shlobj_102894

//===============================================================================================

// Copyright (c) Microsoft Corporation 1991-1994
// File: shlobj.h
//===============================================================================================

#ifndef _SHOBJJ_H_
#define _SHOBJJ_H_

#include <ole2.h>
#include <prshlt.h>
#include <commctrl1.h> // for LPTBBUTTON

#ifndef INITGUID
#include <shguid.h>
#endif /* !INITGUID */

#ifndef RC_INVOKED
#pragma pack(1) /* Assume byte packing throughout */
#endif /* !RC_INVOKED */

#ifndef __cplusplus
extern "C" { /* Assume C declarations for C++ */
#endif /* __cplusplus */

typedef void const * LPCVOID;

//===============================================================================================

// Object identifiers in the explorer's name space (ItemId and IDList)

// All the items that the user can browse with the explorer (such as files, directories, servers, work-groups, etc.) has an identifier which is unique among items within the parent folder. Those identifiers are called item IDs (SHITEMID). Since all its parent folders have their own item IDs, any items can be uniquely identified by a list of item IDs, which is called an ID list (ITEMIDLIST).

// ID lists are almost always allocated by the task allocator (see some description below as well as OLE 2.0 SDK) and may be passed across some of shell interfaces (such as IShellFolder). Each item ID in an ID list is only meaningful to its parent folder (which has generated it), and all the clients must treat it as an opaque binary data except the first two bytes, which indicates the size of the item ID.

// When a shell extension -- which implements the IShellFolder interace -- generates an item ID, it may put any information in it, not only the data with that it needs to identifies the item, but also some additional information, which would help implementing some other functions efficiently. For example, the shell's IShellFolder implementation of file system items stores the primary (long) name of a file or a directory as the item identifier, but it also stores its alternative (short) name, size and date etc.

// When an ID list is passed to one of shell APIs (Such as SHGetPathFromIDList), it is always an absolute path -- relative from the root of the name space, which is the desktop folder. When an ID list is passed to one of IShellFolder member function, it is always a relative path from the folder (unless it is explicitly specified).  

Page 1
shlobj_102894

//
//******************************************************************************
//
// SHITEMID -- Item ID
//
// typedef struct _SHITEMID     // mkid
// {
//    USHORT cb;       // Size of the ID (including cb itself)
//    BYTE obID[1];   // The item ID (variable length)
// } SHITEMID, *LPSHITEMID;
// typedef const SHITEMID * LPCSHITEMID;
//
// ITEMIDLIST -- List of item IDs (combined with 0-terminator)
//
// typedef struct _ITEMIDLIST   // idl
// {
//    SHITEMID mkid;
// } ITEMIDLIST, *LPITEMIDLIST;
// typedef const ITEMIDLIST * LPCITEMIDLIST;
//
//******************************************************************************
//
// Task allocator API
//
// All the shell extensions MUST use the task allocator (see OLE 2.0
// programming guide for its definition) when they allocate or free
// memory objects (mostly ITEMIDLIST) that are returned across any
// shell interfaces. There are two ways to access the task allocator
// from a shell extension depending on whether or not it is linked with
// OLE32.DLL or not (purely for efficiency).
//
// (1) A shell extension which calls any OLE API (i.e., linked with
// OLE32.DLL) should call OLE's task allocator (by retrieving
// the task allocator by calling CoGetMalloc API).
//
// (2) A shell extension which does not call any OLE API (i.e., not linked
// with OLE32.DLL) should call the shell task allocator API (defined
// below), so that the shell can quickly loads it when OLE32.DLL is not
// loaded by any application at that point.
//
// Notes:
// In next version of Windows release, SHGetMalloc will be replaced by
// the following macro.
//
// #define SHGetMalloc(ppmem) CoGetMalloc(MEMCTX_TASK, ppmem)
//
//******************************************************************************

HRESULT WINAPI SHGetMalloc(LPMALLOC * ppMalloc);

//
// IContextMenu interface
//
// [Overview]
//
// The shell uses the IContextMenu interface in following three cases.
shlobj_102894

case-1: The shell is loading context menu extensions.

When the user clicks the right mouse button on an item within the shell's name space (i.e., file, directory, server, work-group, etc.), it creates the default context menu for its type, then loads context menu extensions that are registered for that type (and its base type) so that they can add extra menu items. These context menu extensions are registered at HKCR\{ProgID\}\shell\ContextMenuHandlers.

case-2: The shell is retrieving a context menu of sub-folders in extended name-space.

When the explorer's name space is extended by name space extensions, the shell calls their IShellFolder::GetUIObjectOf to get the IContextMenu objects when it creates context menus for folders under those extended name spaces.

case-3: The shell is loading non-default drag and drop handler for directories.

When the user performed a non-default drag and drop onto one of file system folders (i.e., directories), it loads shell extensions that are registered at HKCR\{ProgID\}\DragDropHandlers.

[Member functions]

IClassFactory::QueryContextMenu

This member function may insert one or more menu items to the specified menu (hmenu) at the specified location (indexMenu which is never be -1). The IDs of those menuitem must be in the specified range (idCmdFirst and idCmdLast). It returns the maximum menuitem ID offset (ushort) in the 'code' field (low word) of the scoce.

The uFlags specify the context. It may have one or more of following flags.

CMF_DEFAULTONLY: This flag is passed if the user is invoking the default action (typically by double-clicking, case 1 and 2 only). Context menu extensions (case 1) should not add any menu items, and returns NOERROR.

CMF_VIEWSONLY: The explorer passes this flag if it is constructing a context menu for a short-cut object (case 1 and case 2 only). If this flag is passed, it should not add any menu-items that is not appropriate from a short-cut.

