

CreateProcess Function

CreateProcess Function

Creates a new process and its primary thread. The new process runs in the security context of the calling process.

If the calling process is impersonating another user, the new process uses the token for the calling process, not the impersonation token. To run the new process in the security context of the user represented by the impersonation token, use the `CreateProcessAsUser` or `CreateProcessWithLogonW` function.

Add an item to Explorer Shell context menu easily – How to add them ?

Add an entry to Explorer Shell context menu easily with Windows Explorer Context Menu. This powerful .Net component for your own, custom items adding to Windows Explorer Shell context menu will add all your items to Windows Explorer context menu. This .Net component with full C# , C++ and VB.NET support include detailed C# / VB.NET samples, tutorials and support all you may need :

- Add all your items to Windows Explorer Shell context menu to be shown on any Windows computer (all OS are supported – Windows XP, Vista, x64 of all types , etc.)
- Add any type of items to Windows Explorer Shell context menu to be shown in any way - with your custom caption and icon, as separator or sub-menu
- Add items to Explorer context menu to be shown for all files or shown only for computer files of particular type (for example, only for .DOC , .MP3,.WMA,.AAC , .MPG files)
- Add items to Windows Explorer Shell context menu, sub-menus, sub-sub-menus, sub-menus of unlimited depth and even much more

Explorer Shell Context Menu - is a .Net component that support all you need to insert all your items to Windows Explorer Shell context menu - in a fast and a very easy way. Add your items to Explorer context menu right now – fast , easy and exactly as you prefer :

Introduction

`CreateProcess` - is a low-level Windows function that is used for new processes (applications) running and (in past) to add custom items to Windows Explorer Shell context menu, because this method could be used for custom items appending to Explorer context menu only for Windows 95 / Windows 98 (not on XP, Vista, x64 - 64-bit Windows), to add items to Windows Explorer context menu you should use, according to Microsoft guidelines, appropriate .Net component - Windows Explorer Context Menu. This developer - friendly .Net framework component will add items to Explorer context menu in a very easy way.

Syntax

```

BOOL WINAPI CreateProcess(
    __in_opt LPCTSTR lpApplicationName,
    __inout_opt LPTSTR lpCommandLine,
    __in_opt LPSECURITY_ATTRIBUTES lpProcessAttributes,
    __in_opt LPSECURITY_ATTRIBUTES lpThreadAttributes,
    __in BOOL bInheritHandles,
    __in DWORD dwCreationFlags,
    __in_opt LPVOID lpEnvironment,
    __in_opt LPCTSTR lpCurrentDirectory,
    __in LPSTARTUPINFO lpStartupInfo,
    __out LPPROCESS_INFORMATION lpProcessInformation
);

```

Parameters

lpApplicationName [in, optional]

The name of the module to be executed. This module can be a Windows-based application. It can be some other type of module (for example, MS-DOS or OS/2) if the appropriate subsystem is available on the local computer.

The string can specify the full path and file name of the module to execute or it can specify a partial name. In the case of a partial name, the function uses the current drive and current directory to complete the specification. The function will not use the search path. This parameter must include the file name extension; no default extension is assumed.

The lpApplicationName parameter can be NULL. In that case, the module name must be the first white space–delimited token in the lpCommandLine string. If you are using a long file name that contains a space, use quoted strings to indicate where the file name ends and the arguments begin; otherwise, the file name is ambiguous. For example, consider the string "c:\program files\sub dir\program name". This string can be interpreted in a number of ways. The system tries to interpret the possibilities in the following order:

```
c:\program.exe files\sub dir\program name
c:\program files\sub.exe dir\program name
c:\program files\sub dir\program.exe name
c:\program files\sub dir\program name.exe
```

If the executable module is a 16-bit application, lpApplicationName should be NULL, and the string pointed to by lpCommandLine should specify the executable module as well as its arguments.

To run a batch file, you must start the command interpreter; set lpApplicationName to cmd.exe and set lpCommandLine to the following arguments: /c plus the name of the batch file.

lpCommandLine [in, out, optional]

The command line to be executed. The maximum length of this string is 32,768 characters, including the Unicode terminating null character. If lpApplicationName is NULL, the module name portion of lpCommandLine is limited to MAX_PATH characters.

The Unicode version of this function, CreateProcessW, can modify the contents of this string. Therefore, this parameter cannot be a pointer to read-only memory (such as a const variable or a literal string). If this parameter is a constant string, the function may cause an access violation.

The lpCommandLine parameter can be NULL. In that case, the function uses the string pointed to by lpApplicationName as the command line.

