Directory Opus 9

VFS Plugin SDK 2.0

http://www.gpsoft.com.au

PO Box 570
Ashgrove, QLD 4060
Australia

Table Of Contents

Introduction

The Directory Opus 9 VFS Plugin SDK lets you extend Directory Opus by writing *VFS plugins* to add support for additional archive formats, virtual file systems and any other devices or methods of representing data in a folder/file hierarchy.

The plugin API describes a *filesystem-like* interface which includes matching functions for many common Windows API functions (e.g. FindFirstFile, FindNextFile, CreateFile, ReadFile, WriteFile, etc.). Not all functions need to be implemented, however; the plugin provides a *capabilities mask* which indicates which functionality it supports.

VFS plugins are invoked by Directory Opus whenever the user attempts to browse to a file or folder that a plugin has indicated it supports. When Opus starts it initializes the plugins, each of which fills out an information structure describing which file extensions it handles. Optionally, a plugin can implement an entire virtual namespace by specifying a URL-style path prefix (e.g. **coll://** is used to represent the virtual File Collection namespace).

Plugins using the path prefix method will be invoked automatically whenever the user enters a matching path into an Opus Lister's location field. On the other hand, plugins that operate on files via specific file extensions may require additional user-configuration via the Opus File Types system, usually in the shape of setting the **dblclk** action to run the **Go** command..

VFS plugins are DLLs which exist in the *VFSPlugins* sub-directory of the Directory Opus program files folder. The Preferences Dialog's Plugins section lists all VFS plugins and allow users to enable, disable and, if the plugin supports it, configure them, *Viewer* plugins, which are covered by a separate SDK, are shown in the Preferences dialog under a separate tab.

Accompanying this SDK is the source to the Directory Opus **RAR** plugin. This plugin uses the freely distributable UnRAR.dll component and as such only supports extraction from RAR files. While it does not support creating or modifying RAR files it should still serve as a good example of a VFS Plugin.

Changes from Version 1

In addition to the *filesystem-like* functions supported by Directory Opus 8, version 2 of the VFS Plugin API (which requires Directory Opus 9) also offers *batch-mode* support. This lets a plugin perform an action (eg, add to archive, extract from archive, etc) on multiple files in one go. This is ideal when writing plugins that interface to third-party archiving libraries that only provide simple atomic functions like “extract these files to this location” or “add these files to this archive”.

Several new functions have been added to the API as of version 2. They are:

- VFS_BatchOperation (A/W)
- VFS_USBSafe
- VFS_Init
- VFS_Uninit

The **VFSPLUGININFO** structure has several new fields:

- dwFlags – addition of the **VFSF_NONREENTRANT** flag value
- dwOpusVerMajor / dwOpusVerMinor
- dwInitFlags
- hIconSmall / hIconLarge

There are three new properties values for **VFS_GetProp**:

- VFSPROP_BATCHOPERATION
- VFSPROP_GETVALIDACTIONS
- VFSPROP_SHOWPICTURESDIRECTLY

Directory Opus 9 also contains a new Plugin Support API, which consists of a set of helper and utility functions exported directly from the Directory Opus executable. Your plugins can use these functions to interface with Opus and use some of the Opus functionality directly. See the separate Plugin Support API SDK for more information.

Limitations of this API

Version 1 of this API did not offer *batch-mode* support which was a major limitation to the performance of some plugins. Version 2 (which requires Directory Opus 9) removes this limitation.

Plugins are currently unable to add items to the Folder Tree, and also can not serve as the source of a drag-and-drop operation to outside of Directory Opus. (Drag-and-drop within Opus itself is supported, however.)

Exported Functions

The VFS Plugin API currently consists of over 40 separate functions (many of which can be provided as ANSI, or Unicode, or both). Not all functions are necessary for a functioning plugin and missing functions *generally* only mean the equivalent user-level action is unavailable. For example, if you do not support VFS_SetFileComment then the user won't be able to set comments for files inside your plugin but other operations, such as renaming and copying files, will be unaffected, assuming they themselves are supported. Plugin authors must decide which functions to implement based on their importance and their relevance to the underlying objects which the plugin deals with.

The functions a VFS plugin can export are listed below. See the following pages for full descriptions. "A/W" means ANSI and Unicode (Wide) versions are allowed..

VFS_Init - initialize the plugin
VFS_Uninit - uninitialize the plugin
VFS_USBSafe - mark plugin as safe for USB export
VFS_Identify (A/W) - identifies the plugin
VFS_QueryPath (A/W) - can the plugin handle a specified path?
VFS_GetPrefixList (A/W) - get a list of handled path prefixes
VFS_Create - create a plugin *namespace instance*
VFS_Clone - clone an existing instance
VFS_Destroy - destroy an instance
VFS_GetCapabilities - get plugin capabilities
VFS_GetCustomColumns (A/W) - get any custom columns
VFS_GetLastError - get last error that occurred
VFS_ReadDirectory (A/W) - read the contents of a directory
VFS_GetPathDisplayName (A/W) - get a version of a path string for display
VFS_GetFileDescription (A/W) - get description for a file
VFS_GetFileIcon (A/W) - get icon for a file
VFS_GetFileInformation (A/W) - get information about a file
VFS_GetFileSize (A/W) - get the size of a file
VFS_BatchOperation (A/W) - perform batch operations
VFS_CreateFile (A/W) - open a file for reading or writing
VFS_ReadFile - read from a file
VFS_WriteFile - write to a file
VFS_SeekFile - seek within a file
VFS_CloseFile - close a file
VFS_MoveFile (A/W) - move or rename a file
VFS_DeleteFile (A/W) - delete a file
VFS_SetFileTime (A/W) - change the timestamp of a file
VFS_SetFileAttr (A/W) - set file attributes
VFS_GetFileAttr (A/W) - get file attributes
VFS_SetFileComment (A/W) - set file comment
VFS_GetFileComment (A/W) - get file comment
VFS_CreateDirectory (A/W) - create a directory
VFS_RemoveDirectory (A/W) - delete a directory
VFS_FindFirstFile (A/W) - find first file matching a pattern
VFS_FindNextFile (A/W) - find the next file
VFS_FindClose (A/W) - cleanup from FindFirstFile
VFS_PropGet (A/W) - get properties
VFS_ContextVerb (A/W) - perform an action on a file
VFS_ExtractFiles (A/W) - extract files to disk
VFS_Properties (A/W) - display Properties dialog for a file
VFS_GetContextMenu (A/W) - build context menu for a file
VFS_GetDropMenu (A/W) - build drop menu for a file
VFS_GetPathParentRoot (A/W) - get parent or root of a path
VFS_GetFreeDiskSpace (A/W) - get free disk space information
VFS_Configure - configure the plugin
VFS_About - display about dialog

Plugin Initialization and USB Export

When Opus first initializes your VFS Plugin, it calls the **VFS_Init** function (if exported). Exporting this function lets you perform initialization that it is not safe to perform within the standard **DllMain** function (eg, opening libraries.) Similarly, **VFS_Uninit** is called when Opus is about to release your plugin.

To support the new USB export feature of Opus 9, plugins need to mark themselves as being “USB-safe”. This generally means that plugins must not write to the registry, but instead store their configuration data in XML format using the routines provided by the Opus Plugin Support API. To mark your plugin as USB-safe you need to export the **VFS_USBSafe** function. This function also lets you indicate to Opus the names of any additional files that must be exported along with your plugin (eg, the rar.dll plugin also requires the unrar.dll library to be present.)

Plugin Identification

When Directory Opus starts up it calls the **VFS_Identify** function for each installed plugin. Each plugin responds by identifying itself and providing information on its capabilities and the types of files or virtual namespaces that it can browse.

BOOL VFS_Identify(LPVFSPLUGININFO lpVFSInfo)

Directory Opus passes you the address of a **VFSPLUGININFO** structure and you must fill out the applicable members of this structure before returning. A return code of **TRUE** indicates that the plugin is active; a return of **FALSE** means that Directory Opus will not call the plugin again in this session.

The members of the **VFSPLUGININFO** structure are as follows:

- **UINT cbSize**

  Directory Opus initializes this field to the size of the structure before passing it to your function. You must only fill in fields in the structure up to the size supplied by Directory Opus. This allows your plugin to remain compatible with future versions of Directory Opus that may support more fields in this structure as well as with older versions of Directory Opus which may support fewer fields.

- **GUID idPlugin**

  This field must be initialized with your plugin's GUID (Globally Unique Identifier). This is used by Directory Opus to distinguish between installed VFS plugins. You can create a GUID using the Microsoft tool provided with Visual Studio and other development packages.

- **DWORD dwVersionHigh, dwVersionLow**

  These fields can be used to indicate to Directory Opus the version number of your plugin. Opus does not use these values directly; however, they are displayed to the user in the standard About dialog for your plugin.

- **DWORD dwFlags**

  The flags field is used to indicate the behavior of your plugin. The currently defined flags are:

  - **VFSF_CANCONFIGURE**

    This flag indicates that your plugin exports the **VFS_Configure** function. Directory Opus will enable the *Configure* option in the plugin context menu and in the Plugins section of the Opus Preferences dialog when your plugin is selected. The **VFS_Configure** function will be called when the user chooses to configure your plugin via either method.

  - **VFSF_CANSHOWABOUT**

    This flag indicates that your plugin exports the **VFS_About** function to display a custom About dialog. If not specified, Directory Opus will display a standard About dialog for your plugin using the information you provide in the other fields of this structure.

- **VFSF_MULTIPLEFORMATS**

  This flag indicates that your plugin can handle multiple file or archive formats. The **lpszHandleExts** should be set to ***** if this flag is specified. Instead of looking for specific file extensions before invoking your plugin, Opus will call your plugin's **VFS_QueryPath** function to enquire whether you can handle a specific file or file prefix.

- **VFSF_NONREENTRANT**

  This flag indicates that your plugin is non-reentrant. That is, that it does not support Directory Opus calling more than one exported function simultaneously from different threads. If you set this flag Opus will use semaphore locking when calling the exported functions in your plugin to ensure that only one thread is calling your plugin at any one time.

- **DWORD dwCapabilities**

  This field indicates, via a bitmask, the *capabilities* of your VFS plugin. The capabilities flags indicate which functions your plugin provides and in some cases is used to optimize certain operations. The capabilities flags provided in this structure define the over-all capabilities of your plugin. If you are using the **VFSF_MULTIPLEFORMATS** flag then you can also provide format-specific capabilities via the **VFS_GetCapabilities** function. The currently defined capabilities flags are:

  - **VFSCAPABILITY_MOVEBYRENAME**

    Indicates that your plugin is able to move files by renaming them. If this flag is specified and the user attempts to move a file from one folder to another, Opus will call your **VFS_MoveFile** function to perform the operation. If this flag is not specified, moving a file will involve creating a copy of it and then deleting the original.

  - **VFSCAPABILITY_COPYINDEFINITESIZES**

    Indicates that the file sizes reported by your plugin are not necessarily accurate. This is used when the user copies files from your plugin. If this flag is set Opus will ignore the stated size of a file and continue to read data until you return an end-of-file indicator. If this flag is not set Opus will only read the number of bytes you have reported for the size of the file.

  - **VFSCAPABILITY_CANRESUMECOPIES**

    Indicates that your plugin can resume interrupted file transfers (or file copy operations). If this flag is set and the user attempts to resume a file transfer either to or from your plugin's namespace, Opus will call your **VFS_SeekFile** function to position the file pointer appropriately.