A good example is the "Delete" menuitem, which confuses the user because it is not clear whether it deletes the link source item or the link itself.

CMF_EXPLORER: The explorer passes this flag if it has the left-side pane (case 1 and 2 only). Context menu extensions should ignore this flag.

High word (16-bit) are reserved for context specific communications and the rest of flags (13-bit) are reserved by the system.

IClassFactory::InvokeCommand

This member is called when the user has selected one of menuitems that are inserted by previous QueryContextMenu member. In this case, the LOWORD(lpici->lpVerb) contains the menuitem ID offset (menuitem ID - idCmdFirst).
shlobj_102894

// This member function may also be called programmatically. In such a case,
// lpici->lpverb specifies the canonical name of the command to be invoked,
// which is typically retrieved by GetCommandString member previously.

Parameters in lpici:
  cbSize -- Specifies the size of this structure (sizeof(*lpici))
  hwnd -- Specifies the owner window for any message/dialog box.
  fMask -- Specifies whether or not dwHotkey/hIcon parameter is valid.
  lpverb -- Specifies the command to be invoked.
  lpParameters -- Parameters (optional)
  lpDirectory -- working directory (optional)
  nShow -- Specifies the flag to be passed to ShowWindow (SW_*)
  dwHotkey -- Hot key to be assigned to the app after invoked (optional).
  hIcon -- Specifies the icon (optional). -- BUGBUG: describe it.

IContextMenu::GetCommandString

This member function is called by the explorer either to get the
canonical (language independent) command name (uFlags == GCS_VERB) or
the help text ((uFlags & GCS_HELPTEXT) != 0) for the specified command.
The retrieved canonical string may be passed to its InvokeCommand
member function to invoke a command programmatically. The explorer
displays the help texts in its status bar; therefore, the length of
the help text should be reasonably short (<40 characters).

Parameters:
  idCmd -- Specifies menuitem ID offset (from idCmdFirst)
  uFlags -- Either GCS_VERB or GCS_HELPTEXT
  pwReserved -- Reserved (must pass NULL when calling, must ignore when called)
  pszName -- Specifies the string buffer.
  cchMax -- Specifies the size of the string buffer.

//#undef INTERFACE
#define INTERFACE IContextMenu

// QueryContextMenu uFlags
#define CMF_NORMAL 0x00000000
#define CMF_DEFAULTONLY 0x00000001
#define CMF_VERBSONLY 0x00000002
#define CMF_EXPLORE 0x00000004
#define CMF_RESERVED 0xffffffff // View specific

// GetCommandString uFlags
#define GCS_VERB 0x00000000 // canonical verb
#define GCS_HELPTEXT 0x00000001 // help text (for status bar)
#define GCS_VALIDATE 0x00000002 // validate command exists

#define CMDSTR_NEWFOLDER "NewFolder"
#define CMDSTR_VIEWLIST "ViewList"
#define CMDSTR_VIEWDETAILS "ViewDetails"

#define CMIC_MASK_HOTKEY SEE_MASK_HOTKEY
#define CMIC_MASK_ICON SEE_MASK_ICON
#define CMIC_MASK_FLAG_NO_UI SEE_MASK_FLAG_NO_UI

#define CMIC_VALIDSEE_FLAGS SEE_VALID_CMIC_FLAGS /*
Internal */

typedef struct _CMInvokeCommandInfo {
shlobj_102894
DWORD cbSize;  // must be sizeof(CMINVOKECOMMANDINFO)
DWORD fMask;  // any combination of CMIC_MASK_*
HWND hwnd;  // might be NULL (indicating no owner window)
LPCSTR lpVerb;  // either a string of MAKEINTRESOURCE(idOffset)
LPCSTR lpParameters;  // might be NULL (indicating no parameter)
LPCSTR lpDirectory;  // might be NULL (indicating no specific directory)
int nShow;  // one of Sw_values for ShowWindow() API
DWORD dwHotKey;
HANDLE hIcon;
} CMINVOKECOMMANDINFO, *LPCMINVOKECOMMANDINFO;

#ifndef INTERFACE
#define INTERFACE IContextMenu

DECLARE_INTERFACE_(ICustomContextMenu, IUnknown)
{
    // ** IUnknown methods **
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    STDMETHOD(QueryContextMenu)(THIS_
        HMENU hmenu,
        UINT indexMenu,
        UINT idCmdFirst,
        UINT idCmdLast,
        UINT uFlags) PURE;

    STDMETHOD(InvokeCommand)(THIS_
        LPCMINVOKECOMMANDINFO lpici) PURE;

    STDMETHOD(GetCommandString)(THIS_
        UINT idCmd,
        UINT uType,
        UINT * pwReserved,
        LPSTR pszName,
        UINT cchMax) PURE;
};

typedef IContextMenu * LPCCONTEXMENU;

// Interface: IShellExtInit

The IShellExtInit interface is used by the explorer to initialize shell extension objects. The explorer (1) calls CoCreateInstance (or equivalent) with the registered CLSID and IID_IShellExtInit, (2) calls its Initialize member, then (3) calls its QueryInterface to a particular interface (such as IContextMenu or IPropSheetExt and (4) performs the rest of operation.

[Member functions]

IShellExtInit::Initialize

This member function is called when the explorer is initializing either context menu extension, property sheet extension or non-default drag-drop
shlobj_102894

// extension.

// Parameters: (context menu or property sheet extension)
// pidlFolder -- Specifies the parent folder.
// lpdobj -- Specifies the set of items selected in that folder.
// hkeyProgID -- Specifies the type of the focused item in the selection.

