

# Creating Shell Extension Handlers

## Creating Shell Extension Handlers

The capabilities of the Shell can be extended with registry entries and .ini files. While this approach to extending the Shell is simple, and adequate for many purposes, it is limited. For example, if you use the registry to specify a custom icon for a file class, the same icon will appear for every file in that class. Extending the Shell with the registry does not allow you to vary the icon for different members of the class. Other aspects of the Shell, such as the Properties property sheet that can be displayed when a file is right-clicked, cannot be modified at all with the registry.

Add items to Explorer context (right-click) menu easily &ndash; How to add ?

Add entries to Windows Explorer Shell context menu easily with Windows Explorer Shell Context Menu. This powerful .Net component for custom items adding to Windows Explorer Shell context menu will add all your custom application items to the 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 OS (all OS are supported &ndash; 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 Windows Explorer Shell context menu to be shown for all types of files or shown only for files of particular type (for example, only for .PDF .TXT , .MP3,.WMA,.AAC , .WMV files)
- Add items to Windows Explorer Shell context menu, sub-menus, sub-menus of unlimited depth and add to Explorer context menu entries of all types

Windows Explorer Shell Context Menu - is a powerful .Net component that support all you may need to add all your application items to the Windows Explorer Shell context menu - in a fast and easy way. Add all your application entries to Explorer context menu right now &ndash; add items to context menu fast , easy and exactly as you want :

Article below tells about shell extension handlers used in old Windows systems, because this they work without issues 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 - Explorer Context Menu. It will help to add your items to Explorer context menu - fast and easy.

A more powerful and flexible approach to extending the Shell is to implement shell extension handlers. These handlers can be implemented for a variety of actions that the Shell can perform. Before taking the action, the Shell queries the extension handler, giving it the opportunity to modify the action. A common example is a shortcut menu extension handler. If one is implemented for a file class, it will be queried every time one of the files is right-clicked. The handler can then specify additional menu items on a file-by-file basis, rather than having the same set for the entire file class.

This document discusses how to implement the extension handlers that allow you to modify a variety of Shell actions. The following handlers are associated with a particular file class and allow you to specify on a file-by-file basis:

Shortcut menu handler - Called before a file's shortcut menu is displayed. It enables you to add items to the shortcut menu on a file-by-file basis.

Data handler - Called when a drag-and-drop operation is performed on dragShell objects. It enables you to provide additional clipboard formats to the drop target.

Drop handler - Called when a data object is dragged over or dropped on a file. It enables you to make a file into a drop target.

Icon handler - Called before a file's icon is displayed. It enables you to replace the file's default icon with a custom icon on a file-by-file basis.

Property sheet handler - Called before an object's Properties property sheet is displayed. It enables you to add or replace pages.

Thumbnail Image handler - Provides an image to represent the item.

Infotip handler - Provides pop-up text when the user hovers the mouse pointer over the object.

Metadata handler - Provides read and write access to metadata (properties) stored in a file. This can be used to extend the Details view, infotips, the property page, and grouping features.

Other handlers are not associated with a particular file class but are called before some Shell operations:

Handler Description

Column handler Called by Microsoft Windows Explorer before it displays the Details view of a folder. It enables you to add custom columns to the Details view.

Copy hook handler Called when a folder or printer object is about to be moved, copied, deleted, or renamed. It enables you to approve or veto the operation.

Drag-and-drop handler Called when a file is dragged with the right mouse button. It enables you to modify the shortcut menu that is displayed.

Icon Overlay handler Called before a file's icon is displayed. It enables you to specify an overlay for the file's icon.

Search handler Called to launch a search engine. It enables you to implement a custom search engine accessible from the Start menu or Windows Explorer.

The details of how to implement specific extension handlers are covered in the sections listed above. The remainder of this document covers some implementation issues that are common to all Shell extension handlers.