  - **VFSCAPABILITY_TRIGGERRESUME**

    Set this flag if you want your plugin to be the *trigger* for a resume of an interrupted copy. If you set this flag and the user attempts to copy a file to or from your namespace that already exists, Opus will give them the option of resuming the transfer. As an example of how this flag differs from **VFSCAPABILITY_CANRESUMECOPIES**, the internal FTP namespace in Opus sets both flags, whereas the standard file system namespace only sets **VFSCAPABILITY_CANRESUMECOPIES**. This means that a user copying an existing file between two file system folders will not be given the option of resuming, but a user copying from or to an FTP namespace will be asked if they wish to resume or not. The standard file system namespace supports resume but only the FTP namespace triggers the option.

- **VFSCAPABILITY_POSTCOPYREREAD**

  If this flag is set Opus will automatically trigger a refresh of the destination Lister whenever files are copied to it or removed from it.

- **VFSCAPABILITY_CASESENSITIVE**

  Set this flag if your plugin's file names and paths are case-sensitive.

- **VFSCAPABILITY_RANDOMSEEK**

  Set this flag if you support random seeking within files. If you only support sequential seeking, or do not support seeking at all, do not set this flag. Opus does not generally use or require random seeking, but some viewer plugins may require the ability.

- **VFSCAPABILITY_FILEDESCRIPTIONS**

  Set this flag if you want your plugin to be able to provide description strings for files in its namespace. Opus will call your **VFS_GetFileDescription** function to retrieve descriptions for files (for example, if the user displays the Description column in a Lister). Note that the strings returned by this function are not (necessarily) user-supplied comments – they can contain any information you desire. A separate function pair (**VFS_GetFileComment** / **VFS_SetFileComment)** is used to implement user-editable comments.

- **VFSCAPABILITY_ALLOWMUSICCOLUMNS**

  Set this flag if you want the music-related Lister information fields to be available from your plugin. Note that Opus does not call your plugin to provide this information – it uses its own routines to open files in your namespace and parse them for the needed information. You can use the **VFS_GetCustomColumns** function to provide your own information columns for display in Listers.

- **VFSCAPABILITY_ALLOWIMAGECOLUMNS**

  Similar to the **VFSCAPABILITY_ALLOWMUSICCOLUMNS** flag, this flag indicates that you want the image-related Lister information fields to be available in your plugin's namespace.

- **VFSCAPABILITY_ALLOWEXTRADATECOLUMNS**

  This flag indicates that you want the last accessed and creation date fields to be available in your plugin's namespace. If this flag is not set the only date fields available will be for last modified date.

- **VFSCAPABILITY_LETMEDOPARENTS**

  If this flag is set then Opus will call your **VFS_GetPathParentRoot** function whenever it needs to calculate the parent or root of a path in your plugin's namespace. If not specified, Opus will apply standard parsing rules to calculate the desired path.

- **VFSCAPABILITY_COMBINEDPROPERTIES**

  Set this flag if your plugin is able to display a combined Properties sheet for multiple files. If this flag is set and the user requests the properties of multiple files at once in your plugin's namespace, Opus will call your **VFS_Properties** function with a double-null terminated list of files to display properties for. If this flag is not set, Opus will call your **VFS_Properties** function once for each selected file.

- **VFSCAPABILITY_COMPARETIMENOSECONDS**

  Set this flag if your plugin does not report or preserve seconds in file times. If this flag is set Opus will discard or ignore the seconds of any file times it needs to compare with times provided by your plugin.

- **VFSCAPABILITY_GETBATCHFILEINFO**

  Set this flag if your plugin is able to handle asynchronous requests for file information (which may involve opening and reading file data). If this flag is not set, Opus will launch a background thread to read all required or desired file information for all files in the folder whenever a directory is read. If this flag is not set, Opus will only request file information when needed.

- **VFSCAPABILITY_SLOW**

  Set this flag if your plugin represents a slow device or media. This flag is passed to viewer plugins as an indication that accesses to your plugin's namespace may take longer than expected. Additionally, Opus will not attempt to determine file type by reading the contents of the file on a slow device, and may also refrain from attempting to extract some file information in some cases.

- **VFSCAPABILITY_MULTICREATEDIR**

  Set this flag if your plugin supports the creation of multiple directories simultaneously. If set, your **VFS_CreateDirectory** function should be able to handle a comma-separated list of folders to create.

- **VFSCAPABILITY_ALLOWFILEHASH**

  Set this flag if you want your plugin to allow the hashing of files within its namespace. The actual hashing is performed by Directory Opus (currently using MD5 functions) – all that is required of your plugin if this flag is set is the ability to read sequentially from files within your namespace. If access to your files is particularly slow you may wish to disable the hash functionality.

- **VFSCAPABILITY_READONLY**

  This flag should be set if your plugin is *read-only* – that is, if you do not support the creation of, writing to or deleting of files within your plugin's namespace. The example unrar plugin sets this flag as the required support library does not support the creation of rar archives.

- **VFSCAPABILITY_CHECKAVAILONDIRCHANGE**

  If this flag is set Opus will call your **VFS_PropGet** function to retrieve the **VFSPROP_FUNCAVAILABILITY** property and update the state of any toolbar buttons whenever a new folder is read within your plugin's namespace. This allows you to selectively enable or disable file functions on a per-folder basis and have the user interface reflect this automatically.

- **LPTSTR lpszHandlePrefix**

  This field allows you to indicate to Directory Opus a URL-style prefix that you want to use to represent your plugin's namespace. For example, the internal File Collections system uses **coll://** as a namespace prefix. If you are using the **VFSF_MULTIPLEFORMATS** flag you can leave this field empty – Opus will call your **VFS_QueryPath** function with the full path to let your plugin determine if it wishes to handle it or not.

  When Directory Opus calls your plugin, this field may point to a buffer into which you can copy your information. The size of the buffer is given by the **cchHandlePrefixMax** field. You can either use this supplied buffer or change the address of the **lpszHandlePrefix** field to point to your own buffer. You **must** check before copying into this field that the buffer pointer is valid; if set to 0 it indicates that Directory Opus does not want this information at this time.

- **LPTSTR lpszHandleExts**

  This field allows you to indicate to Directory Opus the file name extensions that your plugin supports. This should be a semi-colon separated string of file extensions, for example **".tar;.gz"**. If you are using the **VFSF_MULTIPLEFORMATS** flag this string should be set to **"*"**, and Opus will then call your **VFS_QueryPath** function with the file extension of an unknown or unhandled file to let your plugin determine if it wishes to handle it or not.

  When Directory Opus calls your plugin, this field may point to a buffer into which you can copy your information. The size of the buffer is given by the **cchHandleExtsMax** field. You can either use this supplied buffer or change the address of the **lpszHandleExts** field to point to your own buffer. You **must** check before copying into this field that the buffer pointer is valid; if set to 0 it indicates that Directory Opus does not want this information at this time.

- **LPTSTR lpszName**

  This field lets you specify the "name" of your plugin. Ordinarily this will be the primary file format that your plugin supports, eg **TAR**. However, this can be any string. The name is displayed to the user in the Plugin list in Preferences as well as the standard About dialog for your plugin. It is also used as the parameter for certain Opus internal commands that take a plugin name on the command line.

  When Directory Opus calls your plugin, this field may point to a buffer into which you can copy your information. The size of this buffer is given by the **cchNameMax** field. You can either use this supplied buffer or change the address of the **lpszName** field to point to your own buffer. You **must** check before copying into this field that the buffer pointer is valid; if set to 0 it indicates that Directory Opus does not want this information at this time.

- **LPTSTR lpszDescription**

  Lets you specify a description for your plugin that is displayed to the user in the standard About dialog for your plugin. Can be any text string but should not ordinarily be more than about 50 characters in length.

  When Directory Opus calls your plugin, this field may point to a buffer into which you can copy your information. The size of this buffer is given by the **cchDescriptionMax** field. You can either use this supplied buffer or change the address of the **lpszDescription** field to point to your own buffer. You **must** check before copying into this field that the buffer pointer is valid; if set to 0 it indicates that Directory Opus does not want this information at this time.

- **LPTSTR lpszCopyright**

  Use this field to supply a copyright notice that is displayed to the user in the standard About dialog for your plugin.

  When Directory Opus calls your plugin, this field may point to a buffer into which you can copy your information. The size of this buffer is given by the **cchCopyrightMax** field. You can either use this supplied buffer or change the address of the **lpszCopyright** field to point to your own buffer. You **must** check before copying into this field that the buffer pointer is valid; if set to 0 it indicates that Directory Opus does not want this information at this time.

- **LPTSTR lpszURL**

  Use this field to supply a URL string that is displayed to the user in the standard About dialog for your plugin. This can be any URL, for example, the URL for the homepage on your website for your plugin.

  When Directory Opus calls your plugin, this field may point to a buffer into which you can copy your information. The size of this buffer is given by the **cchURLMax** field. You can either use this supplied buffer or change the address of the **lpszURL** field to point to your own buffer. You **must** check before copying into this field that the buffer pointer is valid; if set to 0 it indicates that Directory Opus does not want this information at this time.

- **UINT cchHandlePrefixMax, cchHandleExtsMax, cchNameMax, cchDescriptionMax, cchCopyrightMax, cchURLMax**

  These fields specify the supplied buffer size for the relevant string fields in the structure. You must make sure you do not copy more data into the supplied buffers than is specified in these fields. The size specified is given in characters, not bytes.

- **DWORD dwOpusVerMajor, dwOpusVerMinor**

  These fields (provided in Directory Opus 9 only) specify the current version of Directory Opus.

- **DWORD dwInitFlags**

  Specifies flags the plugin may use to modify its initialization. Currently defined flags are:

  - **VFSINITF_FIRSTTIME**

    This flag is set the first time a new plugin is initialized by Directory Opus.

  - **VFSINITF_USB**

    This flag is set when Directory Opus is running from a USB device.

- **HICON hIconSmall, hIconLarge**

  These fields can be used to provide a custom icon for the plugin that is displayed to the user on the Plugins page in Preferences. Note that Directory Opus will call **DestroyIcon()** on the icons provided here.

Namespace Identification

A *namespace* is, literally, a “space of names” – the virtual folder hierarchy provided by your VFS Plugin. When the user executes the **Go** command (or another internal Opus command), Directory Opus can identify that access to your plugin is needed in two different ways:

- By matching a filename extension to the list provided by you in the **lpszHandleExts** field in the **VFS_Identify** function, or
- By matching a URL-style prefix to that provided by you in the **lpszHandlePrefix** field.

If Opus is able to match a filename extension or path prefix in this manner, it will automatically instantiate an instance of your plugin’s namespace using the **VFS_Create** function.

One alternative identification method is the use of the **VFS_QueryPath** function. If you specify the **VFSF_MULTIPLEFORMATS** flag in the **dwFlags** field of the **VFSPLUGININFO** structure, Opus will call your **VFS_QueryPath** function whenever an unknown path is presented, to see if you are able to handle it.

BOOL VFS_QueryPath(LPTSTR lpszPath, BOOL fPrefix, LPGUID pGUID)

Directory Opus calls this function, passing it the path or filename extension in question. If **fPrefix** is set to **TRUE**, **lpszPath** will point to a string that begins with a URL-style prefix. If **fPrefix** is set to **FALSE**, **lpszPath** will point to a string containing a filename extension.

