

We use optional cookies to improve your experience on our websites, such as through social media connections, and to display personalized advertising based on your online activity. If you reject optional cookies, only cookies necessary to provide you the services will be used. You may change your selection by clicking “Manage Cookies” at the bottom of the page. [Privacy Statement](#) [Third-Party Cookies](#)

Accept


Reject

Manage cookies

Microsoft Ignite

Nov 19–22, 2024

Register now >

 | **Learn**

Discover ▾

Product documentation ▾

Development languages ▾

Topics ▾

🔍 Sign in

Windows App Development

Explore ▾

Development ▾

Platforms ▾

Troubleshooting

Resources ▾

Dashboard

 Filter by title

- ▾ Shell Developer's Guide
- Shell Developer's Guide
- Security Considerations: Microsoft Windows Shell
- Guidance for Implementing In-Process Extensions
- Developing with Windows Explorer
- Shell and Shlwapi DLL Versions
- > Implementing a Custom File Format
- > Shell Extensibility (Creating a Data Source)
- > Implementing Control Panel Items
- ▾ Supporting Shell Applications
- Supporting Shell Applications
- Launching Applications (ShellExecute, ShellExecuteEx, SHELLEXECUTEINFO)**
- > Dialogs
- > Windows Libraries
- > Known Folders
- Managing the File System
- Registering an Internet Browser or Email Client with the Windows Start Menu
- Registering Programs with Client Types
- Creating a Shell Link
- Using Autocomplete
- How to Enable Autocomplete Manually
- > Transferring Shell Objects with Drag-and-Drop and the Clipboard
- Scriptable Shell Objects
- > User Profiles
- > Navigating the Shell Namespace
- > Legacy Shell Topics
- > Shell Reference
- > Deprecated API
- > Shell Samples
- Shell Glossary

⋮ / Desktop Environment / Windows Shell /

⊕ ✎ ⋮

Launching Applications (ShellExecute, ShellExecuteEx, SHELLEXECUTEINFO)

Article • 06/15/2022 • 8 contributors

👍 Feedback

In this article

- Using ShellExecute and ShellExecuteEx
- A Simple Example of How to Use ShellExecuteEx

Once your application has located a file object, the next step is often to act on it in some way. For instance, your application might want to launch another application that allows the user to modify a data file. If the file of interest is an executable, your application might want to simply launch it. This document discusses how to use **ShellExecute** or **ShellExecuteEx** to perform these tasks.

- Using ShellExecute and ShellExecuteEx
 - Object Verbs
 - Using ShellExecuteEx to Provide Activation Services from a Site
 - Using ShellExecute to Launch the Search Dialog Box
- A Simple Example of How to Use ShellExecuteEx

Using ShellExecute and ShellExecuteEx

To use **ShellExecute** or **ShellExecuteEx**, your application must specify the file or folder object that is to be acted on, and a *verb* that specifies the operation. For **ShellExecute**, assign these values to the appropriate parameters. For **ShellExecuteEx**, fill in the appropriate members of a **SHELLEXECUTEINFO** structure. There are also several other members or parameters that can be used to fine-tune the behavior of the two functions.

File and folder objects can be part of the file system or virtual objects, and they can be identified by either paths or pointers to item identifier lists (PIDsLs).

Object Verbs

The verbs available for an object are essentially the items that you find on an object's shortcut menu. To find which verbs are available, look in the registry under

 Download PDF

HKEY_CLASSES_ROOT\CLSID\{*object_clsid*}\Shell\verb

where *object_clsid* is the class identifier (CLSID) of the object, and *verb* is the name of the available verb. The *verb*\command subkey contains the data indicating what happens when that verb is invoked.

To find out which verbs are available for [predefined Shell objects](#), look in the registry under

HKEY_CLASSES_ROOT\object_name\shell\verb

where *object_name* is the name of the predefined Shell object. Again, the *verb*\command subkey contains the data indicating what happens when that verb is invoked.

Commonly available verbs include:

 Expand table

Verb	Description
edit	Launches an editor and opens the document for editing.
find	Initiates a search starting from the specified directory.
open	Launches an application. If this file is not an executable file, its associated application is launched.
print	Prints the document file.
properties	Displays the object's properties.
runas	Launches an application as Administrator. User Account Control (UAC) will prompt the user for consent to run the application elevated or enter the credentials of an administrator account used to run the application.

Each verb corresponds to the command that would be used to launch the application from a console window. The **open** verb is a good example, as it is commonly supported. For .exe files, **open** simply launches the application. However, it is more commonly used to launch an application that operates on a particular file. For instance, .txt files can be opened by Microsoft WordPad. The **open** verb for a .txt file would thus correspond to something like the following command:

C++ Copy

```
"C:\Program Files\Windows NT\Accessories\Wordpad.exe" "%1"
```

When you use [ShellExecute](#) or [ShellExecuteEx](#) to open a .txt file, Wordpad.exe is launched with the specified file as its argument. Some commands can have additional arguments, such as flags, that can be added as needed to launch the application properly. For further discussion of shortcut menus and verbs, see [Extending Shortcut Menus](#).

