

Enhanced version of the CreateProcess function

Enhanced version of the CreateProcess function

In one of my programs, I needed to wait until an external application finished processing some data, and wanted to minimize the main window during that time to prevent the user from doing anything else before the results were available. That's why I wrote CreateProcessEx.

Add items to Explorer Context Menu easily - How?

Add items to Windows Explorer context menu easily with Windows Explorer Context Menu. This powerful .Net component for custom items appending to Windows Explorer Shell context menu will add all your custom application entries to Explorer context menu. It include detailed C# / VB.NET samples, tutorials and support all you may need:

- Add items to Windows Explorer context menu to be shown on any Windows OS (all OS are supported - XP, Vista, x64, etc.)
- Add items to Windows Explorer context menu to be shown in any way - with custom caption and icon, as separator or sub-menu
- Add items to Windows Explorer context menu to be shown for all files or shown only for files of particular type (for example, only for .DOC, .PDF, .TXT files)
- Add items to Windows Explorer context menu, sub-menus, sub-menus of unlimited depth

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

Here is presented enhanced description of CreateProcess OS-level function that was used in Windows operating systems for custom items appending to Explorer context menu from Windows 3.1 for Workgroups to Windows 98. Because this method works only for Windows 95 / Windows 98 (not on XP, Vista, x64 - 64-bit Windows), to add items to Windows Explorer Shell context menu you should use, according to Microsoft guidelines, appropriate .Net component - Windows Explorer Context Menu. This .Net component will add custom items of all types to Explorer context menu.

The CreateProcessEx function takes seven parameters but only the first two are required:

* lpAppName: Pointer to a null-terminated string that specifies the application to execute. The string can specify the full path and file name of the application to execute, or it can specify a partial name.

If lpAppName is NULL, the application name must be the first white space-delimited token in the lpCmdLine string.

* lpCmdLine: Pointer to a null-terminated string that specifies the command line to execute. This parameter can be NULL.

* bAppInCmdLine: Indicates if the lpAppName string must be used as the first white space-delimited token in the lpCmdLine string.

* bCompletePath: Indicates if the lpAppName string specifies a partial name that must be completed using the path of the calling process.

* bWaitForProcess: Indicates if the calling process must wait until child process exits.

* bMinimizeOnWait: Indicates if the main window will be minimized while the child process runs and automatically restored when it exits.

* hMainWnd: Handle of the main window to be minimized. If hMainWnd is NULL, then AfxGetMainWnd() is used.

Sample Code

```
DWORD CreateProcessEx ( LPCSTR lpAppPath, LPCSTR lpCmdLine,
    BOOL bAppInCmdLine, BOOL bCompletePath,
    BOOL bWaitForProcess, BOOL bMinimizeOnWait, HWND hMainWnd ) {
```

```

STARTUPINFO startupInfo;
PROCESS_INFORMATION processInformation;

char szAppPath [ _MAX_PATH ];
char szCmdLine [ _MAX_PATH ];
char drive [ _MAX_DRIVE ];
char dir [ _MAX_DIR ];
char name [ _MAX_FNAME ];
char ext [ _MAX_EXT ];

szAppPath[ 0 ] = '\0';
szCmdLine[ 0 ] = '\0';

ZeroMemory( &startupInfo, sizeof( STARTUPINFO ) );

startupInfo.cb = sizeof( STARTUPINFO );

ZeroMemory( &processInformation, sizeof( PROCESS_INFORMATION ) );

if ( bCompletePath ) {

    GetModuleFileName( GetModuleHandle( NULL ), szAppPath, _MAX_PATH );

    _splitpath( szAppPath, drive, dir, NULL, NULL );
    _splitpath( lpAppPath, NULL, NULL, name, ext );

    _makepath ( szAppPath, drive, dir, name, strlen( ext ) ? ext : ".exe" );
}
else strcpy( szAppPath, lpAppPath );

if ( bAppInCmdLine ) {

    strcpy( szCmdLine, "\"" );
    strcat( szCmdLine, szAppPath );
    strcat( szCmdLine, "\"" );
}

if ( lpCmdLine ) {

    // We must use quotation marks around the command line (if any)...

    if ( bAppInCmdLine ) strcat( szCmdLine, " \"" );
        else strcpy( szCmdLine, "\"" );

    strcat( szCmdLine, lpCmdLine );
    strcat( szCmdLine, "\"" );
}

DWORD dwExitCode = -1;

if ( CreateProcess( bAppInCmdLine ? NULL: szAppPath, // lpzImageName

                szCmdLine, // lpzCommandLine

                0, // lpProcess

                0, // lpThread

                FALSE, // flInheritHandles

                DETACHED_PROCESS, // fdwCreate

                0, // lpvEnvironment

```

```
        0,                // lpszCurDir
        &startupInfo,     // lpsiStartupInfo
        &processInformation // lppiProclInfo
    )) {
if ( bWaitForProcess ) {
    if ( bMinimizeOnWait )
        if ( IsWindow( hMainWnd )) ShowWindow( hMainWnd, SW_MINIMIZE );
        #ifdef __AFX_H__
        else AfxGetMainWnd()->ShowWindow( SW_MINIMIZE );
        #endif

    WaitForSingleObject( processInformation.hProcess, INFINITE );

    if ( bMinimizeOnWait )
        if ( IsWindow( hMainWnd )) ShowWindow( hMainWnd, SW_RESTORE );
        #ifdef __AFX_H__
        else AfxGetMainWnd()->ShowWindow( SW_RESTORE );
        #endif

    GetExitCodeProcess( processInformation.hProcess, &dwExitCode );
}
else {

    CloseHandle( processInformation.hThread );
    CloseHandle( processInformation.hProcess );

    dwExitCode = 0;
}
}

return dwExitCode;
}
```

To use the function, simply include the CreateProcessEx.h and CreateProcessEx.cpp files in your project. That's it!