Función LoadLibraryA (libloaderapi.h)

Artículo • 27/07/20226 minutos para leer

Carga el módulo especificado en el espacio de direcciones del proceso de llamada. El módulo especificado puede hacer que se carguen otros módulos.

Para opciones de carga adicionales, use la función LoadLibraryEx .

Sintaxis

```
C++

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

Parámetros

`[in] lpLibFileName`

El nombre del módulo. Puede ser un módulo de biblioteca (un archivo .dll) o un módulo ejecutable (un archivo .exe). Si el módulo especificado es un módulo ejecutable, las importaciones estáticas no se cargan; en su lugar, el módulo se carga como si fuera LoadLibraryEx con la `DONT_RESOLVE_DLL_REFERENCES` bandera.

El nombre especificado es el nombre de archivo del módulo y no está relacionado con el nombre almacenado en el propio módulo de la biblioteca, como lo especifica la palabra clave **LIBRARY** en el archivo de definición del módulo (.def).

Si la cadena especifica una ruta completa, la función busca solo esa ruta para el módulo.

Si la cadena especifica una ruta relativa o un nombre de módulo sin ruta, la función usa una estrategia de búsqueda estándar para encontrar el módulo; para obtener más información, consulte las Observaciones.

Si la función no puede encontrar el módulo, la función falla. Al especificar una ruta, asegúrese de usar barras invertidas (\), no barras diagonales (/). Para obtener más información acerca de las rutas, consulte Nombrar un archivo o directorio .

Si la cadena especifica un nombre de módulo sin una ruta y se omite la extensión del nombre de archivo, la función agrega la extensión de biblioteca predeterminada ".DLL" al nombre del módulo. Para evitar que la función agregue ".DLL" al nombre del módulo, incluya un carácter de punto final (.) en la cadena del nombre del módulo.

Valor de retorno

Si la función tiene éxito, el valor devuelto es un identificador del módulo.

Si la función falla, el valor devuelto es NULL. Para obtener información de error ampliada, llame a GetLastError .

Observaciones

Para habilitar o deshabilitar los mensajes de error que muestra el cargador durante las cargas de DLL, use la función SetErrorMode .

LoadLibrary se puede usar para cargar un módulo de biblioteca en el espacio de direcciones del proceso y devolver un identificador que se puede usar en GetProcAddress para obtener la dirección de una función DLL. **LoadLibrary** también se puede utilizar para cargar otros módulos ejecutables. Por ejemplo, la función puede especificar un archivo .exe para obtener un identificador que se puede usar en FindResource o LoadResource . Sin embargo, no utilice **LoadLibrary** para ejecutar un archivo .exe. En su lugar, utilice la función CreateProcess .

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.

Security Remarks

No utilice la función SearchPath para recuperar una ruta a un archivo DLL para una llamada posterior a **LoadLibrary** . La función **SearchPath** usa un orden de búsqueda diferente al de **LoadLibrary** y no usa el modo de búsqueda de proceso seguro a menos que esté explícitamente habilitado llamando a SetSearchPathMode con **BASE_SEARCH_PATH_ENABLE_SAFE_SEARCHMODE** . Por lo tanto, es probable que **SearchPath** primero busque en el directorio de trabajo actual del usuario la DLL especificada. Si un atacante ha copiado una versión maliciosa de una DLL en el directorio de trabajo actual, la ruta recuperada por **SearchPath**apuntará a la DLL maliciosa, que luego cargará **LoadLibrary .**

No haga suposiciones sobre la versión del sistema operativo basándose en una llamada a **LoadLibrary** que busca una DLL. Si la aplicación se ejecuta en un entorno donde la DLL no está legítimamente presente pero hay una versión maliciosa de la DLL en la ruta de búsqueda, es posible que se cargue la versión maliciosa de la DLL. En su lugar, utilice las técnicas recomendadas que se describen en Obtención de la versión del sistema .

Ejemplos

Para ver un ejemplo, consulte Uso de enlaces dinámicos en tiempo de ejecución .

> ⓘ **Nota**
>
> The libloaderapi.h header defines LoadLibrary as an alias which 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 not encoding-neutral can lead to mismatches that result in compilation or runtime errors. For more information, see **Conventions for Function Prototypes**.

Requirements

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)
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