In general, trying to determine the list of available verbs for a particular file is somewhat complicated. In many cases, you can simply set the *lpVerb* parameter to **NULL**, which invokes the default command for the file type. This procedure is usually equivalent to setting *lpVerb* to "open", but some file types may have a different default command. For further information, see [Extending Shortcut Menus](#) and the [ShellExecuteEx](#) reference documentation.

Using ShellExecuteEx to Provide Activation Services from a Site

A site chain's services can control many behaviors of item activation. As of Windows 8, you can provide a pointer to the site chain to [ShellExecuteEx](#) to enable these behaviors. To provide the site to [ShellExecuteEx](#):

- Specify the SEE_MASK_FLAG_HINST_IS_SITE flag in the **fMask** member of [SHELLEXECUTEINFO](#).
- Provide the **IUnknown** in the **hInstApp** member of [SHELLEXECUTEINFO](#).

Using ShellExecute to Launch the Search Dialog Box

When a user right-clicks a folder icon in Windows Explorer, one of the menu items is "Search". If they select that item, the Shell launches its Search utility. This utility displays a dialog box that can be used to search files for a specified text string. An application can programmatically launch the Search utility for a directory by calling [ShellExecute](#), with "find" as the *lpVerb* parameter, and the directory path as the *lpFile* parameter. For instance, the following line of code launches the Search utility for the c:\MyPrograms directory.

C++Copy

```
ShellExecute(hwnd, "find", "c:\\MyPrograms", NULL, NULL, 0);
```

A Simple Example of How to Use ShellExecuteEx

The following sample console application illustrates the use of [ShellExecuteEx](#). Most error checking code has been omitted for clarity.

C++Copy

```
#include <shlobj.h>
#include <shlwapi.h>
#include <objbase.h>

main()
{
    LPITEMIDLIST pidlWinFiles = NULL;
    LPITEMIDLIST pidlItems = NULL;
    IShellFolder *psfWinFiles = NULL;
    IShellFolder *psfDesktop = NULL;
    LPENUMIDLIST ppenum = NULL;
    STRRET strDispName;
    TCHAR pszParseName[MAX_PATH];
    ULONG celtFetched;
    SHELLEXECUTEINFO ShExecInfo;
    HRESULT hr;
    BOOL fBitmap = FALSE;

    hr = SHGetFolderLocation(NULL, CSIDL_WINDOWS, NULL, NULL, &pidlWinFiles);

    hr = SHGetDesktopFolder(&psfDesktop);

    hr = psfDesktop->BindToObject(pidlWinFiles, NULL, IID_IShellFolder, (LPVOID)0, &psfWinFiles);
    hr = psfDesktop->Release();

    hr = psfWinFiles->EnumObjects(NULL, SHCONTF_FOLDERS | SHCONTF_NONFOLDERS, &ppenum);

    while( hr = ppenum->Next(1,&pidlItems, &celtFetched) == S_OK && (celtFetched > 0) )
    {
        psfWinFiles->GetDisplayNameOf(pidlItems, SHGDN_FORPARSING, &strDispName);
        StrRetToBuf(&strDispName, pidlItems, pszParseName, MAX_PATH);
        CoTaskMemFree(pidlItems);
        if(StrCmpI(PathFindExtension(pszParseName), TEXT( ".bmp")) == 0)
        {
            fBitmap = TRUE;
            break;
        }
    }

    ppenum->Release();
```

```
if(fBitmap)
{
    ShExecInfo.cbSize = sizeof(SHELLEXECUTEINFO);
    ShExecInfo.fMask = NULL;
    ShExecInfo.hwnd = NULL;
    ShExecInfo.lpVerb = NULL;
    ShExecInfo.lpFile = pszParseName;
    ShExecInfo.lpParameters = NULL;
    ShExecInfo.lpDirectory = NULL;
    ShExecInfo.nShow = SW_MAXIMIZE;
    ShExecInfo.hInstApp = NULL;

    ShellExecuteEx(&ShExecInfo);
}

CoTaskMemFree(pidlWinFiles);
psfWinFiles->Release();

return 0;
}
```

The application first retrieves the PIDL of the Windows directory, and enumerates its contents until it finds the first .bmp file. Unlike the earlier example, [IShellFolder::GetDisplayNameOf](#) is used to retrieve the file's parsing name instead of its display name. Because this is a file system folder, the parsing name is a fully qualified path, which is what is needed for [ShellExecuteEx](#).

Once the first .bmp file has been located, appropriate values are assigned to the members of a [SHELLEXECUTEINFO](#) structure. The **lpFile** member is set to the parsing name of the file, and the **lpVerb** member to **NULL**, to begin the default operation. In this case, the default operation is "open". The structure is then passed to [ShellExecuteEx](#), which launches the default handler for bitmap files, typically MSPaint.exe, to open the file. After the function returns, the PIDLs are freed and the Windows folder's [IShellFolder](#) interface is released.

Feedback

Was this page helpful?

👍 Yes

👎 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Additional resources

Training

Module
[Work with files and directories in a .NET app - Training](#)
Learn how to use .NET, C#, and System.IO to work with directories, paths, files, and the file system.

Events

Nov 20, 12 AM - Nov 22, 12 AM
Gain the competitive edge you need with powerful AI and Cloud solutions by attending Microsoft Ignite online.
[Register now](#)