Either way, if your plugin is able to handle the path, you should return **TRUE** from this function. You also need to fill out the supplied **pGUID** pointer with a GUID to represent this “aspect” of your plugin. This **must** be different from the GUID you supplied in response to the **VFS_Identify** function. The GUID you provide here is passed to the **VFS_Create** function when instantiating your plugin’s namespace, and so should use a different GUID for each different type of file or prefix you are able to handle.

As an example, a plugin that handled ZIP files and RAR files would return different GUIDs for .zip and .rar files, as these are different type of archives. However, .jar files (java archives) are really .zip files, and so for .jar files you would return the same GUID as for .zip files, as there is no need for these two file extensions to be treated differently.

If you do not specify the **VFSF_MULTIPLEFORMATS** flag you do not need to provide a **VFS_QueryPath** function, and the GUID passed to **VFS_Create** will be the one you provided in response to the **VFS_Identify** function.

If you are using the multiple format interface, you may also wish to implement the **VFS_GetPrefixList** and **VFS_GetCapabilities** functions, which are described later in this document.

Namespace Creation

Once Directory Opus has determined that your plugin can handle a file or path, it will *instantiate* your plugin – that is, Opus will ask your plugin to create a unique instance that can be used separately and asynchronously from any other instance of your plugin. Don't forget that Opus is highly multi-threaded. The user may be using your plugin to access multiple archives or virtual namespaces simultaneously – operations such as file copying may be proceeding in the background while the user is browsing a folder in another Lister. ***Above all else, your plugin must be thread-safe!***

The function responsible for plugin namespace instantiation is **VFS_Create**.

HANDLE VFS_Create(LPGUID pGUID, HWND hwndMsgWindow)

Opus calls this function to create an instance of your plugin. The **pGUID** parameter will point to the GUID representing the format Opus is instantiating your plugin for. If you have not used the **VFSF_MULTIPLEFORMATS** flag then this will always be the GUID you provided to the **VFS_Identify** function. If, however, you are using the multiple formats interface, this will be the GUID that your **VFS_QueryPath** function returned when it indicated that it could handle a file or path. You should check the GUID provided to determine which plugin interface you need to instantiate.

The **hwndMsgWindow** parameter provides a window handle that you can save and use for communication with Directory Opus. Currently this parameter is unused.

When your **VFS_Create** function is called, you should allocate and initialize whatever private data structure or class you need to identify an instance of the indicated format. This structure should be cast to a **HANDLE** and returned to Opus. Directory Opus will pass this value in every call to practically every other function in your plugin.

Another function exists that Opus sometimes uses to instantiate your plugin – **VFS_Clone**. This is fundamentally identical to **VFS_Create** and in fact you do not need to provide a **VFS_Clone** function at all. However, Opus often needs to make "clones" of existing namespaces – for example, when you select a file in a Lister and click Copy, Opus launches a new thread which makes a clone of the primary namespace that is used by the Lister to display the folder. The **VFS_Clone** function is passed the **HANDLE** value representing the existing namespace instance, and you should implement the **VFS_Clone** function if you feel you can make a new namespace instance more efficiently if you are given a pointer to an existing one. If **VFS_Clone** is not implemented by a plugin Opus simply creates a brand new instance using **VFS_Create**.

Directory Opus calls the **VFS_Destroy** function to allow you to free your data once it is finished with an instance of your plugin.

Specifying Custom Columns

Directory Opus has many built-in file information that can be displayed in Listers. Many of these can be used with VFS plugins and function automatically. Others, like the Music and Image information fields, only work with plugins if you set the appropriate capabilities flags to enable them. Other columns, like the Program information fields, do not work with VFS plugins at all.

Directory Opus lets plugins provide their own list of custom information columns that display data provided by the plugin. These columns appear to the user in the Special column category in the **Folder Options** dialog.

To implement custom columns you must provide a **VFS_GetCustomColumns** function.

BOOL VFS_GetCustomColumns(HANDLE hVFSData)

Directory Opus calls this function soon after your namespace has been instantiated with **VFS_Create**. The **hVFSData** parameter is your instance handle, returned from **VFS_Create**. The **VFS_GetCustomColumns** function must return a pointer to a linked list of **VFSCUSTOMCOLUMN** structures, each one of which defines a custom column.

The members of the **VFSCUSTOMCOLUMN** structure are:

- **UINT cbSize**

  This field must contain the size of the **VFSCUSTOMCOLUMN** structure you are using. Directory Opus uses this to maintain backwards compatibility with older plugins.

- **LPVFSCUSTOMCOLUMN lpNext**

  This member points to the next **VFSCUSTOMCOLUMN** structure in the chain, or 0 if this is the last column to be defined.

- **LPTSTR lpszLabel**

  This field points to a null-terminated string containing the column label. This is the string that is displayed to the user in the Folder Options dialog, and in the column header in the Lister.

- **LPTSTR lpszKey**

  This field points to a null-terminated string containing a keyword for your column. This is the string that the user can use in Opus raw commands in relation to this column. For example, a column whose label was *Image Resolution* might have the keyword *imageres*. Opus raw commands like **Set COLUMNSADD** would then accept **imageres** as a valid parameter. You must make sure that this keyword does not clash with any internal Opus column keywords – we recommend that you prefix your keywords with the name of your plugin to ensure they are unique (eg *superplugin_imageres*).

- **DWORD dwFlags**

  This member lets you specify flags that control how the column data is displayed to the user. Currently defined flags are:

  - **VFSCCF_LEFTJUSTIFY**

    The column data is to be left-justified. This is the default.

  - **VFSCCF_RIGHTJUSTIFY**

    The column data is to be right-justified.

  - **VFSCCF_CENTERJUSTIFY**

    The column data is to be center-justified.

  - **VFSCCF_NUMBER**

    The column data is to be formatted as a number. Even though column data is always provided as a string, Opus will treat the contents as a number and format it using the current user's locale settings. This flag also ensures that numeric columns are sorted in correct numerical order.

  - **VFSCCF_PERCENT**

    The column data is to be formatted as a percentage. Currently this simply entails displaying the supplied string with a trailing percent sign. This flag also ensures that the column will be sorted in correct numerical order.

  - **VFSCCF_SIZE**

    The column data is to be formatted as a file size. The value you provide for the column data is assumed to be a number of bytes, and will be displayed to the user as either a byte value, a rounded-up kilobyte value, or an automatic value depending on the user's Preferences settings. This flag also ensures that the column will be sorted in correct numerical order.

- **int iID**

  This field is an ID code that your plugin will use when setting the column data of files in your plugin's namespace. You should give each column a unique ID, numbered sequentially and beginning from 1.

The column information you return from the **VFS_GetCustomColumns** function must remain valid for at least the lifetime of the namespace instance. Opus treats the pointer you return as read-only – it will not modify it or any of the data it points to but it may use the data and strings at any time. Therefore you should either define your columns as static data, or allocate the data when your namespace is instantiated and only free it when your namespace is destroyed.

If your plugin does not require custom columns you should simply not provide a **VFS_GetCustomColumns** function.

Indicating Success or Failure

Many functions in the VFS Plugin API are defined as type **BOOL**. Generally, this means they simply return **TRUE** (1) on success, or **FALSE** (0) on failure.

If a function fails for any reason, it should return **FALSE**, and indicate the reason for failure via the **VFS_GetLastError** function. Similar to the Windows **GetLastError** function, **VFS_GetLastError** is called by Directory Opus to retrieve the reason for the last error that occurred in the plugin namespace.

The error code that you return can be any of the standard Windows error codes (eg; **ERROR_FILE_NOT_FOUND**), or it can be one of a number of defined Opus-specific error codes. The following is a list of the currently defined codes and the error message shown to the user when they occur. Generally you should use one of the standard Windows error codes to indicate the reason for failure. Many (most) of the Opus-only error codes are specific to internal Opus functionality and do not have much relevance to VFS plugins, however they are presented here for completeness.

VFSERR_COPY_INTO_ITSELF	You can't copy or move a folder into itself.
VFSERR_NOT_SUPPORTED	The operation is not supported by this VFS.
VFSERR_MOVE_INTO_SAMEDIR	Source and destination must be different to move files.
VFSERR_DIR_ALREADY_EXISTS	Cannot create a folder when that folder already exists.
VFSERR_FILE_IS_DIR	File is really a folder.
VFSERR_BADLINK	Link does not point to a valid file.
VFSERR_NOTEXPORTFILE	File is not a valid exported Preferences file.
VFSERR_NORECYCLEBIN	Object can not be placed in the recycle bin and will be permanently deleted.
VFSERR_RECYCLETOOBIG	Object is too large to be placed in the recycle bin and will be permanently deleted.
VFSERR_NOPRINTHANDLER	File does not have a registered Print handler.
VFSERR_BADZIPFILE	Zip Compressed File is invalid or damaged.
VFSERR_GENERALERRMSG	General error.
VFSERR_WRITEPROTECTED	The disk is write protected.
VFSERR_WRITEPROTECTEDZIP	The Zip Compressed File is write protected.
VFSERR_ZIPISDIR	A folder already exists by that name.
VFSERR_CANTRENAMEFOLDERS	Folders cannot be renamed within ZIP files.
VFSERR_SHARINGVIOLATION	The file that you are trying to copy is in use by another process.
VFSERR_ALREADYINCOLL	The item is already in the collection.
VFSERR_CANTCOPYTOCOLLROOT	Files can not be added to the File Collections root folder.
VFSERR_NOMOVETOCOLLECTION	Files can not be moved into a collection.
VFSERR_NOJOINTOCOLLECTION	Files can not be joined into a collection.
VFSERR_NODROPDATATOCOLLECTION	Data-only files can not be added to a collection.
VFSERR_NOCOLLINCOLL	Collections can not be added to other collections.
VFSERR_FEATURENOTENABLED	The feature is not enabled.
VFSERR_UNKNOWNERROR	An unknown error occurred.
VFSERR_CANTCOPYFILEOVERITSELF	You can’t copy a file over itself.
VFSERR_CANTCHANGEZIPCASE	Limitations of the zip file format mean you can’t change the case of a filename within a zip file.
VFSERR_NOT_EXTRACTABLE	The file is not extractable.

Of these errors codes, **VFSERR_NOT_SUPPORTED** is the most useful to you. You should return this error code whenever your functions do not support some aspect of the API. For example, if you support forwards seeking in files but not random seeking, you should return this error when Opus asks you to seek randomly, and success otherwise.

Reading a Directory

Perhaps the most important function in the plugin API is the **VFS_ReadDirectory** function. This is called by Opus whenever the user browses to a new folder in your plugin's namespace.

BOOL VFS_ReadDirectory(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPVFSREADDIRDATA lpReadDirData)

The **hVFSData** parameter is your instance handle, returned from **VFS_Create**. This parameter is passed to almost all of the functions in the plugin. The **lpFuncData** parameter is a general purpose data pointer that currently is not used. Many functions accept this parameter and it may be used in future revisions of the API to provide additional information or callback parameters to your functions. For now you should ignore this parameter – it will usually be NULL.

The final parameter, **lpReadDirData**, is a pointer to a **VFSREADDIRDATA** structure. This structure provides information about the folder to be read, the type of event that has triggered the read, and provides a mechanism for you to pass back to Opus information about the folder contents.

The members of the **VFSREADDIRDATA** structure are as follows:

- **UINT cbSize**

  Directory Opus initializes this field to the size of the structure before passing it to your function. You must only use fields in the structure up to the size supplied by Directory Opus. This allows your plugin to be compatible with future versions of Opus that may support more fields in this structure.

