Shell Clipboard Formats

05/31/2018 • 17 minutes to read •

In this article

Shell clipboard formats are used to identify the type of Shell data being transferred through the clipboard. Most Shell clipboard formats identify a type of data, such as a list of file names or pointers to item identifier lists (PIDLs). However, some formats are used for communication between source and target. They can expedite the data transfer process by supporting Shell operations such as optimized move and delete on paste. Shell data is always contained in a data object that uses a **FORMATETC** structure as a more general way to characterize data. The structure's **cfFormat** member is set to the clipboard format for the particular item of data. The other members provide additional information, such as the data transfer mechanism. The data is contained in an accompanying **STGMEDIUM** structure.

> **Note**
>
> Standard clipboard format identifiers have the form CF_*XXX*. A common example is CF_TEXT, which is used for transferring ANSI text data. These identifiers have predefined values and can be used directly with **FORMATETC** structures. With the exception of CF_HDROP, Shell format identifiers are not predefined. With the exception of DragWindow, they have the form CFSTR_*XXX*. To differentiate these values from predefined formats, they are often referred to as simply *formats*. However, unlike predefined formats, they must be registered by both source and target before they can be used to transfer data. To register a Shell format, include the Shlobj.h header file and pass the CFSTR_*XXX* format identifier to RegisterClipboardFormat. This function returns a valid clipboard format value, which can then be used as the **cfFormat** member of a **FORMATETC** structure.

The Shell clipboard formats are organized here into three groups, based on how they are used.

- Formats for Transferring File System Objects

Formats for Transferring File System Objects

These formats are used to transfer one or more files or other Shell objects.

- CF_HDROP
- CFSTR_FILECONTENTS
- CFSTR_FILEDESCRIPTOR
- CFSTR_FILENAME
- CFSTR_FILENAMEMAP
- CFSTR_MOUNTEDVOLUME
- CFSTR_SHELLIDLIST
- CFSTR_SHELLIDLISTOFFSET

CF_HDROP

This clipboard format is used when transferring the locations of a group of existing files. Unlike the other Shell formats, it is predefined, so there is no need to call RegisterClipboardFormat. The data consists of an **STGMEDIUM** structure that contains

a global memory object. The structure's **hGlobal** member points to a **DROPFILES** structure as its **hGlobal** member.

The **pFiles** member of the **DROPFILES** structure contains an offset to a double **null**-terminated character array that contains the file names. If you are extracting a CF_HDROP format from a data object, you can use **DragQueryFile** to extract individual file names from the global memory object. If you are creating a CF_HDROP format to place in a data object, you will need to construct the file name array.

The file name array consists of a series of strings, each containing one file's fully qualified path, including the terminating **NULL** character. An additional **null** character is appended to the final string to terminate the array. For example, if the files c:\temp1.txt and c:\temp2.txt are being transferred, the character array looks like this:

CMD Copy

```
c:\temp1.txt'\0'c:\temp2.txt'\0''\0'
```

> **Note**
>
> In this example, '\0' is used to represent the **null** character, not the literal characters that should be included.

If the object was copied to the clipboard as part of a drag-and-drop operation, the **pt** member of the **DROPFILES** structure contains the coordinates of the point where the object was dropped. You can use **DragQueryPoint** to extract the cursor coordinates.

If this format is present in a data object, an OLE drag loop simulates **WM_DROPFILES** functionality with non-OLE drop targets. This is important if your application is the source of a drag-and-drop operation on a Windows 3.1 system.

CFSTR_FILECONTENTS

This format identifier is used with the CFSTR_FILEDESCRIPTOR format to transfer data as if it were a file, regardless of how it is actually stored. The data consists of an **STGMEDIUM** structure that represents the contents of one file. The file is normally represented as a stream object, which avoids having to place the contents of the file in memory. In that case, the **tymed** member of the **STGMEDIUM** structure is set to TYMED_ISTREAM, and the file is represented by an **IStream** interface. The file can also be a storage or global memory object (TYMED_ISTORAGE or TYMED_HGLOBAL). The associated CFSTR_FILEDESCRIPTOR format contains a **FILEDESCRIPTOR** structure for each file that specifies the file's name and attributes.