// Parameters: (non-default drag-and-drop extension)
// pidlFolder -- Specifies the target (destination) folder.
// lpdobj -- Specifies the items that are dropped (see the description
// about shell's clipboard below for clipboard formats).
// hkeyProgID -- Specifies the folder type.

//==============================================================================

#undef INTERFACE
#define INTERFACE IShellExtInit

DECLARE_INTERFACE_(IShellExtInit, IUnknown)
{
    // *** IUnknown methods ***
    STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef) (THIS) PURE;
    STDMETHOD_(ULONG, Release) (THIS) PURE;

    // *** IShellExtInit methods ***
    STDMETHOD(Initialize)(THIS_, LPCITEMIDLIST pidlFolder,
                           LPDATAOBJECT lpdobj, HKEY hkeyProgID) PURE;
};

typedef IShellExtInit * LPSHELLEXTINIT;

//==============================================================================

// Interface: IShellPropSheetExt

// The explorer uses the IShellPropSheetExt to allow property sheet
// extensions or control panel extensions to add additional property
// sheet pages.

// [Member functions]

// IShellPropSheetExt::AddPages

// The explorer calls this member function when it finds a registered
// property sheet extension for a particular type of object. For each
// additional page, the extension creates a page object by calling
// CreatePropertySheetPage API and calls lpfnAddPage.

// Parameters:
// lpfnAddPage -- Specifies the callback function.
// lParam -- Specifies the opaque handle to be passed to the callback function.

// IShellPropSheetExt::ReplacePage

// The explorer never calls this member function of property sheet extensions. The
// explorer calls this member of control panel extensions, so that they
// can replace some of default control panel pages (such as a page of
// mouse control panel).
shlobj_102894

// Parameters:
// wParam -- Specifies the page to be replaced.
// lpfnID -- Specifies the callback function.
// lParam -- Specifies the opaque handle to be passed to the callback function.

#ifndef INTERFACE
#define INTERFACE IShellPropSheetExt

DECLARE_INTERFACE_(IShellPropSheetExt, IUnknown)
{
    // **IUnknown methods **
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef) (THIS) PURE;
    STDMETHOD_(ULONG, Release) (THIS) PURE;

    // **IShellPropSheetExt methods **
    STDMETHOD(AddPages)(THIS, LPFNADDPROPSHEETPAGE lpfnAddPage, LPARAM lParam) PURE;
    STDMETHOD(ReplacePage)(THIS, UINT wParam, LPFNADDPROPSHEETPAGE lpfnReplaceWith,
                           LPARAM lParam) PURE;
}

typedef IShellPropSheetExt * LPSHELLPROPSHEETEXT;

IColor ExtractIcon interface

This interface is used in two different places in the shell.

Case-1: Icons of sub-folders for the scope-pane of the explorer.

It is used by the explorer to get the "icon location" of
sub-folders from each shell folders. When the user expands a folder
in the scope pane of the explorer, the explorer does following:
(1) binds to the folder (gets IShellFolder),
(2) enumerates its sub-folders by calling its EnumObjects member,
(3) calls its GetUIObjectOf member to get IExtractIcon interface
    for each sub-folders.
    In this case, the explorer uses only IExtractIcon::GetIconLocation
    member to get the location of the appropriate icon. An icon location
    always consists of a file name (typically DLL or EXE) and either an icon
    resource or an icon index.

Case-2: Extracting an icon image from a file

It is used by the shell when it extracts an icon image
from a file. When the shell is extracting an icon from a file,

it does following:
(1) creates the icon extraction handler object (by getting its CLSID
under the {ProgID}\shell\ExtractIconHanler key and calling
CoCreateInstance requesting for IExtractIcon interface).
(2) Calls IExtractIcon::GetIconLocation.
(3) Then, calls IExtractIcon::ExtractIcon with the location/index pair.
(4) If (3) returns NOERROR, it uses the returned icon.
(5) Otherwise, it recursively calls this logic with new location
    assuming that the location string contains a fully qualified path name.

From extension programmer's point of view, there are only two cases
shlobj_102894
// where they provide implementations of IExtractIcon:
// Case-1) providing explorer extensions (i.e., IShellFolder).
// Case-2) providing per-instance icons for some types of files.

Because Case-1 is described above, we'll explain only Case-2 here.

When the shell is about display an icon for a file, it does following:
(1) Finds its ProgID and ClassID.
(2) If the file has a ClassID, it gets the icon location string from the
"DefaultIcon" key under it. The string indicates either per-class
icon (e.g., "FOOBAR.DLL.2") or per-instance icon (e.g., "%1,1").
(3) If a per-instance icon is specified, the shell creates an icon
extraction handler object for it, and extracts the icon from it
(which is described above).

It is important to note that the shell calls IExtractIcon::GetIconLocation
first, then calls IExtractIcon::ExtractIcon. Most application programs
that support per-instance icons will probably store an icon location
(DLL/EXE name and index/id) rather than an icon image in each file.
In those cases, a programmer needs to implement only the GetIconLocation
member and it ExtractIcon member simply returns S_FALSE. They need to
implement ExtractIcon member only if they decided to store the icon images
within files themselfed or some other database (which is very rare).

[Member functions]

IExtractIcon::GetIconLocation
This function returns an icon location.

Parameters:
  uFlags [in] -- Specifies if it is opened or not (GIL_OPENICON or 0)
  szIconFile [out] -- Specifies the string buffer buffer for a location name.
  cchMax [in] -- Specifies the size of szIconFile (almost always MAX_PATH)
  piIndex [out] -- Specifies the address of UINT for the index.
  pwFlags [out] -- Returns GIL_* flags

Returns:
  NOERROR, if it returns a valid location; S_FALSE, if the shell use a
default icon.