If both lpApplicationName and lpCommandLine are non-NULL, the null-terminated string pointed to by lpApplicationName specifies the module to execute, and the null-terminated string pointed to by lpCommandLine specifies the command line. The new process can use GetCommandLine to retrieve the entire command line. Console processes written in C can use the argc and argv arguments to parse the command line. Because argv[0] is the module name, C programmers generally repeat the module name as the first token in the command line.

If lpApplicationName is NULL, the first white space–delimited token of the command line specifies the module name. If you are using a long file name that contains a space, use quoted strings to indicate where the file name ends and the arguments begin (see the explanation for the lpApplicationName parameter). If the file name does not contain an extension, .exe is appended. Therefore, if the file name extension is .com, this parameter must include the .com extension. If the file name ends in a period (.) with no extension, or if the file name contains a path, .exe is not appended. If the file name does not contain a directory path, the system searches for the executable file in the following sequence:

1. The directory from which the application loaded.
2. The current directory for the parent process.
3. The 32-bit Windows system directory. Use the GetSystemDirectory function to get the path of this directory.
4. The 16-bit Windows system directory. There is no function that obtains the path of this directory, but it is searched.

The name of this directory is System.

5. The Windows directory. Use the GetWindowsDirectory function to get the path of this directory.
6. The directories that are listed in the PATH environment variable. Note that this function does not search the per-application path specified by the App Paths registry key. To include this per-application path in the search sequence, use the ShellExecute function.

The system adds a terminating null character to the command-line string to separate the file name from the arguments.

This divides the original string into two strings for internal processing.
 lpProcessAttributes [in, optional]

A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle to the new process object can be inherited by child processes. If lpProcessAttributes is NULL, the handle cannot be inherited.

The lpSecurityDescriptor member of the structure specifies a security descriptor for the new process. If lpProcessAttributes is NULL or lpSecurityDescriptor is NULL, the process gets a default security descriptor. The ACLs in the default security descriptor for a process come from the primary token of the creator.

Windows XP/2000: The ACLs in the default security descriptor for a process come from the primary or impersonation token of the creator. This behavior changed with Windows XP SP2 and Windows Server 2003.

lpThreadAttributes [in, optional]

A pointer to a SECURITY_ATTRIBUTES structure that determines whether the returned handle to the new thread object can be inherited by child processes. If lpThreadAttributes is NULL, the handle cannot be inherited.

The lpSecurityDescriptor member of the structure specifies a security descriptor for the main thread. If lpThreadAttributes is NULL or lpSecurityDescriptor is NULL, the thread gets a default security descriptor. The ACLs in the default security descriptor for a thread come from the process token.

Windows XP/2000: The ACLs in the default security descriptor for a thread come from the primary or impersonation token of the creator. This behavior changed with Windows XP SP2 and Windows Server 2003.

blnheritHandles [in]

If this parameter TRUE, each inheritable handle in the calling process is inherited by the new process. If the parameter is FALSE, the handles are not inherited. Note that inherited handles have the same value and access rights as the original handles.

dwCreationFlags [in]

The flags that control the priority class and the creation of the process. For a list of values, see Process Creation Flags.

This parameter also controls the new process's priority class, which is used to determine the scheduling priorities of the process's threads. For a list of values, see GetPriorityClass. If none of the priority class flags is specified, the priority class defaults to NORMAL_PRIORITY_CLASS unless the priority class of the creating process is IDLE_PRIORITY_CLASS or BELOW_NORMAL_PRIORITY_CLASS. In this case, the child process receives the default priority class of the calling process.

lpEnvironment [in, optional]

A pointer to the environment block for the new process. If this parameter is NULL, the new process uses the environment of the calling process.

An environment block consists of a null-terminated block of null-terminated strings. Each string is in the following form:

name=value\0

Because the equal sign is used as a separator, it must not be used in the name of an environment variable.

An environment block can contain either Unicode or ANSI characters. If the environment block pointed to by lpEnvironment contains Unicode characters, be sure that dwCreationFlags includes CREATE_UNICODE_ENVIRONMENT. If this parameter is NULL and the environment block of the parent process contains Unicode characters, you must also ensure that dwCreationFlags includes CREATE_UNICODE_ENVIRONMENT.

The ANSI version of this function, CreateProcessA fails if the total size of the environment block for the process exceeds 32,767 characters.

Note that an ANSI environment block is terminated by two zero bytes: one for the last string, one more to terminate the block. A Unicode environment block is terminated by four zero bytes: two for the last string, two more to terminate the block.

lpCurrentDirectory [in, optional]

The full path to the current directory for the process. The string can also specify a UNC path.