- **HWND hwndParent**

  This field contains a window handle that can be considered the "parent" window for the purposes of displaying any status dialogs, error dialogs or any other user interaction required by your plugin.

- **LPTSTR lpszPath**

  This field is a pointer to a string containing the full path of the folder that Directory Opus wants to read the contents of.

- **vfsReadType vfsReadOp**

  This field contains an enumeration value that indicates the operation, or reason for the directory read. Several values of this enumeration have special meaning and must be observed – many others may be ignored unless your plugin has special behavior in these situations. Note that several codes do not actually indicate directory read functions, and so you should not return any directory information for these operations.

  - **VFSREAD_CHANGEDIR**

    This indicates that this is a "change directory" operation only. You should not actually return the contents of the directory – just use this as a hint to perform any directory change operation that your plugin may require.

- **VFSREAD_NORMAL**

  This indicates a normal directory read operation.

- **VFSREAD_REFRESH**

  The current folder is being refreshed. For all intents and purposes this is a normal directory read operation and you are free to treat it as such – a separate operation code is provided merely for information purposes.

- **VFSREAD_PARENT**

  The directory read is being performed because a **Go UP** function has been executed. Note that the **lpszPath** field already contains the parent folder – you do not need to calculate the parent path yourself. As with **VFSREAD_REFRESH**, you are free to treat this as a normal directory read operation unless your plugin requires special processing for a parent function.

- **VFSREAD_ROOT**

  In the same way as **VFSREAD_PARENT**, this code indicates the directory is being read because a **Go ROOT** function has been executed.

- **VFSREAD_BACK**

  In the same way as the previous two operation codes, this code indicates the directory is being read because of a **Go BACK** function.

- **VFSREAD_FORWARD**

  And again, this code indicates the directory is being read because of a **Go FORWARD** function.

- **VFSREAD_PRINTDIR**

  The directory is being read by the **Print Directory** function.

- **VFSREAD_FREEDIR / VFSREAD_FREEDIRCLOSE**

  These codes indicates a “free directory” operation only. You should not actually return the contents of the directory. If you have any cached data you may treat this code as a hint to free it. The VFSREAD_FREEDIR code indicates that the current directory is being freed but the namespace may be re-used to read another subsequent directory. The VFSREAD_FREEDIRCLOSE code indicates that the Lister (or file display) itself is being closed, and so the namespace will not be used again. If you do not have any processing you need to perform you are free to ignore this code altogether.

- **HANDLE hAbortEvent**

  This may be a handle to a Windows **event** object which is used to abort the reading of a directory if, for example, the user clicks the Abort button or closes the Lister while the directory read is taking place. If you wish to allow the user to abort your plugin (which is highly recommended), you should make sure that you check the event for *signaled* status at regular intervals. For example,

```
if ( hAbortEvent && WaitForSingleObject( hAbortEvent, 0 ) == WAIT_OBJECT_0 )
{
	// aborted
}
```

- **HANDLE hMemHeap**

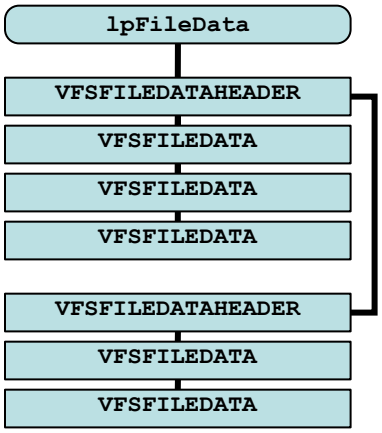
  This is a handle to a memory heap created with the Windows **HeapCreate** function. You will use this heap with the **HeapAlloc** function to allocate memory needed to return the directory contents.

- **LPVFSFILEDATAHEADER lpFileData**

  This field is initially set to 0 when your **VFS_ReadDirectory** function is called. For those operations that require you to return directory information (all except **VFSREAD_CHANGEDIR** and **VFSREAD_FREEDIR**), you use this field to return the actual directory contents.

  The **VFSFILEDATAHEADER** structure is the header directly preceding in memory an arbitrarily sized array of **VFSFILEDATA** structures. The **VFSFILEDATAHEADER** contains an optional pointer to the next header in a linked list – so you are free to return the directory contents as either one single memory allocation or as a linked list of multiple allocations, each containing an arbitrary number of entries.

  The basic structure in memory looks like this:



  The **lpFileData** member of the structure must be set to point to the address of the first allocated **VFSFILEDATAHEADER** structure. The **lpNext** member of the **VFSFILEDATAHEADER** structure is used to chain on to subsequent allocation blocks. Set the **lpNext** member of the final allocation to 0.

  Note that all memory must be allocated using the Windows **HeapAlloc** function and the memory heap passed in **hMemHeap**. Opus will free the memory you allocate using **HeapFree** and/or **HeapDestroy** when it no longer needs it.

  The members of the **VFSFILEDATAHEADER** structure are:

  - **UINT cbSize**

    This field must be set to the size of the **VFSFILEDATAHEADER** structure you are using, ie `sizeof(VFSFILEDATAHEADER)`. It is not the total allocation size. Directory Opus uses this value to find the first of the **VFSFILEDATA** structures, which follow the header structure in memory.

  - **LPVFSFILEDATAHEADER lpNext**

    Points to the next allocated chunk of file data, or set to 0 if there are no further chunks.

  - **int iNumItems**

    Set this to the number of records (each one of which is a **VFSFILEDATA** structure) included in this data chunk.

- **UINT cbFileDataSize**

  Set this to the size of the **VFSFILEDATA** structure you are using, ie `sizeof(VFSFILEDATA)`. Directory Opus uses this value to iterate through the supplied file records.

Immediately following the **VFSFILEDATAHEADER** structure in memory must be one or more **VFSFILEDATA** structures, the number of which is given by the **iNumItems** member in the header structure.

The members of the **VFSFILEDATA** structure are:

- **WIN32_FIND_DATA wfdData**

  This is a standard Windows API **WIN32_FIND_DATA** structure as used by the **FindFirst** / **FindNext** functions. You must initialize all appropriate fields in this structure to represent the directory item you are adding. See the MSDN docs for information on this structure.

- **DWORD dwFlags**

  This is a flags field - although no flags are currently used with the **VFS_ReadDirectory** function, and so this must be set to 0 for now.

- **LPTSTR lpszComment**

  This field points to an optional comment string for the item you are adding. If provided this will be displayed to the user in the Comment or Description fields in the Lister. The string this field points to must be allocated using **HeapAlloc** with the memory heap passed in **hMemHeap** (the same way the **VFSFILEDATAHEADER** etc structure was allocated.) Set to 0 if no comment is to be provided.

- **int iNumColumns**

  This member indicates the number of custom columns for which data is to be provided for this directory item. The **lpvfsColumnData** member points to an array of this number of **VFSFILEDATACOLUMN** structures. Set to 0 if you are not using custom columns or not providing any custom column data for this item.

- **LPVFSFILEDATACOLUMN lpvfsColumnData**

  This member points to an array of **VFSFILEDATACOLUMN** structures. This array must be allocated using **HeapAlloc** with the memory heap passed in **hMemHeap**. The size of the array is given by the **iNumColumns** field. The **VFSFILEDATACOLUMN** structure has two members – **iColumnId** specifies the custom column ID (the same value specified in the column definitions returned by the **VFS_GetCustomColumns** function), and **lpszValue**, which points to a null-terminated string containing the column data. If you are not using custom columns set this member to 0.

Once you have allocated and returned the directory contents, Opus will display the folder in the Lister. By default, icons displayed for your files and folders are the default system ones (that is, the standard folder icon for your folders, and the appropriate icon for your files based on their filename extensions).

If you wish to provide custom icons for your files and folders you should implement the **VFS_GetFileIcon** function. This function will be called by Opus for every file and folder in your plugin's namespace (but only when the icon is actually required, and potentially on a background thread.)

Reading and Writing File Data

As mentioned above, the VFS Plugin API defines a filesystem-like interface for reading and writing of files within the plugin's namespace. There are four key functions that need to be implemented in order to function as a useful plugin (or three for a read-only plugin) – **VFS_CreateFile**, **VFS_ReadFile**, **VFS_WriteFile** and **VFS_CloseFile**. An optional fifth function is **VFS_SeekFile** which is not required, but it is recommended you provide it if possible.

HANDLE VFS_CreateFile(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPTSTR lpszFile, DWORD dwMode,
DWORD dwFlagsAndAttr, DWORD dwFlags,
LPFILETIME lpFT)

The **hVFSData** parameter is your instance handle, returned from **VFS_Create**. This parameter is passed to almost all of the functions in the plugin. The **lpFuncData** parameter is a general purpose data pointer that currently is not used. Many functions accept this parameter and it may be used in future revisions of the API to provide additional information or callback parameters to your functions. For now you should ignore this parameter – it will usually be NULL.

lpszFile is the full pathname of the file to create or open. **dwMode** specifies the required operation – it will equal either **GENERIC_WRITE** or **GENERIC_READ** (but never both – Opus never requests read/write access to a file.) If **GENERIC_WRITE** is specified you should attempt to create the requested file; if **GENERIC_READ** is specified you should attempt to open an existing file.

dwFlagsAndAttr is equivalent to the same parameter in the Windows **CreateFile** function. It specifies the file attributes that the newly created file should be given (if **GENERIC_WRITE** was specified.) and also may contain flags controlling the behavior of the new file. The file attributes will usually be set to **FILE_ATTRIBUTE_NORMAL**. If your namespace does not support the concept of file attributes or does not support all the standard Windows attributes you are free to ignore this parameter. Currently the only flags that Opus uses at times are **FILE_FLAG_SEQUENTIAL_SCAN** and **FILE_FLAG_BACKUP_SEMANTICS** – you are free to ignore both of these flags unless it makes sense for your plugin to support them.

dwFlags is a flags field that combines the "creation disposition" parameter of the Windows **CreateFile** function with several Opus-specific flags. Again, you are free to ignore any of these flags if they do not make sense to your plugin. The "creation disposition" parameters are **CREATE_NEW**, **CREATE_ALWAYS**, **OPEN_EXISTING**, **OPEN_ALWAYS** and **TRUNCATE_EXISTING**. You can retrieve the creation disposition value using the special mask value **VFSCREATEF_MODEMASK**. If the creation disposition value is not specified (ie, the value is 0) you should assume "create new" for a write operation and "open existing" for a read operation.

The Opus-specific flags for this parameter are:

```
VFSCREATEF_RESUME          file is being opened to resume a transfer
VFSCREATEF_RECURSIVECOPY   a recursive copy operation is in progress
VFSCREATEF_THUMBNAIL       Opus wants to generate a thumbnail image
VFSCREATEF_IGNOREIFSLOW    if reading the file will take a long time you should
                           not open it and return failure instead
VFSCREATEF_BUFFERED        the file is being buffered
```

The value you return from **VFS_CreateFile** must be a data structure you allocate that can serve to identify the file in question. You must cast this value to a **HANDLE** and Opus treats it as a black-box – it does not know what the value points to, but merely saves the handle and passes it back to your other file functions.

The **VFS_ReadFile** and **VFS_WriteFile** functions are analogous to the **ReadFile** and **WriteFile** Windows API functions.

