmentStrings( __in __nullnullterminated LPTCH );` `__callback LONG WINAPI UnhandledExceptionFilter( __in struct _EXCEPTION_POINTERS *ExceptionInfo );` See Also SAL Annotations Walkthrough: Analyzing C/C++ Code for Defects Send comments about this topic to Microsoft Build date: 3/27/2008

=====