STARTUPINFOA structure (processthreadsapi.h)

Article11/01/2022

Specifies the window station, desktop, standard handles, and appearance of the main window for a process at creation time.

Syntax

C++

```cpp
typedef struct _STARTUPINFOA {
  DWORD  cb;
  LPSTR  lpReserved;
  LPSTR  lpDesktop;
  LPSTR  lpTitle;
  DWORD  dwX;
  DWORD  dwY;
  DWORD  dwXSize;
  DWORD  dwYSize;
  DWORD  dwXCountChars;
  DWORD  dwYCountChars;
  DWORD  dwFillAttribute;
  DWORD  dwFlags;
  WORD   wShowWindow;
  WORD   cbReserved2;
  LPBYTE lpReserved2;
  HANDLE hStdInput;
  HANDLE hStdOutput;
  HANDLE hStdError;
} STARTUPINFOA, *LPSTARTUPINFOA;
```

Members

`cb`

The size of the structure, in bytes.

`lpReserved`

Reserved; must be NULL.

`lpDesktop`

The name of the desktop, or the name of both the desktop and window station for this process. A backslash in the string indicates that the string includes both the desktop and window station names.

For more information, see Thread Connection to a Desktop.

`lpTitle`

For console processes, this is the title displayed in the title bar if a new console window is created. If NULL, the name of the executable file is used as the window title instead. This parameter must be NULL for GUI or console processes that do not create a new console window.

`dwX`

If **dwFlags** specifies STARTF_USEPOSITION, this member is the x offset of the upper left corner of a window if a new window is created, in pixels. Otherwise, this member is ignored.

The offset is from the upper left corner of the screen. For GUI processes, the specified position is used the first time the new process calls CreateWindow to create an overlapped window if the *x* parameter of **CreateWindow** is CW_USEDEFAULT.

`dwY`

If **dwFlags** specifies STARTF_USEPOSITION, this member is the y offset of the upper left corner of a window if a new window is created, in pixels. Otherwise, this member is ignored.

The offset is from the upper left corner of the screen. For GUI processes, the specified position is used the first time the new process calls CreateWindow to create an overlapped window if the *y* parameter of **CreateWindow** is CW_USEDEFAULT.

`dwXSize`

If **dwFlags** specifies STARTF_USESIZE, this member is the width of the window if a new window is created, in pixels. Otherwise, this member is ignored.

For GUI processes, this is used only the first time the new process calls CreateWindow to create an overlapped window if the *nWidth* parameter of **CreateWindow** is CW_USEDEFAULT.

`dwYSize`

If **dwFlags** specifies STARTF_USESIZE, this member is the height of the window if a new window is created, in pixels. Otherwise, this member is ignored.

For GUI processes, this is used only the first time the new process calls CreateWindow to create an overlapped window if the *nHeight* parameter of **CreateWindow** is CW_USEDEFAULT.

`dwXCountChars`

If **dwFlags** specifies STARTF_USECOUNTCHARS, if a new console window is created in a console process, this member specifies the screen buffer width, in character columns. Otherwise, this member is ignored.

`dwYCountChars`

If **dwFlags** specifies STARTF_USECOUNTCHARS, if a new console window is created in a console process, this member specifies the screen buffer height, in character rows. Otherwise, this member is ignored.

`dwFillAttribute`

If **dwFlags** specifies STARTF_USEFILLATTRIBUTE, this member is the initial text and background colors if a new console window is created in a console application. Otherwise, this member is ignored.

This value can be any combination of the following values: FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, and BACKGROUND_INTENSITY. For example, the following combination of values produces red text on a white background:

```
FOREGROUND_RED| BACKGROUND_RED| BACKGROUND_GREEN| BACKGROUND_BLUE
```

`dwFlags`

A bitfield that determines whether certain **STARTUPINFO** members are used when the process creates a window. This member can be one or more of the following values.