BOOL VFS_ReadFile(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
HANDLE hFile, LPVOID lpData, DWORD dwSize,
LPDWORD lpdwReadSize)

The parameters for this call are mostly self-explanatory. **hVFSData** is your instance handle. **lpFuncData** is not currently used. **hFile** is your file handle, returned by the call to **VFS_CreateFile**. **lpData** points to an area of memory that Opus wants you to return file data in. **dwSize** indicates the requested data amount to read, and **lpdwReadSize** points to a DWORD so you can tell Opus the actual amount of data you read.

Similar to the Windows **ReadFile** function, you do not need to read all requested data. You should not only return as much data as you can, up to and including **dwSize** bytes. Return the actual amount of data you did read in the value pointed to be **lpdwReadSize**. You should return **TRUE** from this function if you were successfully able to read any data at all. If an error occurred or the end-of-file has been reached, you should return **FALSE**. Do not return **FALSE** if you read any data – even if you were only able to read one byte, you should still return **TRUE**.

The **VFS_WriteFile** function is very similar to the **VFS_ReadFile** function, except it has an additional **BOOL fFlush** parameter. This will be set to **TRUE** if Opus wants you to flush any cached data to the file. If your plugin does not perform write caching you should ignore this parameter. If your plugin is read-only and does not support creation of or writing to files at all, you should not provide a **VFS_WriteFile** function, and you should make sure you set the **VFSCAPABILITY_READONLY** capabilities flag.

If your plugin supports random seeking within files you should implement the **VFS_SeekFile** function. Even if you only support forwards seeking you should implement this function and simply return an error for seek attempts that you can't handle.

Additional File Information

As well as the functions already described, there are a number of other functions in your plugin that Opus will call at times to obtain information about files. You should do your best to implement as many of these as possible – most should be quite straight-forward. These functions are all documented fully in the reference section of this SDK.

VFS_GetFileInformation is a function called by Opus to retrieve full information about a specific file or folder. The information is returned in exactly the same way as a directory listing is returned by **VFS_ReadDirectory**, except that only one **VFSFILEDATA** structure is returned.

The SDK also defines a trio of functions that provide an alternative method for reading directory contents. **VFS_FindFirstFile**, **VFS_FindNextFile** and **VFS_FindClose** are analogous to the similar functions in the Windows API. **VFS_FindFirstFile** is often called with an exact filename, in order to obtain file information about the specified file. Other times it is called with a wildcard pattern in order to retrieve information about all matching files in the specified folder. Opus uses this method quite often – for example, all recursive file operations are implemented using **VFS_FindFirstFile/VFS_FindNextFile**. Because the implementation of these functions would be very similar to that of the **VFS_ReadDirectory** function it is strongly suggested that your plugin provides them.

VFS_GetFileDescription is called by Opus to retrieve a description string for files in your namespace. This is displayed to the user in the **Description** column in Listers. Note the distinction between *descriptions* and *comments*. A comment is a string that the user is able to assign to a file or folder. Comments are supported in the plugin API via the functions **VFS_GetFileComment** and **VFS_SetFileComment**. Descriptions, on the other hand, are intended to be dynamically generated arbitrary descriptions of each file. For example, the description Opus displays for image files includes the file format and image resolution. The user does not set the description field themselves – the description is generated by the plugin based on the actual file itself.

VFS_GetFileSize is called to retrieve the size of a specified file, and **VFS_GetFileAttr** is called to retrieve the attributes of a specified file. There is also a **VFS_PropGet** function which can be called to retrieve certain information about files and folders.

File Manipulation

File manipulation returns to operations initiated by the user to make changes to files or folders within your plugin namespace. If your plugin is read-only, you do not need to implement these functions. Even if your plugin does support file creation and writing, you do not need to implement these functions if you are unable to support them. However, the more of these functions you can implement the more satisfying the end user's experience of your plugin will be.

VFS_CreateFile, which has already been described, is the function responsible for creating new files within your namespace. **VFS_DeleteFile** is analogous to the Windows API **DeleteFile** function and is used to delete files from your namespace.

VFS_CreateDirectory and **VFS_RemoveDirectory** are the equivalent functions relating to folders. If your plugin does not support sub-folders you do not need to provide these functions. Note that the Windows **RemoveDirectory** function is not able to delete a folder unless it is empty. Opus does not assume that all plugins have this same behavior, and so will attempt to delete directories using this function even if they are not empty. If your plugin can remove the directory as it is, it should do so – otherwise it should fail with the standard **ERROR_DIR_NOT_EMPTY** error code, and Opus will then delete the contents before trying to remove the directory again.

VFS_MoveFile is similar to the Windows **MoveFile** function, and is responsible for both moving a file (or folder) from one folder to another, and for renaming a file in-place.

VFS_SetFileAttr is used to change file attributes. Opus does not currently allow a plugin to specify its own attributes – that is, only the standard Windows attributes (read only, archive, hidden, etc) are supported. You should try and match these to your own plugin's requirements as best you can.

VFS_SetFileTime is used to change file timestamps. In the same way as the Windows **SetFileTime** function, this function is passed up to three pointers to **FILETIME** structures, for the creation time, last modified time, and last accessed time. If your plugin does not support all of these different time fields you are free to ignore them.

Batch Operations

Directory Opus 9 supports *batch-mode* operations for VFS plugins. If you implement the batch interface via the **VFS_BatchOperation** function, Opus will call this function for any copy or delete operation involving your plugin, rather than using the *filesystem-style* functions described above.

BOOL VFS_BatchOperation(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData, LPTSTR lpszPath, LPVFSBATCHDATA lpBatchData)

The **hVFSData** parameter is your namespace instance handle, and **lpFuncData** is not currently used. **lpszPath** specifies the current path in your namespace, and **lpBatchData** points to a **VFSBATCHDATA** structure that defines the operation parameters.

The fields of the **VFSBATCHDATA** structure are as follows:

- **UINT cbSize**

  This field specifies the size of the **VFSBATCHDATA** structure.

- **HWND hwndParent**

  This specifies the handle of the parent window, and can be used if you need to display any error dialogs or other windows while processing the batch operation.

- **HANDLE hAbortEvent**

  This may be a handle to a Windows **event** object which is used to indicate that a batch operation should be aborted. If you wish to allow the user to abort your plugin (which is highly recommended), you should make sure that you check the event for *signaled* status at regular intervals. For example,

```
if ( hAbortEvent && WaitForSingleObject( hAbortEvent, 0 ) == WAIT_OBJECT_0 )
{
	// aborted
}
```

- **UINT uiOperation**

  Specifies the batch operation type. Currently defined operations are:

  - **VFSBATCHOP_ADD**

    An **add** operation is analogous to a copy **to** your namespace – for example, adding files to an archive.

  - **VFSBATCHOP_EXTRACT**

    An **extract** operation is analogous to a copy **from** your namespace – eg, extracting files from an archive.

  - **VFSBATCHOP_DELETE**

    This is a **delete** operation – for example, removing files from an archive.

- **int iNumFiles**

  Specifies the number of selected files and folders (note: does not include the contents of any sub-folders)

- **LPTSTR pszFiles**

  Points to a double-null terminated list of file paths. For **VFSBATCHOP_EXTRACT** and **VFSBATCHOP_DELETE**, these will be files in your plugin's namespace. For **VFSBATCHOP_ADD**, these will be real files in the filesystem.

- **int* piResults**

  This points to an array of **int** values which lets you provide a result code for each of the files specified via **pszFiles**. You should set each integer value to 0 if the file or folder was processed successfully, or an appropriate error code on failure. The size of the **piResults** array is specified by the **iNumFiles** value.

- **LPTSTR pszDestPath**

  This specifies the destination path for a **VFSBATCHOP_EXTRACT** operation.

- **DWORD dwFlags**

  Flags that indicate how the batch operation should be performed. Currently defined flags are:

```
BATCHF_COPY_PRESERVE_ATTR        preserve file attributes on add/extract
BATCHF_COPY_PRESERVE_DATE        preserve file timestamps on add/extract
BATCHF_COPY_PRESERVE_COMMENTS    preserve file comments on add/extract
BATCHF_COPY_PRESERVE_SECURITY    preserve file security information
BATCHF_COPY_ASK_REPLACE          ask before overwriting existing files
BATCHF_COPY_ASK_REPLACE_RO       ask before overwriting read-only files
BATCHF_COPY_RENAME               ask user for new filenames (copy as)
BATCHF_COPY_DELETE_ORIGINAL      operation is a 'move' - delete original files
BATCHF_DELETE_ASK                ask before beginning delete
BATCHF_DELETE_ASK_FILES          ask for each file before deleting
BATCHF_DELETE_ASK_FOLDERS        ask for each folder before deleting
BATCHF_DELETE_UNPROTECT          automatically unprotect files to delete
BATCHF_DELETE_ALL                delete all without prompting for each
BATCHF_DELETE_QUIET              don't prompt to begin deleting
BATCHF_DELETE_FORCE              force deletion
BATCHF_DELETE_SECURE             secure delete if possible
BATCHF_FILTER                    a filter is in force
BATCHF_FILTER_FOLDERS            a filter is in force for folders
BATCHF_PROGRESS_SUBFOLDERS       progress bar knows about sub-folder contents
```

  Most of these flags correspond to the options in the Opus Preferences (Copying/Deleting) and may not be relevant to your plugin. The 'ask' flags should be respected if possible – you can display confirmation dialogs using the Plugin Support API, which also contains functions to make respecting the 'filter' flags easy as well.

- **LPVOID lpFuncData**

  This provides a parameter that is used in several calls to the Plugin Support API for updating progress bars, filtering files, etc.

When Opus calls your **VFS_BatchOperation** function, you should examine the **VFSBATCHDATA** structure to determine if this is an operation you can process in batch mode. If it isn't (for example, you may support batch extraction but not deletion), you should return **VFSBATCHRES_DODEFAULT**. This indicates to Opus that batch mode is not available and Opus will then proceed to handle the operation in the usual way (via calls to **VFS_CreateFile**, **VFS_ReadFile**, etc.)

If, however, you can process the requested operation, you should then proceed to do so. Basically this is a matter of performing the requested actions, and returning result codes for each file. However, it is also the responsibility of the plugin to emulate Opus as much as possible while processing a batch operation. This may include displaying confirmation dialogs or dialogs asking for new filenames, displaying error messages, applying filters to files and folders and updating the progress bar. It is not required that you perform all these steps, however the user will enjoy a more seamless experience if you do.

The Plugin Support API (documented separately) provides several functions which can make these steps much easier for you. The functions specifically involved with batch mode operations that you may wish to use are:

ShowRequestDlg	generic message dialog helper function
ShowFunctionNewNameDlg	displays a dialog allowing the user to specify a new filename
GetWildNewName	get a new filename from user-entered wildcard pattern
ShowFunctionErrorDlg	display an error message
ShowFunctionReplaceDlg	display 'replace existing file?' confirmation dialog
ShowFunctionInitialDeleteDlg	display initial confirmation dialog when deleting files
ShowFunctionDeleteDlg	display confirmation dialog when deleting a file or folder
FilterFunctionFile	apply filters to files or folders
AddFunctionFileChange	notify Opus of a file change in your plugin's namespace
UpdateFunctionProgressBar	update the progress bar for a batch mode operation
GetFunctionWindow	retrieve window handle for a batch mode operation

For example, if the **BATCHF_FILTER** flag was set in the **VFSBATCHDATA** structure, you would call the **FilterFunctionFile** function to see if a file was to be filtered out.

See the Plugin Support API documentation for full information about these helper functions.

Your **VFS_BatchOperation** function should return a value indicating the result of the batch operation. See the reference section for a full description of the return codes.

