About Error Handling

In general, mCtrl follows Win32API conventions in error handling. On error, the mCtrl functions set the error code with `SetLastError()` and then typically return a value indicating an error has occurred. In most cases this error indicator is 0, -1, `FALSE` or `NULL`, depending on the function and returned value's type.

If a function fails, any output parameters are undefined and you cannot rely on their value.

Also note that for the category of caller's programmatic errors (e.g. when application specifies an invalid value in a function parameter or invalid combination of multiple parameters; or when calling any function without having the particular mCtrl module initialized), mCtrl uses very pragmatic approach: Such error conditions are only checked if it is reasonably easy to do so. More checks can be enabled in the debug build of mCtrl.

About Strings

All strings held internally in the library are encoded in Unicode.

On interface level, `MCTRL.DLL` supports both Unicode and ANSI strings as well. If a function, message of a control or a structure uses string, there are usually two flavors of the entity: one for Unicode and the other one for the ANSI string (ending with "A"). When calling a function or sending a message, ANSI strings in parameters and structure members are converted to Unicode on input and Unicode to ANSI on output.

Identifiers of the Unicode flavor have the suffix `"-W"` and those of ANSI flavor have suffix `"-A"`, in the same way as Win32API does. The public headers also provide preprocessor macros without the suffix, as an alias for the one of the two depending whether `UNICODE` is defined or not.

Also for notifications sent by control to the application, the mCtrl follows the Windows common practice: Controls which may need to send a notification with any string data send `WM_NOTIFYFORMAT` message to its parent during their creation and after that they respect the parent's desire.

This means you may use `MCTRL.DLL` easily in Unicode-enabled application as well as in legacy ANSI applications.

About Initialization and Termination

mCtrl functionality is divided into several modules, each having its own public header file. Each GUI implementation control provided by `MCTRL.DLL` forms a separate module, as well as some other groups of relative functions.

Most modules need to be initialized before they can be used. For controls, the initialization routine registers the control's window class with `RegisterClass()`, and the termination function unregisters it with `UnregisterClass()`.

Note that for performance reasons mCtrl functions do not test whether the module is properly initialized, so the function can fail in any means if the module is not initialized, or can work if the particular function does not currently

rely on the initialization. But note that even in the latter case there is no guaranty the behavior does not change in future versions of mCtrl.

The initialization function can be always called multiple times (even concurrently in multiple threads). Each module has its own initialization counter, incremented in the initialization function and decremented in the termination function. The module is really uninitialized only after the counter drops back down to zero.

Attention

Note that if you are using `MCTRL.DLL` from within a DLL code, you may not call the initialization and termination functions in context of `DllMain()`. Windows severely limits what can be done safely in it. Even if it would be safe for some modules currently there is no guaranty that future version of mCtrl won't use anything problematic in this regard.

About Multi-threading

mCtrl is designed to be multi-threading friendly. In general, all functions are reentrant. I.e. you can call the same `MCTRL.DLL` function concurrently from multiple threads.

However remember that access to data visible externally through `MCTRL.DLL` interface is not synchronized: If you have such data (e.g. **MC_HTABLE**) and then want to manipulate with the data concurrently in context of multiple threads, `MCTRL.DLL` does not synchronize for you: It's application developer's responsibility to do so in order to avoid race conditions.

Also note that some mCtrl modules may include yet another limitations. Any such limitations are described in documentation of such the particular modules. (The **MC_WC_HTML** control is a prominent example of such limitation.)

Furthermore, some care is also needed if your application is multi-threaded and if it relies on any COM interface in any thread which may call into `MCTRL.DLL`. See **Compatibility with COM** for more information about this.

Compatibility with COM

Note

If you are sure that besides `MCTRL.DLL` your application does not use COM, then all you need to know is this: Everything is alright, it will just work.

In short, the COM subsystem needs to be initialized for any thread which wants to use it. The thread may be initialized in one of two so called apartments:

- Single-thread apartment: The thread itself has initialized explicitly with `CoInitializeEx()` and constant `COINIT_APARTMENTTHREADED`.
- Multi-thread apartment: The thread has not been initialized in single-threaded apartment and *any* thread has called `CoInitializeEx()` with `COINIT_MULTITHREADED` (or legacy `CoInitialize()`).

Note that for any GUI thread, Microsoft strongly recommends use of the single-threaded apartment.

As the apartment model is a global property of any thread, any DLL using COM has to be careful to not cause apartment model change which could conflict with the application.

To address the limitations, usage of COM in MCTRL.DLL follows these rules:

1. Any module or feature may or may not use COM as it sees fit. Even modules which do not yet do so today, may start to do so in any future version of mCtrl.
2. MCTRL.DLL uses COM in a way which is compatible both with single-threaded apartments as well as multi-threaded apartments.
3. The very first time, when MCTRL.DLL needs to use COM, it automatically detects whether the calling thread has COM subsystem already initialized. This check is used as a canary whether the application is COM-aware or not. If not, MCTRL.DLL remember it and it explicitly initializes COM (for a single-threaded apartment) in a context of any thread where it needs to use it. However if the application is COM-aware, MCTRL.DLL never attempts to initialize COM subsystem on its own.

Hence, any application which links with MCTRL.DLL, should strictly follow one of these usage patterns as of COM initialization:

1. Let MCTRL.DLL manage COM initialization: Application never performs any COM initialization of any thread which may call into MCTRL.DLL. This is perfectly usable for applications which (on their own or through any 3rd party DLL) never use COM at all.
2. The application itself manages COM initialization. This is required for applications which may use COM directly on their own or through any 3rd party DLL from any thread which also may call any mCtrl function. In this case, the application is responsible to initialize COM for each thread prior calling any mCtrl function in the given thread.

DLL Dependencies

The policy for mCtrl is to make hard dependencies only on DLLs available on on all supported system versions. This includes core system libraries like USER32.DLL, COMMCTRL32.DLL or GDI32.DLL.

However some mCtrl features may depend on use of additional DLLs, which are loaded in runtime. In such cases, mCtrl tries to behave in a reasonable manner even if those libraries or are not available:

- UXTHEME.DLL is used on Windows XP and newer to paint the mCtrl controls using a current visual theme. Note mCtrl does so only if the application uses COMMCTRL32.DLL version 6 or newer, to be visually consistent with the standard controls within the application.
- D2D1.DLL (Direct2d) and DWRITE.DLL (DirectWrite) are used for high quality painting, e.g. when anti-aliasing support or alpha channel is needed. Direct2D and DirectWrite are only available since MS Vista (with some Service Pack or other updates) and newer Windows versions.
- GDIPLUS.DLL (GDI+) is used as a fall back if Direct2d+DirectWrite are not available on the system. GDI+ is available since Windows 2000 with some Service Packs or updates installed.

Attention

To make MCTRL.DLL working on all Windows 2000, even when the system lacks the library, you may need to deploy the redistributable version of GDI+ v. 1.0 into your application directory, when installing your application. The redistributable GDIPLUS.DLL can be obtained from Microsoft.