The target treats the data associated with a CFSTR_FILECONTENTS format as if it were a file. When the target calls **IDataObject::GetData** to extract the data, it specifies a particular file by setting the **lindex** member of the **FORMATETC** structure to the zero-based index of the file's **FILEDESCRIPTOR** structure in the accompanying CFSTR_FILEDESCRIPTOR format. The target then uses the returned interface pointer or global memory handle to extract the data.

CFSTR_FILEDESCRIPTOR

This format identifier is used with the CFSTR_FILECONTENTS format to transfer data as a group of files. These two formats are the preferred way to transfer Shell objects that are not stored as file-system files. For example, these formats can be used to transfer a group of email messages as individual files, even though each email is actually stored as a block of data in a database. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **FILEGROUPDESCRIPTOR** structure that is followed by an array containing one **FILEDESCRIPTOR** structure for each file in the group. For each **FILEDESCRIPTOR** structure, there is a separate CFSTR_FILECONTENTS format that contains the contents of the file. To identify a particular file's CFSTR_FILECONTENTS format, set the **lIndex** value of the **FORMATETC** structure to the zero-based index of the file's **FILEDESCRIPTOR** structure.

The CFSTR_FILEDESCRIPTOR format is commonly used to transfer data as if it were a group of files, regardless of how it is actually stored. From the target's perspective, each CFSTR_FILECONTENTS format represents a single file and is treated accordingly. However, the source can store the data in any way it chooses. While a CSFTR_FILECONTENTS format might correspond to a single file, it could also, for example, represent data extracted by the source from a database or text document.

CFSTR_FILENAME

This format identifier is used to transfer a single file. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a single **null**-terminated string containing the file's fully qualified file path. This format has been superseded by CF_HDROP, but it is supported for backward compatibility with Windows 3.1 applications.

CFSTR_FILENAMEMAP

This format identifier is used when a group of files in CF_HDROP format is being renamed as well as transferred. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a double

null-terminated character array. This array contains a new name for each file, in the same order that the files are listed in the accompanying CF_HDROP format. The format of the character array is the same as that used by CF_HDROP to list the transferred files.

CFSTR_MOUNTEDVOLUME

This format identifier is used to transfer a path on a mounted volume. It is similar to CF_HDROP, but it contains only a single path and can handle the longer path strings that might be needed to represent a path when the volume is mounted on a folder. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a single **null**-terminated string containing the fully qualified file path. The path string must end with a '\' character, followed by the terminating **NULL**.

Prior to Windows 2000, volumes could be mounted only on drive letters. For Windows 2000 and later systems with an NTFS formatted drive, you can also mount volumes on empty folders. This feature allows a volume to be mounted without taking up a drive letter. The mounted volume can use any currently supported format, including FAT, FAT32, NTFS, and CDFS.

You can add pages to a Drive Properties property sheet by implementing a property sheet handler. If the volume is mounted on a drive letter, the Shell passes path information to the handler with the CF_HDROP format. With Windows 2000 and later systems, the CF_HDROP format is used when a volume is mounted on a drive letter, just as with earlier systems. However, if a volume is mounted on a folder, the CFSTR_MOUNTEDVOLUME format identifier is used instead of CF_HDROP.

If only drive letters will be used to mount volumes, only CF_HDROP will be used, and existing property sheet handlers will work as they did with earlier systems. However, if you want your handler to display a page for volumes that are mounted on folders as well as drive letters, the handler must be able to understand both the CSFTR_MOUNTEDVOLUME and CF_HDROP formats.

CFSTR_SHELLIDLIST

This format identifier is used when transferring the locations of one or more existing namespace objects. It is used in much the same way as CF_HDROP, but it contains PIDLs instead of file system paths. Using PIDLs allows the CFSTR_SHELLIDLIST format to handle virtual objects as well as file system objects. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **CIDA** structure.