- \* Implementing Shell Extension Handlers

- \* Registering Shell Extension Handlers

## Implementing Shell Extension Handlers

Much of the implementation of a Shell extension handler object depends on its type. There are, however, some common elements. This section discusses those aspects of implementation that are shared by all Shell extension handlers.

All Shell extension handlers are in-process Component Object Model (COM) objects. They must be assigned a GUID and registered as described in Registering Shell Extension Handlers. They are implemented as DLLs and must export the following standard functions:

- \* DllMain. The standard entry point to the DLL.

- \* DllGetClassObject. Exposes the object's class factory.

- \* DllCanUnloadNow. COM calls this function to determine whether the object is serving any clients. If not, the system can unload the DLL and free the associated memory.

Like all COM objects, Shell extension handlers must implement an IUnknown interface and a class factory. Most extension handlers must also implement either an IPersistFile or IShellExtInit interface in Windows XP or earlier. These were replaced by IInitializeWithStream, IInitializeWithItem and IInitializeWithFile in Windows Vista. The Shell uses these interfaces to initialize the handler.

The IPersistFile interface must be implemented by the following:

- \* Data handlers

- \* Drop handlers

In the past, icon handlers were also required to implement IPersistFile, but this is no longer true. For icon handlers, IPersistFile is now optional and other interfaces such as IInitializeWithItem are preferred.

The IShellExtInit interface must be implemented by the following:

- \* Shortcut menu handlers

- \* Drag-and-drop handlers

- \* Property sheet handlers

## Implementing IPersistFile

The IPersistFile interface is intended to permit an object to be loaded from or saved to a disk file. It has six methods in addition to IUnknown, five of its own, and the GetClassID method that it inherits from IPersist. With Shell extensions, IPersist is used only to initialize a Shell extension handler object. Because there is typically no need to read from or write to the disk, only the GetClassID and Load methods require a nontoken implementation.

The Shell calls `GetClassID` first, and the function returns the class identifier (CLSID) of the extension handler object. The Shell then calls `Load` and passes in two values. The first, `pszFileName`, is a Unicode string with the name of the file or folder that Shell is about to operate on. The second is `dwMode`, which indicates the file access mode. Because there is typically no need to access files, `dwMode` is usually zero. The method stores these values as needed for later reference.

The following code fragment illustrates how a typical Shell extension handler implements the `GetClassID` and `Load` methods. It is designed to handle either ANSI or Unicode. `CLSID_SampleExtHandler` is the extension handler object's GUID, and `CSampleExtHandler` is the name of the class used to implement the interface. The `m_szFileName` and `m_dwMode` variables are private variables that are used to store the file's name and access flags.

Copy Code

```
TCHAR m_szFileName[MAX_PATH]; // The file name
DWORD m_dwMode; // The file access mode

CSampleExtHandler::GetClassID(CLSID *pCLSID)
{
    *pCLSID = CLSID_SampleExtHandler;
}

CSampleExtHandler::Load(PCWSTR pszFile, DWORD dwMode)
{
    m_dwMode = dwMode;
    return StringCchCopy(_szFileName, ARRAYSIZE(m_szFileName), pszFile);
}
```

### Implementing IShellExtInit

The `IShellExtInit` interface has only one method, `Initialize`, in addition to `IUnknown`. The method has three parameters that the Shell can use to pass in various types of information. The values passed in depend on the type of handler, and some can be set to `NULL`.

- \* `pIDFolder` holds a folder's pointer to an item identifier list (PIDL). For property sheet extensions, it is `NULL`. For shortcut menu extensions, it is the PIDL of the folder that contains the item whose shortcut menu is being displayed. For nondefault drag-and-drop handlers, it is the PIDL of the target folder.

- \* `pDataObject` holds a pointer to a data object's `IDataObject` interface. The data object holds one or more file names in `CF_HDROP` format.

- \* `hRegKey` holds a registry key for the file object or folder type.