Notes: The location may or may not be a path to a file. The caller can
not assume anything unless the subsequent ExtractIcon member call returns
S_FALSE.

IExtractIcon::ExtractIcon
This function extracts an icon image from a specified file.

Parameters:
  pszFile [in] -- Specifies the icon location (typically a path to a file).
  nIconIndex [in] -- Specifies the icon index.
  phiconLarge [out] -- Specifies the HICON variable for large icon.
  phiconSmall [out] -- Specifies the HICON variable for small icon.
  nIconSize [in] -- Specifies the size icon required (size of large icon)

Returns:
  NOERROR, if it extracted the from the file.
  S_FALSE, if the caller should extract from the file specified in the
  location.

================================================================
shlobj_102894

#define INTERFACE  
#define INTERFACE  IExtractIcon

// GetIconLocation() input flags
#define GIL_OPENICON 0x0001  // allows containers to specify an "open" look
// return FALSE to get the standard look

// GetIconLocation() return flags
#define GIL_SIMULATEDDOC 0x0001  // simulate this document icon for this
#define GIL_PERINSTANCE 0x0002  // icons from this class are per instance (each
file has its own)
#define GIL_PERCLASS 0x0004  // icons from this class per class (shared for
all files of this type)
#define GIL_NOTFILENAME 0x0008  // location is not a filename, must call
::ExtractIcon

DECLARE_INTERFACE_(IExtractIcon, IUnknown)  // exic
{
  // *** IUnknown methods ***
  STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID *ppvObj) PURE;
  STDMETHOD_(ULONG, AddRef) (THIS_) PURE;
  STDMETHOD_(ULONG, Release) (THIS_) PURE;

  // *** IExtractIcon methods ***
  STDMETHOD(GetIconLocation)(THIS_
    UINT uFlags,
    LPSTR pszIconFile,
    UINT cchMax,
    int *piIndex,
    UINT *pwFlags) PURE;

  STDMETHOD(ExtractIcon)(THIS_
    LPCSTR pszFile,
    UINT nIconIndex,
    HICON *phiconlarge,
    HICON *phiconsmall,
    UINT nIconSize) PURE;
};

typedef IExtractIcon * LPEXTRACTICON;

//=============================================//
// IShellLink Interface
//=============================================//

// IShellLink::Resolve fFlags
typedef enum {
  SLR_NO_UI = 0x0001,
  SLR_ANY_MATCH = 0x0002,
  SLR_UPDATE = 0x0004,
} SLR_FLAGS;

// IShellLink::GetPath fFlags
typedef enum {
  SLGP_SHORTPATH = 0x0001,
  SLGP_UNCPRIORITY = 0x0002,
} SLGP_FLAGS;
shlobj_102894

#define INTERFACE IShellLink

DECLARE_INTERFACE_(IShellLink, IUnknown) // sl

{ // *** IUnknown methods ***
    STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef)(THIS) PURE;
    STDMETHOD_(ULONG, Release)(THIS) PURE;

    STDMETHOD_(GetPath)(THIS_ LPSTR pszFile, int cchMaxPath, WIN32_FIND_DATA *pfld,
                        DWORD fFlags) PURE;
    STDMETHOD_(GetIDList)(THIS_ LPITEMIDLIST * ppidl) PURE;
    STDMETHOD_(SetIDList)(THIS_ LPCITEMIDLIST pidl) PURE;

    STDMETHOD_(GetDescription)(THIS_ LPSTR pszName, int cchMaxName) PURE;
    STDMETHOD_(SetDescription)(THIS_ LPCSTR pszName) PURE;

    STDMETHOD_(GetWorkingDirectory)(THIS_ LPSTR pszDir, int cchMaxPath) PURE;
    STDMETHOD_(SetWorkingDirectory)(THIS_ LPCSTR pszDir) PURE;

    STDMETHOD_(GetArguments)(THIS_ LPSTR pszArgs, int cchMaxPath) PURE;
    STDMETHOD_(SetArguments)(THIS_ LPCSTR pszArgs) PURE;

    STDMETHOD_(GetHotkey)(THIS_ WORD * pwHotkey) PURE;
    STDMETHOD_(SetHotkey)(THIS_ WORD wHotkey) PURE;

    STDMETHOD_(GetShowCmd)(THIS_ int * piShowCmd) PURE;
    STDMETHOD_(SetShowCmd)(THIS_ int iShowCmd) PURE;

    STDMETHOD_(GetIconLocation)(THIS_ LPSTR pszIconPath, int cchIconPath, int
                                *piIcon) PURE;
    STDMETHOD_(SetIconLocation)(THIS_ LPCSTR pszIconPath, int iIcon) PURE;

    STDMETHOD_(SetRelativePath)(THIS_ LPCSTR pszPathRel, LPCITEMIDLIST pidlRel) PURE;
    STDMETHOD_(Resolve)(THIS_ HWND hwnd, DWORD fFlags) PURE;
    STDMETHOD_(SetPath)(THIS_ LPCSTR pszFile) PURE;
}

//ICopyHook Interface

The copy hook is called whenever file system directories are
copy/moved/deleted/renamed via the shell. It is also called by the shell
on changes of status of printers.