File and Folder Properties

The **VFS_PropGet** function is used to obtain plugin behavior information. The difference between this and other functions is that the **VFS_PropGet** function can be passed the name of a file or folder. This lets your plugin modify its behavior for different items. This function is also used to obtain meta-information about specific files and folders in your plugin namespace.

BOOL VFS_PropGet(HANDLE hVFSData, vfsProperty propId, LPVOID lpPropData,
LPVOID lpData1, LPVOID lpData2, LPVOID lpData3)

The **hVFSData** parameter is your plugin instance data, and **propId** specifies the particular property that Opus is querying for. **lpPropData, lpData1**, **lpData2** and **lpData3** are general purpose arguments whose meaning changes depending on the property in question.

If your plugin supports the property Opus is querying for, you should return any necessary data using the supplied arguments, and return **TRUE**. If you do not support the property you should return **FALSE**. If your plugin does not support any properties at all you do not need to provide this function.

See the reference section for a full description of the various property values.

Special File Actions

Three functions are provided to support “special” actions on files.

VFS_ContextVerb is called whenever the user attempts to “invoke” a file in your namespace. This may be by double-clicking on it, right-clicking and choosing Open from the context menu, or running the internal **FileType** command on it. The most common requirement of the **VFS_ContextVerb** function is to be able to open or launch files from your namespace.

int VFS_ContextVerb(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPVFSCONTEXTVERBDATA lpVerbData)

hVFSData is your namespace instance handle, and **lpFuncData** is currently unused. **lpVerbData** points to a **VFSCONTEXTVERBDATA** structure which contains information about the file and which action to perform on it. This structure is defined as follows:

- **UINT cbSize**

  This is set to the size of the structure and is used to preserve compatibility with older versions of the API. You should only access any values up to and including the size of the structure.

- **HWND hwndParent**

  This is the handle to the parent window that is launching this operation. You should use this handle if you need to display any dialogs or status indicators, etc.

- **LPTSTR lpszVerb**

  This points to a verb string that indicates the action you are to perform. If this value is NULL it indicates the user double-clicked on the item and you are to perform the default action.

- **LPTSTR lpszPath**

  This string provides the full pathname of the item the action is to be performed on.

- **LPTSTR lpszNewPath**

  This points to a buffer into which you can place a new filename path. If you return **VFSCVRES_CHANGE** from the **VFS_ContextVerb** function, Opus will automatically re-invoke the action on the new path you provided. For example, this lets your plugin extract the specified file to a temporary local file and then have the double-click action performed on the temporary file.

- **int cchNewPathMax**

  This value specifies the size in characters of the buffer pointed to by **lpszNewPath**.

- **UINT uMsg**

  This value specifies the message code that triggered this action. This will normally be either 0 or **WM_LBUTTONDBLCLK.**

- **WPARAM fwKeys**

  This provides any qualifiers that were in effect when the action was triggered. For example, **MK_SHIFT**, **MK_CONTROL**, **MK_ALT**, **MK_LBUTTON**, etc.

- **DWORD dwFlags**

  This value provides additional flags relating to the operation. Currently the only defined flag is **DOPUSCVF_ISDIR** which indicates that the item in question is a directory rather than a file.

- **int iRotateAngle**

  This value indicates the currently displayed rotation angle of the item's thumbnail. When the user views files in thumbnails mode they are able to rotate the display of the thumbnail, and this value lets you determine the current rotation angle if it is applicable to your plugin.

There are a number of different return codes defined for **VFS_ContextVerb**. These are:

- **VFSCVRES_FAIL**

  This code indicates that the operation has failed.

- **VFSCVRES_HANDLED**

  This return code indicates that you have successfully handled the operation.

- **VFSCVRES_DEFAULT**

  This code indicates that you want Opus to handle the operation itself using the default action for this file type. This return code is only applicable if the path provided in **lpszPath** refers to a real, physical file. Therefore you should only use this code if you have previously returned **VFSCVRES_CHANGE** and specified a real filesystem path in **lpszNewPath**.

- **VFSCVRES_EXTRACT**

  Return this code if you want Opus to extract the specified file from your plugin using the **VFS_ExtractFile** function. Opus will extract the file to a temporary local file and then perform the default action upon it.

- **VFSCVRES_CHANGE**

  Return this code if you have placed a new file path string in the buffer pointed to by **lpszNewPath**. Opus will then invoke the **ContextVerb** function again on that new path string. Note that this may or may not involve your **VFS_ContextVerb** function being called again with the new string, depending on whether the new path references your namespace or another namespace.

- **VFSCVRES_CHANGEDIR**

  Return this code if you want Opus to read a new folder as the result of the operation. You should place the path of the folder to read in the **lpszNewPath** field. Opus will execute the **Go** command on this path string.

VFS_ExtractFiles is a function that Opus calls from time to time to extract one or more files from your plugin's namespace to a local file. For example, if you return **VFSCVRES_EXTRACT** from the **VFS_ContextVerb** function, Opus will call this function to extract the file to a local temporary file. If you do not provide this function then Opus will attempt to extract the file itself using **VFS_CreateFile** and **VFS_ReadFile**. If you are able to extract files in a more efficient manner than this you should implement this function.

BOOL VFS_ExtractFiles(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPVFSEXTRACTFILESDATA lpExtractData)

hVFSData is your namespace instance handle, and **lpFuncData** is currently unused. **lpExtractData** points to a **VFSEXTRACTFILESDATA** structure which contains information about the file or files to be extracted. This structure is defined as follows:

- **UINT cbSize**

  This field specifies the size of the structure and is used to maintain compatibility with older versions of the API. You should only access any values up to and including the size of the structure.

- **HWND hwndParent**

  This is the handle to the parent window that is launching this operation. You should use this handle if you need to display any dialogs or other user interaction.

- **LPTSTR lpszDestPath**

  This field points to a string containing the desired destination path for the extracted file or files.

- **LPTSTR lpszFiles**

  This field points to a double null-terminated list of strings, each one specifying a file to extract from the plugin's namespace.

- **DWORD dwFlags**

  This is a flags field and is currently unused.

When this function is called in your plugin you should attempt to extract the specified file or files to the specified destination folder. If the extraction operation will take a significant time you may wish to consider displaying a progress dialog and giving the user the option of aborting the operation.

If the file extraction was successful you should return **TRUE**, otherwise you should set an appropriate error code and return **FALSE**.

The **VFS_Properties** function (not to be confused with **VFS_PropGet**) is called when the user wants to view a Properties dialog for one or more files in your namespace. If you specified the **VFSCAPABILITY_COMBINEDPROPERTIES** capability flag your **VFS_Properties** function may be called with multiple filenames and you should display an appropriate dialog that combines the properties of these files as best you can. If you did not specify the combined properties capability flag then **VFS_Properties** will only ever be called with a single filename.

HWND VFS_Properties(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
HWND hwndParent, LPTSTR lpszFiles)

hVFSData is your namespace instance handle, and **lpFuncData** is currently unused. **hwndParent** is the handle to the parent window – any dialog you display should use this window as a parent and should position itself relative to this window. **lpszFiles** is a double null-terminated list of strings containing the full pathnames of the files to show Properties for. If you do not support combined properties you can treat this as a simple string.

Your **VFS_Properties** function can either be *modal* or *modeless*. Modal means that you will run a message loop to display the dialog, and destroy the dialog when the user closes it. If you are implementing this as a modal function you should simply return **TRUE** or **FALSE** from this function (cast to **HWND**). A modeless function, on the other hand, returns the handle of a dialog window. Opus then runs a standard dialog message loop, and will destroy the dialog itself when the user clicks the close button or the Ok or Cancel button (that is, in response to a **WM_CLOSE** or appropriate **WM_COMMAND** message.)

The dialog you display should be as similar in appearance to the standard Windows file Properties dialog as possible, to avoid confusing the user and to create a more seamless environment for them. If your plugin does not support the displaying of a Properties dialog at all then simply do not implement this function.

Context and Drop Menus

When the user clicks the right button on a file in your namespace, or drags a file with the right button into your namespace, Opus calls functions in your plugin to build a context menu for display to the user. The two functions responsible for this are **VFS_GetContextMenu** and **VFS_GetDropMenu** and they are very similar.

BOOL VFS_GetContextMenu(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPTSTR lpszFiles,
LPVFSCONTEXTMENUDATA lpMenuData)

hVFSData is your namespace instance handle, and **lpFuncData** is currently unused. **lpszFiles** is a double null-terminated list of files, providing the full pathnames of the files the user right-clicked on. **lpMenuData** points to a **VFSCONTEXTMENUDATA** structure.

By comparison, the **VFS_GetDropMenu** function is almost identical:

BOOL VFS_GetDropMenu(HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPTSTR lpszFiles,
LPVFSCONTEXTMENUDATA lpMenuData,
DWORD dwEffects)

The only addition to this function definition is the **dwEffects** parameter which is a mask of the OLE drag-and-drop "effects" currently in use when the drop occurred (that is, **DROPEFFECT_COPY** indicates the control key was held down**, DROPEFFECT_MOVE** indicates the shift key and **DROPEFFECT_LINK** indicates the alt key. **DROPEFFECT_NONE** indicates no qualifier keys were held down when the drop occurred.)

The **VFSCONTEXTMENUDATA** structure is defined as follows:

- **UINT cbSize**

  This will be set to the size of the **VFSCONTEXTMENUDATA** structure that Opus is using. This is to maintain compatibility with older versions of the API. You should only access fields up to and including the size of this structure.

- **BOOL fAllowContextMenu**

  Set this to **FALSE** to disallow this context menu, or **TRUE** to allow it.

- **BOOL fDefaultContextMenu**

  Set this to **TRUE** to have Opus include default context menu items in the context menu shown to the user. Any items you provide will be displayed either above or below the default items. If you set this to **FALSE** then only the items you provide will be displayed.

- **BOOL fCustomItemsBelow**

  Set this to **TRUE** to have your custom items displayed below the default ones, or **FALSE** to have your items displayed above them. Has no effect if **fDefaultContextMenu** is set to **FALSE**.

- **LPVFSCONTEXTMENUITEM lpCustomItems**

  Set this value to point to an array of **VFSCONTEXTMENUITEM** structures, one for each custom item you wish to add to the context menu. The array can either by statically defined in your plugin, or you can allocate it at run-time using **LocalAlloc**. If you choose to allocate the menu array at run-time you must set the **fFreeCustomItems** flag to enable Opus to free the array via **LocalFree** once it has displayed the context menu.

- **int iNumCustomItems**

  Set this field to the number of **VFSCONTEXTMENUITEM** structures in the array provided by **lpCustomItems**.

- **BOOL fFreeCustomItems**

  Set this value to **TRUE** to have Opus free your **lpCustomItems** array via **LocalFree** once the context menu has been displayed. If you set this to **FALSE** Opus will not free your array.

The **lpCustomItems** member points to an array of **VFSCONTEXTMENUITEM** structures that you supply. This structure is defined as follows:

- **UINT cbSize**

  You must set this field to the size of the **VFSCONTEXTMENUITEM** structure you are using (that is, `sizeof(VFSCONTEXTMENUITEM)`.) Opus uses this value to iterate through the array you provide.