The **aoffset** member of the **CIDA** structure is an array containing offsets to the beginning of the **ITEMIDLIST** structure for each PIDL that is being transferred. To extract a particular PIDL, first determine its index. Then, add the **aoffset** value that corresponds to that index to the address of the **CIDA** structure.

The first element of **aoffset** contains an offset to the fully qualified PIDL of a parent folder. If this PIDL is empty, the parent folder is the desktop. Each of the remaining elements of the array contains an offset to one of the PIDLs to be transferred. All of these PIDLs are relative to the PIDL of the parent folder.

The following two macros can be used to retrieve PIDLs from a **CIDA** structure. The first takes a pointer to the structure and retrieves the PIDL of the parent folder. The second takes a pointer to the structure and retrieves one of the other PIDLs, identified by its zero-based index.

C++ Copy

```
#define GetPIDLFolder(pida) (LPCITEMIDLIST)(((LPBYTE)pida)+(pida)->aoffset[0])

#define GetPIDLItem(pida, i) (LPCITEMIDLIST)(((LPBYTE)pida)+(pida)->aoffset[i+1])
```

> **Note**
>
> The value that is returned by these macros is a pointer to the PIDL's **ITEMIDLIST** structure. Since these structures vary in length, you must determine the end of the structure by walking through each of the **ITEMIDLIST** structure's **SHITEMID** structures until you reach the two-byte **NULL** that marks the end.

CFSTR_SHELLIDLISTOFFSET

This format identifier is used with formats such as CF_HDROP, CFSTR_SHELLIDLIST, and CFSTR_FILECONTENTS to specify the position of a group of objects following a transfer. The data consists of an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to an array of **POINT** structures. The first structure specifies the screen coordinates, in pixels, of the upper-left corner of the rectangle that encloses the group. The remainder of the structures specify the locations of the individual objects relative to the group's position. They must be in the same order as that used to list the objects in the associated format.

Formats for Transferring Virtual Objects

The CFSTR_SHELLIDLIST format can be used to transfer both file system and virtual objects. However, there are also several specialized formats for transferring particular types of virtual objects.

- CFSTR_NETRESOURCES
- CFSTR_PRINTERGROUP
- CFSTR_INETURL
- CFSTR_SHELLURL (deprecated)

CFSTR_NETRESOURCES

This format identifier is used when transferring network resources, such as a domain or server. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **NRESARRAY** structure. The **nr** member of that structure indicates a **NETRESOURCE** structure whose **lpRemoteName** member contains a **null**-terminated string identifying the network resource. The drop target can then use the data with any of the Windows Networking (WNet) API functions, such as **WNetAddConnection**, to perform network operations on the object.

CFSTR_PRINTERGROUP

This format identifier is used when transferring the friendly names of printers. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a string in the same format as that used with CF_HDROP. However, the **pFiles** member of the **DROPFILES** structure contains one or more friendly names of printers instead of file paths.

CFSTR_INETURL

This format identifier replaces CFSTR_SHELLURL (deprecated). If you want your application to manipulate clipboard URLs, use CFSTR_INETURL instead of CFSTR_SHELLURL (deprecated). This format gives the best clipboard representation of a single URL. If UNICODE is not defined, the application retrieves the CF_TEXT/CFSTR_SHELLURL version of the URL. If UNICODE is defined, the application retrieves the CF_UNICODE version of the URL.

CFSTR_SHELLURL (deprecated)

> **Note**
>
> This format identifier has been deprecated; use CFSTR_INETURL instead.

Formats for Communication Between Source and Target

These format identifiers allow communication between source and target. The formats accompany the data and give applications a greater degree of control over move-copy-paste or drag-and-drop operations involving Shell objects.