Clients register their id under STRREG_SHEX_COPYHOOK for file system hooks
and STRREG_SHEX_PRNCCOPYHOOK for printer hooks.
the CopyCallback is called prior to the action, so the hook has the chance
to allow, deny or cancel the operation by returning the values:
    IDYES  - means allow the operation
    IDNO   - means disallow the operation on this file, but continue with
              any other operations (eg. batch copy)
    IDCANCEL - means disallow the current operation and cancel any pending
                operations

arguments to the CopyCallback
shlobj_102894

hwnd - window to use for any UI
wFunc - what operation is being done
wFlags - and flags (FOF_*) set in the initial call to the file operation
pszSrcFile - name of the source file
dwSrcFileAttributes - file attributes of the source file
pszDestFile - name of the destination file (for move and renames)
dwDestFileAttributes - file attributes of the destination file

/*
 *===================================================================
 *
 * #undef INTERFACE
 * #define INTERFACE ICopyHook
 *
 * ifndef FO_MOVE // these need to be kept in sync with the ones in shell.h
 *
 * // file operations
 *
 * #define FO_MOVE 0x0001
 * #define FO_COPY 0x0002
 * #define FO_DELETE 0x0003
 * #define FO_RENAME 0x0004
 *
 * #define FOF_MULTIDESTFILES 0x0001
 * #define FOF_CONFIRMMOUSE 0x0002
 * #define FOF_SILENT 0x0004 // don't create progress/report
 * #define FOF_RENAMEONCOLLISION 0x0008
 * #define FOF_NOCONFIRMATION 0x0010 // Don't prompt the user.
 * #define FOF_WANTMAPPINGHANDLE 0x0020 // Fill in SHFILEOPOSTRUCT.hNameMappings
 *                                                        // Must be freed using SHFileOPOSTRUCT.hNameMappings
 * #define FOF_ALLOWUNDO 0x0040
 * #define FOF_FILESONLY 0x0080 // on .*, do only files
 * #define FOF_SIMPLEPROGRESS 0x0100 // means don't show names of files
 * #define FOF_NOCONFIRMDELETE 0x0200 // don't confirm making any needed dirs
 *
 * typedef UINT FILEOP_FLAGS;
 *
 * // printer operations
 *
 * #define PO_DELETE 0x0013 // printer is being deleted
 * #define PO_RENAME 0x0014 // printer is being renamed
 * #define PO_PORTCHANGE 0x0020 // port this printer connected to is being changed
 *                                      // if this id is set, the strings received by
 *                                      // the copyhook are a doubly-null terminated
 *                                      // list of strings. The first is the printer
 *                                      // name and the second is the printer port.
 *
 * #define PO_REN_PORT 0x0034 // PO_RENAME and PO_PORTCHANGE at same time.
 *
 * // no POF_ flags currently defined
 *
 * typedef UINT PRINTEROP_FLAGS;
 *
 * #endif // FO_MOVE
 *
 * DECLARE_INTERFACE_(ICopyHook, IUnknown) // sl
 *
 * { // *** IUnknown methods ***
 *     STDMETHOD(QueryInterface)(THIS_ REFIID riid, LPVOID * ppvObj) PURE;
 *     STDMETHOD_(ULONG, AddRef)(THIS) PURE;
 *     STDMETHOD_(ULONG, Release)(THIS) PURE;
 *
 *     STDMETHOD_(UINT, CopyAllback)(THIS_ HWND hwnd, UINT wFunc, UINT wFlags, LPCSTR pszSrcFile, DWORD dwSrcFileAttributes,
typedef ICopyHook * LPCOPYHOOK;

//........................................................................
// IFileViewerSite Interface
//........................................................................

#undef INTERFACE
#define INTERFACE IFileViewerSite

DECLARE_INTERFACE(IFileViewerSite)
{
    STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef) (THIS) PURE;
    STDMETHOD_(ULONG, Release) (THIS) PURE;

    STDMETHOD(SetPinnedWindow) (THIS_ HWND hwnd) PURE;
    STDMETHOD(GetPinnedWindow) (THIS_ HWND * phwnd) PURE;
};

typedef IFileViewerSite * LPFILEVIEWERSITE;

//........................................................................
// IFileViewer Interface
//........................................................................

// Implemented in a FileViewer component object. Used to tell a
// FileViewer to PrintTo or to view, the latter happening though
// ShowInitialize and Show. The filename is always given to the
// viewer through IPersistFile.
//........................................................................

#undef INTERFACE
#define INTERFACE IFileViewer

typedef struct
{
    // Stuff passed into viewer (in)
    DWORD cbSize;  // Size of structure for future expansion...
    HWND hwndOwner; // who is the owner window.
    int iShow;  // The show command

    // Passed in and updated (in/out)
    DWORD dwFlags; // flags
    RECT rect; // where to create the window may have defaults
    LPUNKNOWN punkRel; // Release this interface when window is visible

    // Stuff that might be returned from viewer (out)
    OLECHAR strNewFile[MAX_PATH]; // New File to view.
} FVSHOWINFO, *LPFVSHOWINFO;

// Define File View Show Info Flags.
#define FVSIF_RECT 0x00000001 // The rect variable has valid data.
#define FVSIF_PINNED 0x00000002 // we should Initialize pinned
shlobj_102894

#define FVSIF_NEWFAILED 0x08000000 // The new file passed back failed
  // to be viewed.
#define FVSIF_NEWFILE  0x80000000 // A new file to view has been returned
#define FVSIF_CANVIEWIT 0x40000000 // The viewer can view it.