The `Initialize` method stores the file name, `IDataObject` pointer, and registry key as needed for later use. The following code fragment illustrates an implementation of `Initialize`. For simplicity, this example assumes that the data object contains only a single file. In general, it might contain multiple files that will each need to be extracted.

Copy Code

```
LPCITEMIDLIST m_pIDFolder; //The folder's PIDL
TCHAR m_szFile[MAX_PATH]; //The file name
IDataObject *m_pDataObj; //The IDataObject pointer
HKEY m_hRegKey; //The file or folder's registry key

STDMETHODIMP CShellExt::Initialize(LPCITEMIDLIST pIDFolder,
    IDataObject *pDataObj,
    HKEY hRegKey)
{
    // If Initialize has already been called, release the old PIDL
    IIFree(m_pIDFolder);
    m_pIDFolder = NULL;

    //Store the new PIDL.
    if(pIDFolder)
    {
        m_pIDFolder = ILCClone(pIDFolder);
    }

    // If Initialize has already been called, release the old
    // IDataObject pointer.
```

```

if (m_pDataObj)
{
    m_pDataObj->Release();
}

// If a data object pointer was passed in, save it and
// extract the file name.
if (pDataObj)
{
    m_pDataObj = pDataObj;
    pDataObj->AddRef();

    STGMEDIUM medium;
    FORMATETC fe = {CF_HDROP, NULL, DVASPECT_CONTENT, -1, TYMED_HGLOBAL};
    UINT uCount;

    if(SUCCEEDED(m_pDataObj->GetData(&fe, &medium)))
    {
        // Get the count of files dropped.
        uCount = DragQueryFile((HDROP)medium.hGlobal, (UINT)-1, NULL, 0);

        // Get the first file name from the CF_HDROP.
        if(uCount)
            DragQueryFile((HDROP)medium.hGlobal, 0, m_szFile,
                sizeof(m_szFile)/sizeof(TCHAR));

        ReleaseStgMedium(&medium);
    }
}

// Duplicate the registry handle.
if (hRegKey)
    RegOpenKeyEx(hRegKey,
        NULL,
        0L,
        MAXIMUM_ALLOWED,
        &m_hRegKey);
return S_OK;
}

```

CSampleExtHandler is the name of the class used to implement the interface. The m\_pIDFolder, m\_pDataObject, m\_szFileName, and m\_hRegKey variables are private variables used to store the information that is passed in. For simplicity, this example assumes that only one file name will be held by the data object. After the FORMATETC structure is retrieved from the data object, DragQueryFile is used to extract the file name from the FORMATETC structure's medium.hGlobal member. If a registry key is passed in, the method uses RegOpenKeyEx to open the key and assigns the handle to m\_hRegKey.

Infotip Customization

There are two ways to customize infotips:

- \* Implement an object that supports IQueryInfo and then register that object under the proper subkey in the registry (see Registering Shell Extension Handlers below).

- \* Specify a fixed string or a list of specific file properties to be displayed.

To display a fixed string for a namespace extension, create an entry called InfoTip in the {CLSID} key of your namespace extension. Set the value of that entry to be either the literal string you want to display, as shown in this example, or an indirect string that specifies a resource and index within that resource (for localization purposes).

```

* HKEY_CLASSES_ROOT
  o CLSID
    + {CLSID}

```

InfoTip = InfoTip string for your namespace extension

To display a fixed string for a file type, create an entry called InfoTip in the ProgID key of that file type. Set the value of that entry to be either the literal string you want to display or an indirect string that specifies a resource and index within that resource (for localization purposes), as shown in this example.

```
* HKEY_CLASSES_ROOT
  o ProgID

    InfoTip = Resource.dll, 3
```

If you want the Shell to display specific file properties in the infotip for a specific file type, create an entry called InfoTip in the ProgID key for that file type. Set the value of that entry to be a semicolon-delineated list of canonical property names, format identifier (FMTID)/property identifier (PID) pairs, or both. This value must begin with "prop:" to identify it as a property list string. If you omit "prop:", the value is seen as a literal string and displayed as such.