- CFSTR_INDRAGLOOP
- CFSTR_LOGICALPERFORMEDDROPEFFECT
- CFSTR_PASTESUCCEEDED
- CFSTR_PERFORMEDDROPEFFECT
- CFSTR_PREFERREDDROPEFFECT
- CFSTR_TARGETCLSID
- CFSTR_UNTRUSTEDDRAGDROP
- DragWindow

CFSTR_INDRAGLOOP

This format identifier is used by a data object to indicate whether it is in a drag-and-drop loop. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **DWORD** value. If the **DWORD** value is nonzero, the data object is within a drag-and-drop loop. If the value is set to zero, the data object is not within a drag-and-drop loop.

Some drop targets might call **IDataObject::GetData** and attempt to extract data while the object is still within the drag-and-drop loop. Fully rendering the object for each such occurrence might cause the drag cursor to stall. If the data object supports CFSTR_INDRAGLOOP, the target can instead use that format to check the status of the drag-and-drop loop and avoid memory intensive rendering of the object until it is actually dropped. The formats that are memory intensive to render should still be included in the **FORMATETC** enumerator and in calls to **IDataObject::QueryGetData**. If the data object does not set CFSTR_INDRAGLOOP, it should act as if the value is set to zero.

CFSTR_LOGICALPERFORMEDDROPEFFECT

Version 5.0.This format identifier allows a drop source to call the data object's **IDataObject::GetData** method to determine the outcome of a Shell data transfer. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a DWORD containing a **DROPEFFECT** value.

The CFSTR_PERFORMEDDROPEFFECT format identifier was intended to allow the target to indicate to the data object what operation actually took place. However, the Shell uses optimized moves for file system objects whenever possible. In that case, the Shell normally sets the CFSTR_PERFORMEDDROPEFFECT value to DROPEFFECT_NONE, to indicate to the data object that the original data has been deleted. Thus, the source cannot use the CFSTR_PERFORMEDDROPEFFECT value to determine which operation has taken place. While most sources do not need this information, there are some exceptions. For instance, even though optimized moves eliminate the need for a source to delete any data, the source might still need to update a related database to indicate that the files have been moved or copied.

If a source needs to know which operation took place, it can call the data object's **IDataObject::GetData** method and request the CFSTR_LOGICALPERFORMEDDROPEFFECT format. This format essentially reflects what happens from the user's point of view after the operation is complete. If a new file is created and the original file is deleted, the user sees a move operation and the format's data value is set to DROPEFFECT_MOVE. If the original file is still there, the user sees a copy operation and the format's data value is set to DROPEFFECT_COPY. If a link was created, the format's data value will be DROPEFFECT_LINK.

CFSTR_PASTESUCCEEDED

This format identifier is used by the target to inform the data object, through its **IDataObject::SetData** method, that a delete-on-paste operation succeeded. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **DWORD** containing a **DROPEFFECT** value. This format is used to notify the data object that it should complete the cut operation and delete the original data, if necessary. For more information, see Delete-on-Paste Operations.

CFSTR_PERFORMEDDROPEFFECT

This format identifier is used by the target to inform the data object through its **IDataObject::SetData** method of the outcome of a data transfer. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **DWORD** set to the appropriate **DROPEFFECT** value, normally DROPEFFECT_MOVE or DROPEFFECT_COPY.

This format is normally used when the outcome of an operation can be either move or copy, such as in an optimized move or delete-on-paste operation. It provides a reliable way for the target to tell the data object what actually happened. It was introduced because the value of *pdwEffect* returned by **DoDragDrop** did not reliably indicate which

operation had taken place. The CFSTR_PERFORMEDDROPEFFECT format is the reliable way to indicate that an unoptimized move has taken place.

CFSTR_PREFERREDDROPEFFECT

This format identifier is used by the source to specify whether its preferred method of data transfer is move or copy. A drop target requests this format by calling the data object's **IDataObject::GetData** method. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **DWORD** value. This value is set to DROPEFFECT_MOVE if a move operation is preferred or DROPEFFECT_COPY if a copy operation is preferred.