DECLARE_INTERFACE(IFileViewer)
{
    STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddrRef) (THIS) PURE;
    STDMETHOD_(ULONG, Release) (THIS) PURE;

    STDMETHOD(ShowInitialize) (THIS_ LPFILEVIEWERSITE lpfsi) PURE;
    STDMETHOD(Show) (THIS_ LPFVSHOWINFO pvsi) PURE;
    STDMETHOD(PrintTo) (THIS_ LPSTR pszDriver, BOOL fSuppressUI) PURE;
};

typedef IFileViewer * LPFILEVIEWER;

//---

// struct STRRET
//
// structure for returning strings from IShellFolder member functions
//
//---

#define STRRET_WSTR 0x0000
#define STRRET_OFFSET 0x0001
#define STRRET_CSTR   0x0002

typedef struct _STRRET
{
    UINT uType; // One of the STRRET_* values
    union
    {
        LPWSTR pOLEstr;  // OLESTR that will be freed
        UINT uOffset;   // Offset into SHITEMID (ANSI)
        char cStr[MAX_PATH]; // Buffer to fill in
    }
} STRRET, *LPSTRRET;

//---

// SHGetPathFromIDList
//
// This function assumes the size of the buffer (MAX_PATH). The pidl
// should point to a file system object.
//
//---

BOOL WINAPI SHGetPathFromIDList(LPCITEMIDLIST pidl, LPSTR pszPath);

//---

// SHGetSpecialFolderLocation
//


shlobj_102894

// Caller should call SHFree to free the returned pidl.

//------------------------------------------------------------------------------

// registry entries for special paths are kept in:
#define REGSTR_PATH_SPECIAL_FOLDERS  REGSTR_PATH_EXPLORER "\Shell Folders"

#define CSIDL_DESKTOP 0x0000
#define CSIDL_PROGRAMS 0x0002
#define CSIDL_CONTROLS 0x0003
#define CSIDL_PRINTER 0x0004
#define CSIDL_PERSONAL 0x0005
#define CSIDL_STARTUP 0x0007
#define CSIDL_RECENT 0x0008
#define CSIDL_SENDTO 0x0009
#define CSIDL_BITBUCKET 0x000a
#define CSIDL_STARTMENU 0x000b
#define CSIDL_DESKTOPDIRECTORY 0x0010
#define CSIDL_DRIVES 0x0011
#define CSIDL_NETWORK 0x0012
#define CSIDL_NETHOOD 0x0013
#define CSIDL_FONTS 0x0014
#define CSIDLTEMPLATES 0x0015

HRESULT WINAPI SHGetSpecialFolderLocation(HWND hwndOwner, int nFolder, LPITEMIDLIST * ppidl);

//------------------------------------------------------------------------------

// old API get rid of it.

//------------------------------------------------------------------------------

HICON WINAPI SHGetFileIcon(HINSTANCE hinst, LPCSTR pszPath, DWORD dwFileAttribute, UINT uFlags);

//------------------------------------------------------------------------------

// SHBrowseForFolder API

//------------------------------------------------------------------------------

typedef int (CALLBACK* BFFCALLBACK)(HWND hwnd, UINT uMsg, LPARAM lParam, LPARAM lpData);

typedef struct _browseinfo {
    HWND hwndOwner;
    LPITEMIDLIST pidlRoot;
    LPSTR pszDisplayName; // Return display name of item selected.
    LPCSTR lpstrTitle; // resource (or text to go in the banner over the
tree.
    UINT uFlags; // Flags that control the return stuff
    BFFCALLBACK lpfn;
    LPARAM lParam; // extra info that's passed back in callbacks
    int iImage; // output var: where to return the Image index.
} BROWSEINFO, *PDBROWSEINFO, *LPBROWSEINFO;

//------------------------------------------------------------------------------

// Browsing for directory.
#define BIF_RETURNONLYFSDIRS 0x0001 // For finding a folder to start document searching
#define BIF_DONTGOBELOWDOMAIN 0x0002 // For starting the Find Computer
#define BIF_STATUSTEXT 0x0004
#define BIF_RETURNANCESTORS 0x0008

Page 14
LPITEMIDLIST WINAPI SHBrowseForFolder(LPBROWSEINFO lpbi);

SHLoadInProc

HRESULT WINAPI SHLoadInProc(REFCLSID rclsid);

typedef struct IEnumIDList *LPENUMIDLIST;

DECLARE_INTERFACE_(IEnumIDList, IUnknown)
{
    // *** IUnknown methods ***
    STDMETHOD(QueryInterface) (THIS_ REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD_(ULONG, AddRef) (THIS) PURE;
    STDMETHOD_(ULONG, Release) (THIS) PURE;

    // *** IEnumIDList methods ***
    STDMETHOD(Next) (THIS_ ULONG celt, LPITEMIDLIST *rgelt, ULONG *pceltFetched) PURE;
    STDMETHOD(Skip) (THIS_ ULONG celt) PURE;
    STDMETHOD(Reset) (THIS) PURE;
    STDMETHOD(Clone) (THIS_ IEnumIDList **ppenum) PURE;
};
shlobj_102894

IShellFolder::BindToObject(pidl, pbc, riid, ppvOut)
// This function returns an instance of a sub-folder which is specified
// by the IDList (pidl).

IShellFolder::BindToStorage(pidl, pbc, riid, ppvObj)
// This function returns a storage instance of a sub-folder which is
// specified by the IDList (pidl). The shell never calls this member
// function in the first release of Chicago.

IShellFolder::CompareIDs(lParam, pidl1, pidl2)
// This function compares two IDLists and returns the result. The shell
// explorer always passes 0 as lParam, which indicates "sort by name".
// It should return 0 (as CODE of the scode), if two id indicates the
// same object; negative value if pidl1 should be placed before pidl2;
// positive value if pidl2 should be placed before pidl1.

IShellFolder::CreateViewObject(hwndOwner, riid, ppvOut)
// This function creates a view object of the folder itself. The view
// object is a difference instance from the shell folder object.