In the following example, propName is a canonical property name (such as System.Date) and {fmtid},pid is an FMTID/PID pair.

```
* HKEY_CLASSES_ROOT
  o ProgID

    InfoTip = prop:propname;propname;{fmtid},pid;{fmtid},pid
```

The following property names can be used:

Property Name	Description	Retrieved From
Author	Author of the document	OLE document properties (PIDS_I_AUTHOR)
Title	Title of the document	OLE document properties (PIDS_I_TITLE)
Subject	Subject summary	OLE document properties (PIDS_I_SUBJECT)
Comment	Document comments	OLE document properties (PIDS_I_COMMENT) or folder/drive properties
PageCount	Number of pages	OLE document properties (PIDS_I_PAGECOUNT)
Name	Friendly name	Standard folder view
OriginalLocation	Location of original file	Briefcase folder and Recycle Bin folder
DateDeleted	Date file was deleted	Recycle Bin folder
Type	Type of file	Standard folder details view
Size	Size of file	Standard folder details view
SyncCopyIn	Same as OriginalLocation	Same as OriginalLocation
Modified	Date last modified	Standard folder details view
Created	Date created	Standard folder details view
Accessed	Date last accessed	Standard folder details view
InFolder	Directory containing the file	Document search results
Rank	Quality of search match	Document search results
FreeSpace	Available storage space	Disk drives
NumberOfVisits	Number of visits	Favorites folder
Attributes	File Attributes	Standard folder details view
Company	Company name	OLE document properties (PIDDSI_COMPANY)
Category	Document category	OLE document properties (PIDDSI_CATEGORY)
Copyright	Media copyright	OLE document properties (PIDMSI_COPYRIGHT)
HTMLInfoTipFile	HTML InfoTip file	Desktop.ini file for folder

Registering Shell Extension Handlers

A Shell extension handler object must be registered before the Shell can use it. This section is a general discussion of how to register a Shell extension handler.

Any time you create or change a Shell extension handler, it is important to notify the system that you have made a change with SHChangeNotify, specifying the SHCNE\_ASSOCCHANGED event. If you do not call SHChangeNotify, the change might not be recognized until the system is rebooted.

There are some additional factors that apply to Microsoft Windows NT and Windows 2000 systems. For details, see Registering Shell Extension Handlers on Windows NT and Windows 2000 Systems.

As with all COM objects, you must create a GUID for the handler using a tool such as UUIDGEN.exe. Create a key under HKEY\_CLASSES\_ROOT\CLSID whose name is the string form of the GUID. Because Shell extension handlers are in-process servers, you must create an InProcServer32 key under the GUID key with the default value set to the path of the handler's DLL. Use the Apartment threading model.

Any time the Shell takes an action that can involve a Shell extension handler, it checks the appropriate registry key. The

key under which an extension handler is registered thus controls when it will be called. For instance, it is a common practice to have a shortcut menu handler called when the Shell displays a shortcut menu for a member of a file class. In this case, the handler must be registered under the file class's ProgID key.

#### Handler Names

To enable a Shell extension handler, create a subkey with the handler subkey name (see below) under the ShellEx subkey of either the ProgID (for file classes) or the Shell object type name (for Predefined Shell Objects).

For example, if you wanted to register a shortcut menu extension handler for MyProgram.1, you would begin by creating the following subkey:

```
* HKEY_CLASSES_ROOT
  o MyProgram.1
    + ShellEx
      # ContextMenuHandlers
```

For the following handlers, create a subkey underneath the "Handler Subkey Name" key whose name is the string version of the CLSID of the Shell extension. Multiple extensions can be registered under the handler subkey name key by creating multiple subkeys. (For backward compatibility with Windows 95, you can also create a key under the handler subkey name key with a descriptive name, and store the string version of the CLSID of the Shell extension as the default value of the key. This older method of registering Shell extensions is deprecated, but is described here for comprehensiveness.)