- **DWORD dwFlags**

  This is a flags field that controls the behavior and appearance of the menu item. The currently defined flags are:

```
VFSCMF_CHECKED       displays a checkmark next to the menu item
VFSCMF_RADIOCHECK    the checkmark is displayed as a radio button
VFSCMF_DISABLED      the item is disabled and cannot be selected
VFSCMF_SEPARATOR     the item is a separator bar
VFSCMF_BEGINSUBMENU  the item begins a submenu
VFSCMF_ENDSUBMENU    the item is the last in a submenu
```

- **LPTSTR lpszLabel**

  This points to a string that is the label of the context menu item. To assign a hotkey to a menu item prefix the letter with an ampersand (&) character.

- **LPTSTR lpszCommand**

  This is the command string that will be executed if the user chooses this menu item.

Note that if you set the **fFreeCustomItems** flag to **TRUE** to have Opus free the menu item array, only the memory pointed to by **lpCustomItems** is freed - the individual strings in each **VFSCONTEXTMENUITEM structure** are not. Therefore the strings must either be statically defined in your plugin, or must be part of the memory chunk that you allocated for

lpCustomItems. Otherwise you may leak memory when the user displays your context menu.

The command string pointed to by **lpszCommand** defines the command that is executed when the user chooses that menu item. This can be any raw Opus command (for example, **“Go OPENINDUAL”**) or it can be a user-defined command verb. To specify your own commands, select a verb keyword and prefix it with a $ character. This verb (minus the leading $ character) will be passed to your **VFS_ContextVerb** function when the user selects this command from the context menu. For example, a context menu item whose **lpszCommand** string was set to **“$viewfile”** would trigger a call to **VFS_ContextVerb** with the **lpszVerb** parameter set the **“viewfile”**.

If you do not want to add your own context menu items and simply want Opus to display default context menus, you do not need to provide the context menu functions.

Miscellaneous Functions

The **VFS_GetPathParentRoot** function can be provided by your plugin if you want to do custom path processing whenever Opus needs to calculate the parent or root folder of a folder in your plugin's namespace. Opus calls this function if you set the **VFSCAPABILITY_LETMEDOPARENTS** capabilities flag.

BOOL VFS_GetPathParentRoot (HANDLE hVFSData, LPTSTR lpszPath, BOOL fRoot,
LPTSTR lpszNewPath, int cchNewPathMax)

hVFSData is your plugin's namespace instance data. **lpszPath** points to the current path within your plugin's namespace. If **fRoot** is set to **TRUE** then Opus wants you to calculate the root folder of this path – if set to **FALSE**, Opus wants the parent folder. **lpszNewPath** points to a buffer into which you copy the new path string – **cchNewPathMax** specifies the size of this buffer in characters.

If it is possible to calculate the desired path you should place the new path in the supplied buffer and return **TRUE**. If it is not possible to calculate the desired path, you should return **FALSE**. For example, if the supplied path indicates the root of your namespace then it is not possible to go up any further, and you should return **FALSE**.

If you do not provide this function then Opus applies standard path parsing rules to calculate the desired path.

The **VFS_GetFreeDiskSpace** function should be provided if your plugin is able to supply meaningful information about free and used disk space. If this is not meaningful for your plugin do not provide this function.

BOOL VFS_GetFreeDiskSpace (HANDLE hVFSData, LPVFSFUNCDATA lpFuncData,
LPTSTR lpszPath,
unsigned __int64* piFreeBytesAvailable,
unsigned __int64* piTotalBytes,
unsigned __int64* piTotalFreeBytes)

hVFSData is your namespace instance data, and **lpFuncData** is currently unused. **lpszPath** points to a string containing the full path for which you should return disk space information. The three remaining parameters all point to unsigned 64-bit values into which you should place the appropriate information. Note that Opus may not necessarily request all three values at the same time, so you should check if the pointers provided are non-NULL before calculating and returning the values. **piFreeBytesAvailable** is the total number of free bytes available for the current user, **piTotalBytes** is the total number of bytes on the disk (both used and free) and **piTotalFreeBytes** is the total number of free bytes on the disk. **piFreeBytesAvailable** and **piTotalFreeBytes** are often the same value.

For prefix-style plugins, the **VFS_GetPathDisplayName** function can be used to convert the path formats used by your plugin to a pretty path for display to the user.

BOOL VFS_GetPathDisplayName (HANDLE hVFSData, LPTSTR lpszPath, LPTSTR lpszDisplayName, int cbDisplayNameMax)

hVFSData is your plugin's namespace instance data. **lpszPath** points to the current path within your plugin's namespace. **lpszDisplayName** and **cbDisplayNameMax** provide a buffer into which you can copy the "pretty" form of the path.

For example, you may wish to encode user-name or password information into your URL-style paths, but obviously would not want this information to be displayed to the user. The Opus internal FTP support uses this method. Internally, FTP paths are stored as ***ftp://user:password@ftpsite.com/path/to/folder***, but the path displayed to the user has the user and password information removed – **ftp://ftpsite.com/path/to/folder**.

If you return **FALSE** from this function (or do not provide the function at all) then Opus will display paths as-is to the user.

Configure and About

If the **VFSF_CANCONFIGURE** plugin flag is set, Directory Opus will enable a **Configure** function for your plugin. This is accessed via the Plugins page in Preferences. When the user selects this function, your plugin's **VFS_Configure** function is called.

The **VFS_Configure** function should create and display a dialog that allows the user to configure your plugin. The dialog should be modeless – the **VFS_Configure** function must return the window handle to Directory Opus.

HWND VFS_Configure(HWND hwndParent, HWND hwndNotify,
DWORD dwNotifyData)

hwndParent is the parent window handle for your configuration dialog. **hwndNotify** and **dwNotifyData** are used to notify Directory Opus that your plugin needs to be reinitialized because of configuration changes.

If the **VFSF_CANSHOWABOUT** plugin flag is set, Directory Opus will call your **VFS_About** function when the user selects the **About** function for your plugin.

HWND VFS_About(HWND hwndParent)

The **VFS_About** function should create a modeless dialog using **hwndParent** as the parent window handle, and return the handle to the dialog window. If you do not provide this function then Opus will display a generic About window for your plugin.

Exported Functions Reference

VFS_About

The **VFS_About** function displays an About dialog for your plugin.

```
HWND VFS_About(
      HWND hwndParent
);
```

Parameters

hwndParent
[in] The handle to the parent window

Return Values

The return value is the window handle of the About dialog created by the function.

Remarks

This function is called when the user chooses to display About information for your plugin. It is only called if the **VFSF_CANSHOWABOUT** plugin flag is specified in the call to **VFS_Identify**.

VFS_BatchOperation

The **VFS_BatchOperation** function is called to initiate a batch-mode file operation involving your plugin's namespace.

```
UINT VFS_BatchOperation (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPVFSBATCHDATA lpBatchData
);
```

Parameters

hVFSData
[in] The handle to the existing namespace instance

lpFuncData
[in] unused

lpszPath
[in] Null-terminated string representing the path in your plugin's namespace involved in the batch operation

lpBatchData
[in] Pointer to a **VFSBATCHDATA** structure that defines the operation

Return Values

The return value indicates the result of the batch operation. Currently defined return codes are:

VFSBATCHRES_DODEFAULT
The operation cannot be performed in batch mode.

VFSBATCHRES_SKIP
Skip the first file in the **pszFiles** array and call again for the next file

VFSBATCHRES_HANDLED
File has been handled by the batch operation

VFSBATCHRES_ABORT
Abort the function

VFSBATCHRES_COMPLETE
Operation has been completed for all files

VFSBATCHRES_CALLFOREACH
Set this flag in conjunction with the **SKIP or HANDLED** return codes to force the **VFS_BatchOperation** function to be called for each file

Remarks

As well as returning one of the above codes to indicate the result of a batch operation, you should also fill in the **piResults** array (supplied in the **VFSBATCHDATA** structure) with a result code for each of the specified files – either 0 for success or an error code on failure.

VFS_Clone

The **VFS_Clone** function is called to create a clone of an existing namespace instance.

```
HANDLE VFS_Clone (
      HANDLE hVFSData
);
```

Parameters

hVFSData
[in] The handle to the existing namespace instance

Return Values

The return value is the handle to a new namespace instance of the same type as the passed-in handle.

Remarks

If this function is not provided and Opus needs to clone an existing namespace it simply falls back to creating a new one via **VFS_Create**.

VFS_CloseFile

The **VFS_CloseFile** function is called to close a file handle previously opened by **VFS_CreateFile**.

```
void VFS_CloseFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      HANDLE hFile
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

hFile
[in] the handle to the file that is to be closed

Return Values

There is no return value.

Remarks

You should take any steps necessary to close the file (including flushing any write buffers if appropriate) and free any memory involved in the original allocation of the file handle.

VFS_Configure

The **VFS_Configure** function displays a configuration dialog for your plugin.

```
HWND VFS_Configure (
      HWND hwndParent,
      HWND hwndNotify,
      DWORD dwNotifyData
);
```

Parameters

hwndParent
[in] The handle to the parent window

hwndNotify
[in] A window handle you can use to notify Opus that your plugin needs to be reinitialized

dwNotifyData
[in] Message data for the notification message

Return Values

The return value is the window handle of the configuration dialog created by the function, or a boolean value cast as a **HWND** if you wish to run a modal dialog.

Remarks

This function is called when the user chooses to configure your plugin. It is only called if the **VFSF_CANCONFIGURE** plugin flag is specified in the call to **VFS_Identify**. You can either return a valid **HWND**, in which case Opus will run a standard dialog message loop, or a boolean value, if you wish to run the message loop yourself.

If the user makes changes that require your plugin to be reinitialized (eg, adds new file extensions that you support), you can use the **hwndNotify** and **dwNotifyData** parameters to notify Opus of this. You need to post the **DVFSPLUGINMSG_REINITIALIZE** message to the specified window handle, and pass **dwNotifyData** as the **lParam** value of the message.

VFS_ContextVerb

The **VFS_ContextVerb** function is called to perform an action upon a file in your plugin's namespace.

```
int VFS_ContextVerb (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPVFSCONTEXTVERBDATA lpVerbData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpVerbData
[in/out] Address of a **VFSCONTEXTVERBDATA** structure

Return Values

The following return values are valid:

VFSCVRES_FAIL
The action has failed

VFSCVRES_HANDLED
The action was performed successfully

VFSCVRES_DEFAULT
Opus should perform the default action

VFSCVRES_EXTRACT
Opus should extract the file and call **VFS_ContextVerb** again

VFSCVRES_CHANGE
Opus should call **VFS_ContextVerb** with the path in **lpszNewPath**

VFSCVRES_CHANGEDIR
Opus should read the folder specified by **lpszNewPath**

Remarks

The **lpszVerb** field will be set to NULL for "default action" – meaning the user has double-clicked on the item. If you add custom commands to the context menu via the **VFS_GetContextMenu** function then those commands will be sent to your **VFS_ContextVerb** function.

If you return **VFSCVRES_CHANGE** or **VFSCVRES_CHANGEDIR** you must provide a new path string in the **lpszNewPath** field. You must only copy up to **cchNewPathMax** characters into this buffer.

VFS_Create

The **VFS_Create** function is called to create a unique instance of your plugin's namespace.

```
HANDLE VFS_Create (
      LPGUID pGUID,
      HWND hwndMsgWindow
);
```

Parameters

pGUID
[in] A pointer to a GUID representing the type of namespace to create

hwndMsgWindow
[in] A window handle with which you can communicate with Directory Opus

Return Values

The function must return a **HANDLE** value which points to a private data structure that provides everything necessary for your plugin to identify and keep track of this namespace instance.