IShellFolder::GetAttributesOf(cidl, apidl, prgfloInOut)
// This function returns the attributes of specified objects in that
// folder. "cidl" and "apidl" specifies objects. "apidl" contains only
// simple IDLists. The explorer initializes *prgFloInOut with a set of
// flags to be evaluated. The shell folder may optimize the operation
// by not returning unspecified flags.

IShellFolder::GetUIObjectOf(hwndOwner, cidl, apidl, riid, prgfloInOut, ppvOut)
// This function creates a UI object to be used for specified objects.
// The shell explorer passes either IID_IDataObject (for transfer operation)
// or IID_IContextMenu (for context menu operation) as riid.

IShellFolder::GetDisplayNameOf
// This function returns the display name of the specified object.
// If the ID contains the display name (in the locale character set),
// it returns the offset to the name. Otherwise, it returns a pointer
// to the display name string (UNICODE), which is allocated by the
// task allocator, or fills in a buffer.

IShellFolder::SetDisplayNameOf
// This function sets the display name of the specified object.
// If it changes the ID as well, it returns the new ID which is
// allocated by the task allocator.

#ifndef INTERFACE
#define INTERFACE     IShellFolder

// IShellFolder::GetDisplayNameOf/SetNameOf uFlags
typedef enum tagSHGDN
{
    SHGDN_NORMAL       = 0,  // default (display purpose)
    SHGDN_IN_FOLDER    = 1,  // displayed under a folder (relative)
    SHGDN_NO_EXTENSION = 0,  // no extension (BUGBUG: being removed)
    SHGDN_DIRPARSING   = 0x8000, // for ParseDisplayName or path
} SHGDN;

// IShellFolder::EnumObjects
typedef enum tagSHCONTF
{
    SHCONTF_FOLDERS       = 32, // for shell browser

Page 16
shlobj_102894
SHCONTF_NONFOLDERS = 64, // for default view
SHCONTF_INCLUDEHIDDEN = 128, // for hidden/system objects
}

// IShellFolder::GetAttributesOf flags
#define SFGAO_CANCOPY DROPFFFECT_COPY // Objects can be copied
#define SFGAO_CANMOVE DROPFFFECT_MOVE // Objects can be moved
#define SFGAO_CANLINK DROPFFFECT_LINK // Objects can be linked
#define SFGAO_CANRENAMEN 0x00000010L // Objects can be renamed
#define SFGAO_CANDELETE 0x00000020L // Objects can be deleted
#define SFGAO_HASPROPSHEET 0x00000040L // Objects have property sheets
#define SFGAO_DROPTARGET 0x00000100L // Objects are drop target
#define SFGAO_CAPABILITYMASK 0x0000017FL
#define SFGAO_LINK 0x00010000L // Shortcut (link)
#define SFGAO_SHARE 0x00020000L // shared
#define SFGAO_READONLY 0x00040000L // read-only
#define SFGAO_GHOSTED 0x00080000L // ghosted icon
#define SFGAO_DISPLAYATTACK MASK 0x000F0000L
#define SFGAO_FILESYANCF OFER 0x10000000L // It contains file system folder (file/folder/root)
#define SFGAO_FOLDER 0x20000000L // It's a folder.
#define SFGAO_FILESYSTEM 0x40000000L // is a file system thing
#define SFGAO_HASSUBFOLDER 0x80000000L // Expandable in the map pane
#define SFGAO_CONTENTSMASK 0x80000000L
#define SFGAO_VALIDATE 0x01000000L // invalidate cached information
#define SFGAO_REMOVABLE 0x02000000L // is this removable media?

DECLARE_INTERFACE_(IShellFolder, IUnknown)
{
    // *** IUnknown methods ***
    STDMETHOD(QueryInterface) (THIS REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD (_ULONG, AddRef) (THIS) PURE;
    STDMETHOD (_ULONG, Release) (THIS) PURE;

    // *** IShellFolder methods ***
    STDMETHOD(ParseDisplayName) (THIS_ Hwnd hwndOwner, LPOLESTR 1pszDisplayName,
        ULONG * pchEaten, LPIITEMIDLIST * ppidl, ULONG * pdwAttributes) PURE;
    STDMETHOD (EnumObjects) (THIS_ Hwnd hwndOwner, DWORD grfFlags, LPIENUMIDLIST * ppenumIDList) PURE;
    STDMETHOD (BindToObject) (THIS_ LPIITEMIDLIST pidl, LPOLESTR bcbReserved, REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD (BindToStorage) (THIS_ LPIITEMIDLIST pidl, LPOLESTR bcbReserved, REFIID riid, LPVOID * ppvObj) PURE;
    STDMETHOD (CompareIDs) (LPCITEMIDLIST pidl1, LPCITEMIDLIST pidl2) PURE;
    STDMETHOD (CreateViewObject) (THIS_ Hwnd hwndOwner, REFIID riid, LPVOID * ppvOut) PURE;
    STDMETHOD (GetAttributesOf) (THIS_ UINT cidl, LPCITEMIDLIST * apidl, REFIID riid, UINT * prgfnOut, LPVOID * ppvOut) PURE;
    STDMETHOD (GetUIObjectOf) (THIS_ Hwnd hwndOwner, UINT cidl, LPCITEMIDLIST * apidl, REFIID riid, UINT * prgfnOut, LPVOID * ppvOut) PURE;
    STDMETHOD (GetDisplayNameOf) (THIS_ LPIITEMIDLIST pidl, DWORD uFlags, LPSTRRET lpname) PURE;
    STDMETHOD (SetNameOf) (THIS_ Hwnd hwndOwner, LPCITEMIDLIST pidl, LPPOLESTR lpszName, DWORD uFlags, LPIITEMIDLIST * ppidlOut) PURE;
};
typedef IShellFolder * LPSHELLFOLDER;