This feature is used when a source can support either a move or copy operation. It uses the CFSTR_PREFERREDDROPEFFECT format to communicate its preference to the target. Because the target is not obligated to honor the request, the target must call the source's **IDataObject::SetData** method with a CFSTR_PERFORMEDDROPEFFECT format to tell the data object which operation was actually performed.

With a delete-on-paste operation, the CFSTR_PREFERREDDROPFORMAT format is used to tell the target whether the source did a cut or copy. With a drag-and-drop operation, you can use CFSTR_PREFERREDDROPFORMAT to specify the Shell's action. If this format is not present, the Shell performs a default action, based on context. For instance, if a user drags a file from one volume and drops it on another volume, the Shell's default action is to copy the file. By including a CFSTR_PREFERREDDROPFORMAT format in the data object, you can override the default action and explicitly tell the Shell to copy, move, or link the file. If the user chooses to drag with the right button, CFSTR_PREFERREDDROPFORMAT specifies the default command on the drag-and-drop shortcut menu. The user is still free to choose other commands on the menu.

Before Microsoft Internet Explorer 4.0, an application indicated that it was transferring shortcut file types by setting FD_LINKUI in the **dwFlags** member of the **FILEDESCRIPTOR** structure. Targets then had to use a potentially time-consuming call to **IDataObject::GetData** to find out if the FD_LINKUI flag was set. Now, the preferred way to indicate that shortcuts are being transferred is to use the CFSTR_PREFERREDDROPEFFECT format set to DROPEFFECT_LINK. However, for backward compatibility with older systems, sources should still set the FD_LINKUI flag.

CFSTR_TARGETCLSID

This format identifier is used by a target to provide its CLSID to the source. The data is an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to the CLSID GUID of the drop target.

This format is used primarily to allow objects to be deleted by dragging them to the Recycle Bin. When an object is dropped in the Recycle Bin, the source's **IDataObject::SetData** method is called with a CFSTR_TARGETCLSID format set to the Recycle Bin's CLSID (CLSID_RecycleBin). The source can then delete the original object.

CFSTR_UNTRUSTEDDRAGDROP

This format identifier is used by Windows Internet Explorer and the Windows Shell to provide a mechanism through which to block or prompt for drag-and-drop operations originating from Internet Explorer in conjunction with the **URLACTION_SHELL_ENHANCED_DRAGDROP_SECURITY** flag.

CFSTR_UNTRUSTEDDRAGDROP is added by the source of a drag-and-drop operation to specify that the data object might contain untrustworthy data. The data is represented by an **STGMEDIUM** structure that contains a global memory object. The structure's **hGlobal** member points to a **DWORD** set to an appropriate **URL Action** flag to cause a policy check through the **IInternetSecurityManager::ProcessUrlAction** method, using the **PUAF_ENFORCERESTRICTED** flag.

DragWindow

This format is used in a drag-and-drop operation to identify an object's drag image (window) so that its visual information can be updated dynamically. When an object is dragged over a drop target, an application updates its **DROPDESCRIPTION** structure in response to the **IDropTarget::DragOver** or **IDropSource::GiveFeedback** method. The **DROPDESCRIPTION** is updated with a new **DROPIMAGETYPE** value that indicates the decoration to be applied to the drag window's visual; for instance, an indication that the file is being copied rather than moved or that the object cannot be dropped to that location. However, until the object receives a **DDWM_UPDATEWINDOW** message, the visuals are not updated. This format provides the **HWND** of the recipient drag window to the sender of the **DDWM_UPDATEWINDOW** message.

The clipboard data is of type **TYMED_HGLOBAL**. It is a **DWORD** representation of an **HWND**. The data can be passed to the **ULongToHandle** function, defined in Basetsd.h, to provide a 64-bit **HWND** for use on 64-bit Windows.

This format does not require the inclusion of Shlobj.h.

Is this page helpful?

Yes No