Expand table

Value	**Meaning**
STARTF_FORCEONFEEDBACK 0x00000040	Indicates that the cursor is in feedback mode for two seconds after CreateProcess is called. The Working in Background cursor is displayed (see the Pointers tab in the Mouse control panel utility). If during those two seconds the process makes the first GUI call, the system gives five more seconds to the process. If during those five seconds the process shows a window, the system gives five more seconds to the process to finish drawing the window. The system turns the feedback cursor off after the first call to GetMessage, regardless of whether the process is drawing.
STARTF_FORCEOFFFEEDBACK 0x00000080	Indicates that the feedback cursor is forced off while the process is starting. The Normal Select cursor is displayed.
STARTF_PREVENTPINNING 0x00002000	Indicates that any windows created by the process cannot be pinned on the taskbar. This flag must be combined with STARTF_TITLEISAPPID.
STARTF_RUNFULLSCREEN 0x00000020	Indicates that the process should be run in full-screen mode, rather than in windowed mode. This flag is only valid for console applications running on an x86 computer.
STARTF_TITLEISAPPID 0x00001000	The **lpTitle** member contains an AppUserModelID. This identifier controls how the taskbar and **Start** menu present the application, and enables it to be associated

	with the correct shortcuts and Jump Lists. Generally, applications will use the SetCurrentProcessExplicitAppUserModelID and **GetCurrentProcessExplicitAppUserModelID** functions instead of setting this flag. For more information, see Application User Model IDs. If STARTF_PREVENTPINNING is used, application windows cannot be pinned on the taskbar. The use of any AppUserModelID-related window properties by the application overrides this setting for that window only. This flag cannot be used with STARTF_TITLEISLINKNAME.
STARTF_TITLEISLINKNAME 0x00000800	The **lpTitle** member contains the path of the shortcut file (.lnk) that the user invoked to start this process. This is typically set by the shell when a .lnk file pointing to the launched application is invoked. Most applications will not need to set this value. This flag cannot be used with STARTF_TITLEISAPPID.
STARTF_UNTRUSTEDSOURCE 0x00008000	The command line came from an untrusted source. For more information, see Remarks.
STARTF_USECOUNTCHARS 0x00000008	The **dwXCountChars** and **dwYCountChars** members contain additional information.
STARTF_USEFILLATTRIBUTE 0x00000010	The **dwFillAttribute** member contains additional information.
STARTF_USEHOTKEY 0x00000200	The **hStdInput** member contains additional information. This flag cannot be used with **STARTF_USESTDHANDLES**.
STARTF_USEPOSITION 0x00000004	The **dwX** and **dwY** members contain additional information.
STARTF_USESHOWWINDOW 0x00000001	The **wShowWindow** member contains additional information.
STARTF_USESIZE	The **dwXSize** and **dwYSize** members contain additional

0x00000002	information.
STARTF_USESTDHANDLES 0x00000100	The **hStdInput**, **hStdOutput**, and **hStdError** members contain additional information. If this flag is specified when calling one of the process creation functions, the handles must be inheritable and the function's *bInheritHandles* parameter must be set to TRUE. For more information, see Handle Inheritance. If this flag is specified when calling the GetStartupInfo function, these members are either the handle value specified during process creation or INVALID_HANDLE_VALUE. Handles must be closed with CloseHandle when they are no longer needed. This flag cannot be used with **STARTF_USEHOTKEY**.

`wShowWindow`

If **dwFlags** specifies STARTF_USESHOWWINDOW, this member can be any of the values that can be specified in the *nCmdShow* parameter for the ShowWindow function, except for SW_SHOWDEFAULT. Otherwise, this member is ignored.

For GUI processes, the first time ShowWindow is called, its *nCmdShow* parameter is ignored **wShowWindow** specifies the default value. In subsequent calls to ShowWindow, the **wShowWindow** member is used if the *nCmdShow* parameter of **ShowWindow** is set to SW_SHOWDEFAULT.

`cbReserved2`

Reserved for use by the C Run-time; must be zero.

`lpReserved2`

Reserved for use by the C Run-time; must be NULL.

`hStdInput`

If **dwFlags** specifies STARTF_USESTDHANDLES, this member is the standard input handle for the process. If STARTF_USESTDHANDLES is not specified, the default for standard input is the keyboard buffer.

If **dwFlags** specifies STARTF_USEHOTKEY, this member specifies a hotkey value that is sent as the *wParam* parameter of a WM_SETHOTKEY message to the first eligible top-level window created by the application that owns the process. If the window is created with the WS_POPUP window style, it is not eligible unless the WS_EX_APPWINDOW extended window style is also set. For more information, see CreateWindowEx.