// Helper function which returns a IShellFolder interface to the desktop
// folder. This is equivalent to calling CoCreateInstance with CLSID_ShellDesktop.
// CoCreateInstance(CLSIDDesktop, NULL, 
// CLSCTX_INPROC, IID_IShellFolder, &pshf);
//
HRESULT WINAPI SHGetDesktopFolder(LPSHELLFOLDER *ppshf);

#define CFSTR_SHELLIDLIST "Shell IDList Array" // CF_IDLIST
#define CFSTR_SHELLIDLISTOFFSET "Shell Object Offsets" // CF_OBJECTPOSITIONS
#define CFSTR_NETRESOURCES "Net Resource" // CF_NETRESOURCE
#define CFSTR_FILEDESCRIPTOR "FileGroupDescriptor" // CF_FILEGROUPDESCRIPTOR
#define CFSTR_FILECONTENTS "FileContents" // CF_FILECONTENTS
#define CFSTR_FILENAME "FileName" // CF_FILENAME
#define CFSTR_PRINTERGROUP "PrinterFriendlyName" // CF_PRINTERGROUP

// CF_OBJECTPOSITIONS

#define DVASPECT_SHORTNAME 2 // use for CF_HDROP to get short name version
// format of CF_NETRESOURCE
//
typedef struct _NRESARRAY {
    UINT citems;
    NETRESOURCE nr[1];
} NRESARRAY, *LPNRESARRAY;

// format of CF_IDLIST
//
typedef struct _IDA {
    UINT cid;
    UINT aoffset[1]; // [0]: folder IDList, [1][cid]: item IDList
} IDA, *LPIDA;

// FILEDESCRIPTOR.dwFlags field indicate which fields are to be used
//
typedef enum {
    FD_CLSID = 0x0001,
    FD_SIZEPOINT = 0x0002,
    FD_ATTRIBUTES = 0x0004,
    FD_CREATETIME = 0x0008,
    FD_ACCESSTIME = 0x0010,
    FD_WRITETIME = 0x0020,
    FD_FILESIZE = 0x0040,
} FD_FLAGS;
typedef struct _FILEDESCRIPTOR { // ffd
    DWORD dwFlags;
    CLSID clsid;
    SIZEĽ szsize;
    POINTI pointl;
    DWORD dwFileAttributes;
    FILETIME ftCreationTime;
    FILETIME ftLastAccessTime;
    FILETIME ftLastWriteTime;
    DWORD nFileSizeHigh;
    DWORD nFileSizeLow;
    CHAR cFileName[ MAX_PATH ];
} FILEDESCRIPTOR, *LPFILEDESCRIPTOR;

// format of CF_FILEGROUPDESCRIPTOR
//
typedef struct _FILEGROUPDESCRIPTOR { // fgd
    UINT cItems;
    FILEDESCRIPTOR fgd[1];
} FILEGROUPDESCRIPTOR, *LPFILEGROUPDESCRIPTOR;

// format of CF_HDROP and CF_PRINTER, in the HDROP case the data that follows
// is a double null terminated list of file names, for printers they are printer
// friendly names
//
typedef struct _DROPFILES {
    WORD pFiles; // offset to double nul list of files
    POINTI pt; // drop point (client coords)
    WORD fNC; // is it on non client area
} DROPFILES, *LPDROPFILES;

////// File System Notification APIs //////////////

//

// File System Notification flags

#define SHCNE_RENAME 0x00000001L // GOING AWAY
#define SHCNE_RENAMEITEM 0x00000001L
#define SHCNE_CREATE 0x00000002L
#define SHCNE_DELETE 0x00000004L
#define SHCNE_MKDIR 0x00000008L
#define SHCNE_RMDIR 0x00000010L
#define SHCNE_MEDIAINSERTED 0x00000020L
#define SHCNE_MEDIAREMOVED 0x00000040L
#define SHCNE_DRIVEREMOVED 0x00000080L
#define SHCNE_DRIVEADD 0x00000100L
#define SHCNE_NETSHARE 0x00000200L
#define SHCNE_NETUNSHARE 0x00000400L
#define SHCNE_ATTRIBUTES 0x00000800L
#define SHCNE_UPDATEDIR 0x00001000L
// Flags
// 
// uFlags & SHCNF_TYPE is an ID which indicates what dwItem1 and dwItem2 mean
#define SHCNF_IDLIST 0x0000 // LPITEMIDLIST
#define SHCNF_PATH 0x0001 // path name
#define SHCNF_PRINTER 0x0002 // printer friendly name
#define SHCNF_DWORD 0x0003 // DWORD
#define SHCNF_TYPE 0x00FF
#define SHCNF_FLUSH 0x1000
#define SHCNF_FLUSHNOWAIT 0x2000

// APIs
//
void WINAPI SHChangeNotify(LONG wEventId, UINT uFlags,
                            LPCVOID dwItem1, LPCVOID dwItem2);

// SHAddToRecentDocs
//
#define SHARD_PIDL 0x00000000L
#define SHARD_PATH 0x00000002L

void WINAPI SHAddToRecentDocs(UINT uFlags, LPCVOID pv);

HRESULT WINAPI SHGetInstanceExplorer(IUnknown **ppunk);

#ifdef _cplusplus
}
#endif /* _cplusplus */
#ifdef RC_INVOKED
#pragma pack()
#endif /* !RC_INVOKED */
#endif // _SHOBJ_H_