LoadLibraryA function (libloaderapi.h) - Win32 apps

By karl-bridge-microsoft

Archived: 2026-04-02 12:19:01 UTC

Loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded.

For additional load options, use the LoadLibraryEx function.

Syntax

```
HMODULE LoadLibraryA(
  [in] LPCSTR lpLibFileName
);
```

Parameters

`[in] lpLibFileName`

The name of the module. This can be either a library module (a .dll file) or an executable module (an .exe file). If the specified module is an executable module, static imports are not loaded; instead, the module is loaded as if by LoadLibraryEx with the `DONT_RESOLVE_DLL_REFERENCES` flag.

The name specified is the file name of the module and is not related to the name stored in the library module itself, as specified by the **LIBRARY** keyword in the module-definition (.def) file.

If the string specifies a full path, the function searches only that path for the module.

If the string specifies a relative path or a module name without a path, the function uses a standard search strategy to find the module; for more information, see the Remarks.

If the function cannot find the module, the function fails. When specifying a path, be sure to use backslashes (\), not forward slashes (/). For more information about paths, see Naming a File or Directory.

If the string specifies a module name without a path and the file name extension is omitted, the function appends the default library extension ".DLL" to the module name. To prevent the function from appending ".DLL" to the module name, include a trailing point character (.) in the module name string.

Return value

If the function succeeds, the return value is a handle to the module.

If the function fails, the return value is NULL. To get extended error information, call GetLastError.

To enable or disable error messages displayed by the loader during DLL loads, use the SetErrorMode function.

LoadLibrary can be used to load a library module into the address space of the process and return a handle that can be used in GetProcAddress to get the address of a DLL function. **LoadLibrary** can also be used to load other executable modules. For example, the function can specify an .exe file to get a handle that can be used in FindResource or LoadResource. However, do not use **LoadLibrary** to run an .exe file. Instead, use the CreateProcess function.

If the specified module is a DLL that is not already loaded for the calling process, the system calls the DLL's DllMain function with the **DLL_PROCESS_ATTACH** value. If **DllMain** returns **TRUE**, **LoadLibrary** returns a handle to the module. If **DllMain** returns **FALSE**, the system unloads the DLL from the process address space and **LoadLibrary** returns **NULL**. It is not safe to call **LoadLibrary** from **DllMain**. For more information, see the Remarks section in **DllMain**.

Module handles are not global or inheritable. A call to **LoadLibrary** by one process does not produce a handle that another process can use — for example, in calling GetProcAddress. The other process must make its own call to **LoadLibrary** for the module before calling **GetProcAddress**.

If *lpFileName* does not include a path and there is more than one loaded module with the same base name and extension, the function returns a handle to the module that was loaded first.

If no file name extension is specified in the *lpFileName* parameter, the default library extension .dll is appended. However, the file name string can include a trailing point character (.) to indicate that the module name has no extension. When no path is specified, the function searches for loaded modules whose base name matches the base name of the module to be loaded. If the name matches, the load succeeds. Otherwise, the function searches for the file.

The first directory searched is the directory containing the image file used to create the calling process (for more information, see the CreateProcess function). Doing this allows private dynamic-link library (DLL) files associated with a process to be found without adding the process's installed directory to the PATH environment variable. If a relative path is specified, the entire relative path is appended to every token in the DLL search path list. To load a module from a relative path without searching any other path, use GetFullPathName to get a nonrelative path and call **LoadLibrary** with the nonrelative path. For more information on the DLL search order, see Dynamic-Link Library Search Order.

The search path can be altered using the SetDllDirectory function. This solution is recommended instead of using SetCurrentDirectory or hard-coding the full path to the DLL.

If a path is specified and there is a redirection file for the application, the function searches for the module in the application's directory. If the module exists in the application's directory, **LoadLibrary** ignores the specified path and loads the module from the application's directory. If the module does not exist in the application's directory, **LoadLibrary** loads the module from the specified directory. For more information, see Dynamic Link Library Redirection.

If you call **LoadLibrary** with the name of an assembly without a path specification and the assembly is listed in the system compatible manifest, the call is automatically redirected to the side-by-side assembly.

The system maintains a per-process reference count on all loaded modules. Calling **LoadLibrary** increments the reference count. Calling the FreeLibrary or FreeLibraryAndExitThread function decrements the reference count. The system unloads a module when its reference count reaches zero or when the process terminates (regardless of the reference count).

Windows Server 2003 and Windows XP: The Visual C++ compiler supports a syntax that enables you to declare thread-local variables: **_declspec(thread)**. If you use this syntax in a DLL, you will not be able to load the DLL explicitly using **LoadLibrary** on versions of Windows prior to Windows Vista. If your DLL will be loaded explicitly, you must use the thread local storage functions instead of **_declspec(thread)**. For an example, see Using Thread Local Storage in a Dynamic Link Library.

Do not use the SearchPath function to retrieve a path to a DLL for a subsequent **LoadLibrary** call. The **SearchPath** function uses a different search order than **LoadLibrary** and it does not use safe process search mode unless this is explicitly enabled by calling SetSearchPathMode with **BASE_SEARCH_PATH_ENABLE_SAFE_SEARCHMODE**. Therefore, **SearchPath** is likely to first search the user's current working directory for the specified DLL. If an attacker has copied a malicious version of a DLL into the current working directory, the path retrieved by **SearchPath** will point to the malicious DLL, which **LoadLibrary** will then load.

Do not make assumptions about the operating system version based on a **LoadLibrary** call that searches for a DLL. If the application is running in an environment where the DLL is legitimately not present but a malicious version of the DLL is in the search path, the malicious version of the DLL may be loaded. Instead, use the recommended techniques described in Getting the System Version.

Examples

For an example, see Using Run-Time Dynamic Linking.

Note

The libloaderapi.h header defines LoadLibrary as an alias that automatically selects the ANSI or Unicode version of this function based on the definition of the UNICODE preprocessor constant. Mixing usage of the encoding-neutral alias with code that is not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see Conventions for Function Prototypes.

Requirements

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Target Platform	Windows
Header	libloaderapi.h (include Windows.h)

Requirement	Value
Library	Kernel32.lib
DLL	Kernel32.dll

See also

DllMain

Dynamic-Link Library Functions

FindResource

FreeLibrary

GetProcAddress

GetSystemDirectory

GetWindowsDirectory

LoadLibraryEx

LoadResource

Run-Time Dynamic Linking

SetDllDirectory

SetErrorMode
