Dynamic-link library search order - Win32 apps

By stevewhims

Archived: 2026-04-05 17:05:49 UTC

It's common for multiple versions of the same dynamic-link library (DLL) to exist in different file system locations within an operating system (OS). You *can* control the specific location from which any given DLL is loaded by specifying a full path. But if you don't use that method, then the system searches for the DLL at load time as described in this topic. The *DLL loader* is the part of the operating system (OS) that loads DLLs and/or resolves references to DLLs.

Factors that affect searching

Here are some special search factors that are discussed in this topic—you can consider them to be part of the DLL search order. Later sections in this topic list these factors in the appropriate search order for certain app types, together with other search locations. This section is just to introduce the concepts, and to give them names that we'll use to refer to them later in the topic.

- **DLL redirection**. For details, see Dynamic-link library redirection.
- **API sets**. For details, see Windows API sets.
- **Side-by-side (SxS) manifest redirection**—desktop apps only (not UWP apps). You can redirect by using an application manifest (also known as a side-by-side application manifest, or a fusion manifest). For details, see Manifests.
- **Loaded-module list**. The system can check to see whether a DLL with the same module name is already loaded into memory (no matter which folder it was loaded from).
- **Known DLLs**. If the DLL is on the list of known DLLs for the version of Windows on which the application is running, then the system uses its copy of the known DLL (and the known DLL's dependent DLLs, if any). For a list of known DLLs on the current system, see the registry key `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs`.

If a DLL has dependencies, then the system searches for the dependent DLLs as if they were loaded by using only their module names. That's true even if the first DLL was loaded by specifying a full path.

Search order for packaged apps

When a packaged app loads a packaged module (specifically, a library module—a `.dll` file) by calling the **LoadPackagedLibrary** function, the DLL must be in the package dependency graph of the process. For more information, see **LoadPackagedLibrary**. When a packaged app loads a module by other means, and doesn't specify a full path, the system searches for the DLL and its dependencies at load time as described in this section.

When the system searches for a module or its dependencies, it always uses the search order for packaged apps; even if a dependency is not packaged app code.

Standard search order for packaged apps

The system searches in this order:

1. DLL redirection.
2. API sets.
3. **Desktop apps only (not UWP apps)**. SxS manifest redirection.
4. Loaded-module list.
5. Known DLLs.
6. The package dependency graph of the process. This is the application's package plus any dependencies specified as `<PackageDependency>` in the `<Dependencies>` section of the application's package manifest. Dependencies are searched in the order they appear in the manifest.
7. The folder the calling process was loaded from (the executable's folder).
8. The system folder (`%SystemRoot%\system32`).

If a DLL has dependencies, then the system searches for the dependent DLLs as if they were loaded with just their module names (even if the first DLL was loaded by specifying a full path).

Alternate search order for packaged apps

If a module changes the standard search order by calling the **LoadLibraryEx** function with **LOAD_WITH_ALTERED_SEARCH_PATH**, then the search order is the same as the standard search order except that in step 7 the system searches the folder that the specified module was loaded from (the top-loading module's folder) instead of the executable's folder.

Search order for unpackaged apps

When an unpackaged app loads a module and doesn't specify a full path, the system searches for the DLL at load time as described in this section.

Important

If an attacker gains control of one of the directories that's searched, then it can place a malicious copy of the DLL in that folder. For ways to help prevent such attacks, see Dynamic-link library security.

Standard search order for unpackaged apps

The standard DLL search order used by the system depends on whether or not *safe DLL search mode* is enabled.

Safe DLL search mode (which is enabled by default) moves the user's current folder later in the search order. To disable safe DLL search mode, create the `HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\SafeDllSearchMode` registry value, and set it to 0. Calling the **SetDllDirectory** function effectively disables safe DLL search mode (while the specified folder is in the search path), and changes the search order as described in this topic.

If safe DLL search mode is enabled, then the search order is as follows:

1. DLL Redirection.
2. API sets.
3. SxS manifest redirection.
4. Loaded-module list.
5. Known DLLs.
6. **Windows 11, version 21H2 (10.0; Build 22000), and later**. The package dependency graph of the process. This is the application's package plus any dependencies specified as `<PackageDependency>` in the `<Dependencies>` section of the application's package manifest. Dependencies are searched in the order they appear in the manifest.
7. The folder from which the application loaded.
8. The system folder. Use the **GetSystemDirectory** function to retrieve the path of this folder.
9. The 16-bit system folder. There's no function that obtains the path of this folder, but it is searched.
10. The Windows folder. Use the **GetWindowsDirectory** function to get the path of this folder.
11. The current folder.
12. The directories that are listed in the `PATH` environment variable. This doesn't include the per-application path specified by the **App Paths** registry key. The **App Paths** key isn't used when computing the DLL search path.