Handler	Interface	Handler Subkey Name
Column provider handler	IColumnProvider	ColumnHandlers
Shortcut menu handler	IContextMenu	ContextMenuHandlers
Copyhook handler	ICopyHook	CopyHookHandlers
Drag-and-drop handler	IContextMenu	DragDropHandlers
Property sheet handler	IShellPropSheetExt	PropertySheetHandlers

For the following handlers, the default value of the "Handler Subkey Name" key is the string version of the CLSID of the Shell extension. Only one extension can be registered for these handlers.

Handler	Interface	Handler Subkey Name
Data handler	IDataObject	DataHandler
Drop handler	IDropTarget	DropHandler
Icon handler	IExtractIconA/W	IconHandler
Thumbnail image handler	IThumbnailProvider	{E357FCCD-A995-4576-B01F-234630154E96}
Infotip handler	IQueryInfo	{00021500-0000-0000-C000-000000000046}
Shell link (ANSI)	IShellLinkA	{000214EE-0000-0000-C000-000000000046}
Shell link (UNICODE)	IShellLinkW	{000214F9-0000-0000-C000-000000000046}
Structured storage	IStorage	{0000000B-0000-0000-C000-000000000046}
Metadata	IPropertySetStorage	PropertyHandler
Predefined Shell Objects		

The Shell defines additional objects under HKEY\_CLASSES\_ROOT which can be extended in the same way as file types. For example, to add a property sheet handler for all files, you can register under the PropertySheetHandlers key.

```
* HKEY_CLASSES_ROOT
  o *
    + shellex
      # PropertySheetHandlers
```

The following table gives the various subkeys of HKEY\_CLASSES\_ROOT under which extension handlers can be registered. Note that many extension handlers cannot be registered under all of the listed subkeys. For further details, see the specific handler's documentation.

Subkey	Description	Possible Handlers	Version
* All files	Shortcut Menu, Property Sheet, Verbs (see below)	All	
AllFileSystemObjects	All files and file folders	Shortcut Menu, Property Sheet, Verbs	4.71
Folder	All folders	Shortcut Menu, Property Sheet, Verbs	All
Directory	File folders	Shortcut Menu, Property Sheet, Verbs	All
Directory\Background	File folder background	Shortcut Menu only	4.71
Drive	All drives in MyComputer, such as "C:\"	Shortcut Menu, Property Sheet, Verbs	All
Network	Entire network (under My Network Places)	Shortcut Menu, Property Sheet, Verbs	All
Network\Type\#	All objects of type # (see below)	Shortcut menu, Property Sheet, Verbs	4.71
NetShare	All network shares	Shortcut menu, Property Sheet, Verbs	4.71
NetServer	All network servers	Shortcut menu, Property Sheet, Verbs	4.71

network\_provider\_name All objects provided by network provider "network\_provider\_name " Shortcut menu, Property Sheet, Verbs All  
 Printers All printers Shortcut Menu, Property Sheet All  
 AudioCD Audio CD in CD drive Verbs only All  
 DVDFile DVD drive (Windows 98) Shortcut Menu, Property Sheet, Verbs 4.71  
 DVD DVD drive (Windows 2000) Shortcut Menu, Property Sheet, Verbs 4.71

#### Notes:

\* The file folder background shortcut menu is accessed by right-clicking within a file folder, but not over any of the folder's contents.

\* "Verbs" are special commands registered under HKEY\_CLASSES\_ROOT\Subkey\Shell\Verb .

\* For Network\Type\#, "#" is a network provider type code in decimal. The network provider type code is the high word of a network type. The list of network types is given in the Winnetwk.h header file (WNINC\_NET\_\* values). For example, WNINC\_NET\_SHIVA is 0x00330000, so the corresponding type key would be HKEY\_CLASSES\_ROOT\Network\Type\51 .