If this parameter is NULL, the new process will have the same current drive and directory as the calling process. (This feature is provided primarily for shells that need to start an application and specify its initial drive and working directory.)
lpStartupInfo [in]

A pointer to a STARTUPINFO or STARTUPINFOEX structure.

To set extended attributes, use a STARTUPINFOEX structure and specify EXTENDED_STARTUPINFO_PRESENT in the dwCreationFlags parameter.
lpProcessInformation [out]

A pointer to a PROCESS_INFORMATION structure that receives identification information about the new process.

Handles in PROCESS_INFORMATION must be closed with CloseHandle when they are no longer needed.

Return Value

If the function succeeds, the return value is nonzero.

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

Remarks

The process is assigned a process identifier. The identifier is valid until the process terminates. It can be used to identify the process, or specified in the OpenProcess function to open a handle to the process. The initial thread in the process is also assigned a thread identifier. It can be specified in the OpenThread function to open a handle to the thread. The identifier is valid until the thread terminates and can be used to uniquely identify the thread within the system. These identifiers are returned in the PROCESS_INFORMATION structure.

The name of the executable in the command line that the operating system provides to a process is not necessarily identical to that in the command line that the calling process gives to the CreateProcess function. The operating system may prepend a fully qualified path to an executable name that is provided without a fully qualified path.

The calling thread can use the WaitForInputIdle function to wait until the new process has finished its initialization and is waiting for user input with no input pending. This can be useful for synchronization between parent and child processes, because CreateProcess returns without waiting for the new process to finish its initialization. For example, the creating process would use WaitForInputIdle before trying to find a window associated with the new process.

The preferred way to shut down a process is by using the ExitProcess function, because this function sends notification of approaching termination to all DLLs attached to the process. Other means of shutting down a process do not notify the attached DLLs. Note that when a thread calls ExitProcess, other threads of the process are terminated without an opportunity to execute any additional code (including the thread termination code of attached DLLs). For more information, see Terminating a Process.

A parent process can directly alter the environment variables of a child process during process creation. This is the only situation when a process can directly change the environment settings of another process. For more information, see Changing Environment Variables.

If an application provides an environment block, the current directory information of the system drives is not automatically propagated to the new process. For example, there is an environment variable named =C: whose value is the current directory on drive C. An application must manually pass the current directory information to the new process. To do so, the application must explicitly create these environment variable strings, sort them alphabetically (because the system uses a sorted environment), and put them into the environment block. Typically, they will go at the front of the environment block, due to the environment block sort order.

One way to obtain the current directory information for a drive X is to make the following call: GetFullPathName("X:", ...). That avoids an application having to scan the environment block. If the full path returned is X:\, there is no need to pass that value on as environment data, since the root directory is the default current directory for drive X of a new process.

When a process is created with CREATE_NEW_PROCESS_GROUP specified, an implicit call to SetConsoleCtrlHandler(NULL, TRUE) is made on behalf of the new process; this means that the new process has CTRL+C disabled. This lets shells handle CTRL+C themselves, and selectively pass that signal on to sub-processes. CTRL+BREAK is not disabled, and may be used to interrupt the process/process group.

Security Remarks

The first parameter, `lpApplicationName`, can be `NULL`, in which case the executable name must be in the white space–delimited string pointed to by `lpCommandLine`. If the executable or path name has a space in it, there is a risk that a different executable could be run because of the way the function parses spaces. The following example is dangerous because the function will attempt to run "Program.exe", if it exists, instead of "MyApp.exe".

```
LPTSTR szCmdline = _tcsdup(TEXT("C:\\Program Files\\MyApp -L -S"));
CreateProcess(NULL, szCmdline, ...)
```

If a malicious user were to create an application called "Program.exe" on a system, any program that incorrectly calls `CreateProcess` using the Program Files directory will run this application instead of the intended application.

To avoid this problem, do not pass `NULL` for `lpApplicationName`. If you do pass `NULL` for `lpApplicationName`, use quotation marks around the executable path in `lpCommandLine`, as shown in the example below.

```
LPTSTR szCmdline[] = _tcsdup(
    TEXT("\"C:\\Program Files\\MyApp\" -L -S"));
CreateProcess(NULL, szCmdline, ...)
```

Example Code

For an example, see [Creating Processes](#).

Declared in `Winbase.h`; include `Windows.h`.

Library

Use `Kernel32.lib`.

DLL

Requires `Kernel32.dll`.

Unicode/ANSI

Implemented as `CreateProcessW` (Unicode) and `CreateProcessA` (ANSI)