If safe DLL search mode is *disabled*, then the search order is the same except that *the current folder* moves from position 11 to position 8 in the sequence (immediately after step *7. The folder from which the application loaded*).

Alternate search order for unpackaged apps

To change the standard search order used by the system, you can call the **LoadLibraryEx** function with **LOAD_WITH_ALTERED_SEARCH_PATH**. You can also change the standard search order by calling the **SetDllDirectory** function.

Note

The standard search order of the process will also be affected by calling the **SetDllDirectory** function in the parent process before the start of the current process.

If you specify an alternate search strategy, then its behavior continues until all associated executable modules have been located. After the system starts processing DLL initialization routines, the system reverts to the standard search strategy.

The **LoadLibraryEx** function supports an alternate search order if the call specifies **LOAD_WITH_ALTERED_SEARCH_PATH**, and the *lpFileName* parameter specifies an absolute path.

- The standard search strategy begins (after the initial steps) in the calling application's folder.
- The alternate search strategy specified by **LoadLibraryEx** with **LOAD_WITH_ALTERED_SEARCH_PATH** begins (after the initial steps) in the folder of the executable module that **LoadLibraryEx** is loading.

That's the only way in which they differ.

If safe DLL search mode is enabled, then the alternate search order is as follows:

Steps 1-6 are the same as the standard search order.

7. The folder specified by *lpFileName*.
8. The system folder. Use the **GetSystemDirectory** function to retrieve the path of this folder.
9. The 16-bit system folder. There's no function that obtains the path of this folder, but it is searched.
10. The Windows folder. Use the **GetWindowsDirectory** function to get the path of this folder.
11. The current folder.
12. The directories that are listed in the `PATH` environment variable. This doesn't include the per-application path specified by the **App Paths** registry key. The **App Paths** key isn't used when computing the DLL search path.

If safe DLL search mode is *disabled*, then the alternate search order is the same except that *the current folder* moves from position 11 to position 8 in the sequence (immediately after step *7. The folder specified by lpFileName*).

The **SetDllDirectory** function supports an alternate search order if the *lpPathName* parameter specifies a path. The alternate search order is as follows:

Steps 1-6 are the same as the standard search order.

7. The folder from which the application loaded.
8. The folder specified by the *lpPathName* parameter of **SetDllDirectory**.
9. The system folder.
10. The 16-bit system folder.
11. The Windows folder.
12. The directories listed in the `PATH` environment variable.

If the *lpPathName* parameter is an empty string, then the call removes the current folder from the search order.

SetDllDirectory effectively disables safe DLL search mode while the specified folder is in the search path. To restore safe DLL search mode based on the **SafeDllSearchMode** registry value, and restore the current folder to the search order, call **SetDllDirectory** with *lpPathName* as NULL.

Search order using LOAD_LIBRARY_SEARCH flags

You can specify a search order by using one or more **LOAD_LIBRARY_SEARCH** flags with the **LoadLibraryEx** function. You can also use **LOAD_LIBRARY_SEARCH** flags with the **SetDefaultDllDirectories** function to establish a DLL search order for a process. You can specify additional directories for the process DLL search order by using the **AddDllDirectory** or **SetDllDirectory** functions.

The directories that are searched depend on the flags specified with **SetDefaultDllDirectories** or **LoadLibraryEx**. If you use more than one flag, then the corresponding directories are searched in this order:

1. **LOAD_LIBRARY_SEARCH_DLL_LOAD_DIR**. The folder that contains the DLL is searched. This folder is searched only for dependencies of the DLL to be loaded.

2. **LOAD_LIBRARY_SEARCH_APPLICATION_DIR**. The application folder is searched.
3. **LOAD_LIBRARY_SEARCH_USER_DIRS**. Paths explicitly added with the **AddDllDirectory** function or the **SetDllDirectory** function are searched. If you add more than one path, then the order in which the paths are searched is unspecified.
4. **LOAD_LIBRARY_SEARCH_SYSTEM32**. The System folder is searched.

If you call **LoadLibraryEx** with no **LOAD_LIBRARY_SEARCH** flags, or you establish a DLL search order for the process, then the system searches for DLLs using either the standard search order or the alternate search order.

- Application registration
- Dynamic-link library redirection
- Dynamic-link library security
- Side-by-side components
- AddDllDirectory
- LoadLibrary
- LoadLibraryEx
- LoadPackagedLibrary
- SetDefaultDllDirectories
- SetDllDirectory