\* "network\_provider\_name " is a network provider name as specified by WNetGetProviderName, with the spaces converted into underscores. For example, if the Microsoft Networking network provider is installed, its provider name is "Microsoft Windows Network", and the corresponding network\_provider\_name is Microsoft\_Windows\_Network .

#### Example of an Extension Handler Registration

To enable a particular handler, create a subkey under the extension handler type key with the name of the handler. The Shell does not use the handler's name, but it must be different from all other names under that type subkey. Set the default value of the name subkey to the string form of the handler's GUID.

The following example illustrates registry entries that enable shortcut menu and property sheet extension handlers, using an example .myp file class:

```
* HKEY_CLASSES_ROOT
  o .myp
    (Default) = MyProgram.1
  o CLSID
    + {00000000-1111-2222-3333-444444444444}
      # InProcServer32
        (Default) = C:\MyDir\MyCommand.dll
        ThreadingModel = Apartment
    + {11111111-2222-3333-4444-555555555555}
      # InProcServer32
        (Default) = C:\MyDir\MyPropSheet.dll
        ThreadingModel = Apartment
  o MyProgram.1
    (Default) = MyProgram Application
    + Shellex
      # ContextMenuHandler
        * MyCommand
        (Default) = {00000000-1111-2222-3333-444444444444}
      # PropertySheetHandlers
        * MyPropSheet
        (Default) = {11111111-2222-3333-4444-555555555555}
```

The registration procedure discussed in this section must be followed for all Windows systems. However, with Windows NT and Windows 2000 systems, an additional step might be necessary:  
 Registering Shell Extension Handlers on Windows NT and Windows 2000 Systems

Because Windows NT and Windows 2000 systems are designed to be used in a managed environment, installing a Shell extension handler might be somewhat different than for Windows 95 and Windows 98 systems. In particular, access to the registry could be administratively restricted, requiring a somewhat different approach to installation than the process

described in the previous section.

Note The following discussion applies to both Windows NT and Windows 2000 systems. However, with Windows 2000 and later systems, setup programs should generally not write directly to the registry. Instead, setup should be accomplished with Windows Installer packages. These tools ensure that software runs well under Windows 2000, and provides access to new capabilities, such as per-user class registration.

Shell extension handlers run in the Shell process. Because it is a system process, the administrator of a Windows NT system can limit Shell extension handlers to those on an approved list by setting the EnforceShellExtensionSecurity value of the Explorer key to 1 (one), as shown here.

```
* HKEY_CURRENT_USER
  o Software
    + Microsoft
      # Windows
        * CurrentVersion
          o Policies
            + Explorerer
```

EnforceShellExtensionSecurity = 1

To place a Shell extension handler on the approved list, create a REG\_SZ value named as the string form of the handler's GUID under the Approved key.

```
* HKEY_LOCAL_MACHINE
  o Software
    + Microsoft
      # Windows
        * CurrentVersion
          o Shell Extensions
            + Approved
```

The following example adds the MyCommand and MyPropSheet handlers to the approved list:

```
* HKEY_LOCAL_MACHINE
  o Software
    + Microsoft
      # Windows
        * CurrentVersion
          o Shell Extensions
            + Approved
```

{00000000-1111-2222-3333-444444444444} = MyCommand  
{11111111-2222-3333-4444-555555555555} = MyPropSheet

The Shell does not use the value that is assigned to the GUID, but it should be set to make inspecting the registry easier.

Your setup application can add values to the Approved key only if the person installing the application has sufficient privileges. If the attempt to add an extension handler fails, you should inform the user that administrative privileges are required to fully install the application. If the handler is essential to the application, you should fail the setup and notify the user to contact an administrator.

While there is no need to add values to the Approved key on Windows 95 or Windows 98 systems, there is no harm in doing so. The system will simply ignore them. However, there is no guarantee that the key will exist on these systems. Your setup program must be able to handle this case.