Otherwise, this member is ignored.

`hStdOutput`

If **dwFlags** specifies STARTF_USESTDHANDLES, this member is the standard output handle for the process. Otherwise, this member is ignored and the default for standard output is the console window's buffer.

If a process is launched from the taskbar or jump list, the system sets **hStdOutput** to a handle to the monitor that contains the taskbar or jump list used to launch the process. For more information, see Remarks.**Windows 7, Windows Server 2008 R2, Windows Vista, Windows Server 2008, Windows XP and Windows Server 2003:** This behavior was introduced in Windows 8 and Windows Server 2012.

`hStdError`

If **dwFlags** specifies STARTF_USESTDHANDLES, this member is the standard error handle for the process. Otherwise, this member is ignored and the default for standard error is the console window's buffer.

Remarks

For graphical user interface (GUI) processes, this information affects the first window created by the CreateWindow function and shown by the ShowWindow function. For console processes, this information affects the console window if a new console is created for the process. A process can use the GetStartupInfo function to retrieve the **STARTUPINFO** structure specified when the process was created.

If a GUI process is being started and neither STARTF_FORCEONFEEDBACK or

STARTF_FORCEOFFFEEDBACK is specified, the process feedback cursor is used. A GUI process is one whose subsystem is specified as "windows."

If a process is launched from the taskbar or jump list, the system sets GetStartupInfo to retrieve the **STARTUPINFO** structure and check that **hStdOutput** is set. If so, use GetMonitorInfo to check whether **hStdOutput** is a valid monitor handle (HMONITOR). The process can then use the handle to position its windows.

If the **STARTF_UNTRUSTEDSOURCE** flag is specified, the application should be aware that the command line is untrusted. If this flag is set, applications should disable potentially dangerous features such as macros, downloaded content, and automatic printing. This flag is optional, but applications that call CreateProcess are encouraged to set this flag when launching a program with untrusted command line arguments (such as those provided by web content) so that the newly created process can apply appropriate policy.

The **STARTF_UNTRUSTEDSOURCE** flag is supported starting in Windows Vista, but it is not defined in the SDK header files prior to the Windows 10 SDK. To use the flag in versions prior to Windows 10, you can define it manually in your program.

Examples

The following code example shows the use of **StartUpInfoA**.

C++

```cpp
#include <windows.h>
#include <stdio.h>
#include <tchar.h>

void _tmain( int argc, TCHAR *argv[] )
{
    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    ZeroMemory( &si, sizeof(si) );
    si.cb = sizeof(si);
    ZeroMemory( &pi, sizeof(pi) );

    if( argc != 2 )
    {
        printf("Usage: %s [cmdline]\n", argv[0]);
        return;
    }

    // Start the child process. 
    if( !CreateProcess( NULL,   // No module name (use command 
line)
        argv[1],        // Command line
        NULL,           // Process handle not inheritable
        NULL,           // Thread handle not inheritable
        FALSE,          // Set handle inheritance to FALSE
        0,              // No creation flags
        NULL,           // Use parent's environment block
        NULL,           // Use parent's starting directory 
        &si,            // Pointer to STARTUPINFO structure
        &pi )           // Pointer to PROCESS_INFORMATION structure
    ) 
    {
        printf( "CreateProcess failed (%d).\n", GetLastError() );
        return;
    }

    // Wait until child process exits.
    WaitForSingleObject( pi.hProcess, INFINITE );

    // Close process and thread handles. 
    CloseHandle( pi.hProcess );
    CloseHandle( pi.hThread );
}
```

For more information about this example, see Creating Processes.

ⓘ **Note**

The processthreadsapi.h header defines STARTUPINFO 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

Expand table

Requirement	Value
Minimum supported client	Windows XP [desktop apps only]
Minimum supported server	Windows Server 2003 [desktop apps only]
Header	processthreadsapi.h (include Windows.h on Windows Server 2003, Windows Vista, Windows 7, Windows Server 2008 Windows Server 2008 R2)

See also

CreateProcess

CreateProcessAsUser

CreateProcessWithLogonW

CreateProcessWithTokenW

GetStartupInfo