Remarks

If you set the **VFSF_MULTIPLEFORMATS** plugin flag then *pGUID* will be a GUID retrieved from the **VFS_QueryPath** function. If the multiple format flag is not set the GUID will be that supplied by the **VFS_Identify** function.

Currently there are no defined messages that you can send to the window handle supplied in *hwndMsgWindow* although it is likely this will be used in the future.

VFS_CreateDirectory

The **VFS_CreateDirectory** function is called to create a new sub-directory in your plugin's namespace.

```
BOOL VFS_CreateDirectory (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      DWORD dwFlags
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] Null-terminated string representing the full path of the directory to create

dwFlags
[in] Flags relating to the create directory operation.

```
VFSCREATEDIRF_COPY
   The folder is being created as part of a recursive copy operation

VFSCREATEDIRF_MULTIPLE
   The user wants to create multiple folders at once
```

Return Values

The function must return **TRUE** if the folder was successfully created, or **FALSE** on failure. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError**.

Remarks

If you set the **VFSCAPABILITY_MULTICREATEDIR** capabilities flag, Opus will allow the user to create multiple folders at once. The **VFSCREATEDIRF_MULTIPLE** flag in the *dwFlags* parameter indicates this, and you should treat any commas in the supplied path as separator characters.

VFS_CreateFile

The **VFS_CreateFile** function is called to open or create a file in your plugin's namespace.

```
HANDLE VFS_CreateFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszFile,
      DWORD dwMode,
      DWORD dwFlagsAndAttr,
      DWORD dwFlags,
      LPFILETIME lpFT
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszFile
[in] Null-terminated string representing the full pathname of the file to open or create

dwMode
[in] Indicates the type of access desired for the new file handle.

```
GENERIC_READ
   An existing file is being opened for reading

GENERIC_WRITE
   An existing file is being opened or a new file is being created for
   writing
```

dwFlagsAndAttr
[in] Specifies the attributes to assign to a newly created file, and optional flags to control the behavior of the new file. This value is equivalent to the similar parameter in the Windows **CreateFile** function – see the Windows API SDK for information on these flags.

dwFlags
[in] Combines the Windows API "creation disposition" parameter with several Opus-specific flags.

```
VFSCREATEF_RESUME
   File is being opened to resume a transfer

VFSCREATEF_RECURSIVECOPY
   A recursive copy operation is in progress

VFSCREATEF_THUMBNAIL
   Opus is opening the file in order to generate a thumbnail

VFSCREATEF_IGNOREIFSLOW
   If reading the file will take a long time you should not open it and
   return failure instead

VFSCREATEF_BUFFERED
   The file is being buffered
```

lpFT
[in] Unless set to NULL, this points to a **FILETIME** structure specifying the creation timestamp that is to be used for the newly created file.

Return Values

The function must return a **HANDLE** value that represents the file if the file was successfully opened or created. Directory Opus will pass this handle to the other file read/write API functions.

Remarks

You will only ever be called with **GENERIC_WRITE** if you do not specify the **VFSCAPABILITY_READONLY** capabilities flag. Directory Opus never opens a file for both read and write access – the *dwMode* parameter will always equal **GENERIC_READ** or **GENERIC_WRITE**.

As well as the flags listed above for *dwFlags this* parameter also contains the Windows **CreateFile** "creation disposition" parameter – **CREATE_NEW**, **CREATE_ALWAYS**, etc. You can retrieve the creation disposition value using the special mask value **VFSCREATEF_MODEMASK**. If creation disposition value is not specified you should assume "create new" for a write operation and "open existing" for a read operation.

In general, if your plugin does not support some of the concepts or features of this function you are free to ignore them – the only really important thing is that you return a **HANDLE** that can be passed to the **VFS_ReadFile**, **VFS_WriteFile**, **VFS_SeekFile** and **VFS_CloseFile** functions.

VFS_DeleteFile

The **VFS_DeleteFile** function is called to delete a file in your plugin's namespace.

```
BOOL VFS_DeleteFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      DWORD dwFlags,
      int iSecurePasses
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] Null-terminated string representing the full path of the file to delete

dwFlags
[in] Flags relating to the delete operation.

```
VFSDELETEF_FORCE
   Force delete even if delete protected

VFSDELETEF_RECYCLE
   Delete to recycle bin if possible

VFSDELETEF_REPLACE
   Replacing an existing file during a copy operation

VFSDELETEF_COPYFAILED
   Being called to cleanup after a failed copy operation

VFSDELETEF_SOURCERESUME
   The source of the copy supports resume
```

iSecurePasses
[in] Number of secure delete passes to perform or 0 for normal delete

Return Values

The function must return **TRUE** if the file was successfully deleted, or **FALSE** on failure. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError**.

Remarks

The only flag it is important to support, if relevant to your plugin, is **VFSDELETEF_FORCE**. If your plugin supports resume of an interrupted copy and the **VFSDELETEF_COPYFAILED** flag is set you may elect to not delete the file.

VFS_Destroy

The **VFS_Destroy** function is called to destroy a namespace instance.

```
Void VFS_Destroy (
      HANDLE hVFSData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance to destroy

Return Values

There is no return value.

Remarks

You should free any allocated resources and delete the data object as required.

VFS_ExtractFiles

The **VFS_ExtractFiles** function is called to extract files from your plugin's namespace to a local disk file.

```
BOOL VFS_ExtractFiles (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPVFSEXTRACTFILESDATA lpExtractData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpExtractData
[in] Address of a **VFSEXTRACTFILESDATA** structure

Return Values

If the extraction was successful you should return **TRUE**, otherwise you should set an appropriate error code and return **FALSE**.

Remarks

The **lpszDestPath** field of the **VFSEXTRACTFILESDATA** structure contains the destination path for the extracted files. The **lpszFiles** is a double null-terminated list of strings, each one specifying a file to extract from your plugin's namespace.

If you do not provide this function then Opus will simulate it with a combination of **VFS_CreateFile** / **VFS_ReadFile** / **VFS_CloseFile** – so unless you are able to extract files more efficiently than this you may feel free to not implement it.

VFS_FindClose

The **VFS_FindClose** function is called to close a find handle returned by **VFS_FindFirstFile**.

```
void VFS_FindClose (
      HANDLE hVFSData,
      HANDLE hFind
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

hFind
[in] The find handle to close

Return Values

There is no return value.

Remarks

You should release all allocated resources and free the handle passed in *hFind*.

VFS_FindFirstFile

The **VFS_FindFirstFile** function is called to obtain information about a single file or folder, or about all items in a directory that match a wildcard pattern.

```
HANDLE VFS_FindFirstFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPWIN32_FIND_DATA lpwfdData,
      HANDLE hAbortEvent
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] The full pathname of the item to retrieve information for, or a path plus wildcard pattern to examine all matching items.

lpwfdData
[out] Pointer to a **WIN32_FIND_DATA** into which you place information about the first matching item

hAbortEvent
[in] May be NULL, otherwise provides a Windows event that you can use to detect when the user wants to abort a lengthy operation

Return Values

If the operation succeeds, you must allocate and return a data structure that will passed to the **VFS_FindNextFile** function to continue the enumeration of all matching items. If the operation fails you should return NULL.

Remarks

You should check the final path component of *lpszPath* to see if it is an actual filename, or a wildcard pattern. Currently Opus does not use wildcard patterns other than **"*"** – that is, either Opus wants just a single file, or it wants every file in the directory.

You should place as much information as possible about the first (or only) matching file into *lpwfdData* and return a data structure cast to a **HANDLE** that Opus can use to continue the enumeration.

If Opus only asks for information about a single file there is no need to allocate a data structure – in this case you could just return **TRUE** cast to a **HANDLE**, as long as you check for this value in **VFS_FindNextFile** and **VFS_FindClose**. If Opus supplies a wildcard pattern then the structure pointed to by the handle you return must provide enough information for your **VFS_FindNextFile** function to continue the enumeration of all items in the directory.

If *hAbortEvent* is non-NULL it is a windows event handle that you can use to check if the user wants to abort the enumeration. You should store this handle in your data structure so you can check it in your **VFS_FindNextFile** function. You can check for the abort signal with the following code:

```
if ( hAbortEvent && WaitForSingleObject( hAbortEvent, 0 ) == WAIT_OBJECT_0 )
{
      // aborted
}
```

VFS_FindNextFile

The **VFS_FindNextFile** function is called to continue the enumeration of the contents of a folder begun with **VFS_FindFirstFile**.

```
BOOL VFS_FindNextFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      HANDLE hFind,
      LPWIN32_FIND_DATA lpwfdData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

hFind
[in] The find handle returned by **VFS_FindFirstFile**

lpwfdData
[out] Pointer to a **WIN32_FIND_DATA** structure into which you place information about the next matching item

Return Values

If the operation succeeds, you must copy the file information to *lpwfdData* and return **TRUE**. If the operation fails you must return **FALSE**. You should set the **ERROR_NO_MORE_FILES** error code and return **FALSE** when there are no more items to enumerate.

Remarks

Currently the only wildcard pattern Opus uses in calls to this function is a **"*"** meaning return all items. This means you do not need to actually check that the filename you are returning matches the pattern – simply iterate through all files in the originally specified folder.

VFS_GetCapabilities

The **VFS_GetCapabilities** function is called to retrieve the capabilities flags of the namespace.

```
DWORD VFS_GetCapabilities (
      HANDLE hVFSData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

Return Values

Return a bitmask representing the VFS capabilities flags relevant to your namespace. See the **Capabilities Flags Reference** section for a full list of current flags.

Remarks

If you are supporting multiple namespaces via the **VFSF_MULTIPLEFORMATS** flag then the capabilities flags you return may be different for each of your namespace types.

VFS_GetContextMenu

The **VFS_GetContextMenu** function is called to allow your plugin to add its own items to the context menu shown when the user clicks the right mouse button on a file or folder in your plugin's namespace.

```
BOOL VFS_GetContextMenu (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszFiles,
      LPVFSCONTEXTMENUDATA lpMenuData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszFiles
[in] A double null-terminated list of files that the context menu is being shown for.

lpwfdData
[in/out] Pointer to a **VFSCONTEXTMENUDATA** structure into which you place information about the context menu items you wish to add.

Return Values

If you wish to add items to the context menu you must set the appropriate fields in **VFSCONTEXTMENUDATA** and return **TRUE**. If you do not want to add to the context menu you should return **FALSE**.

Remarks

Custom commands you add to the context menu will invoke your **VFS_ContextVerb** function when the user selects them. See the **Context and Drop Menus** section for more information on building your context menu.

VFS_GetCustomColumns

The **VFS_GetCustomColumns** function is called to allow your plugin to add its own information columns to Directory Opus whenever your namespace is displayed in a Lister.

```
LPVFSCUSTOMCOLUMN VFS_GetCustomColumns (
      HANDLE hVFSData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

Return Values

If you wish to add custom columns you must return a pointer to a linked list of **VFSCUSTOMCOLUMN** structures. If you do not want to add custom columns you must return **NULL**.

Remarks

The linked list of **VFSCUSTOMCOLUMN** structures that you return will not be freed by Opus. It must remain valid for as long as the namespace instance handle remains valid (that is, until **VFS_Destroy** is called.)

Each custom column must be assigned an ID number via the *iID* field of the structure. These should be sequential beginning from 1.

See the **Specifying Custom Columns** section for more information on adding custom columns to the Lister.

VFS_GetDropMenu

The **VFS_GetDropMenu** function is called to allow your plugin to add its own items to the context menu shown when the user drags items with the right mouse button and drops them in your plugin's namespace.

```
BOOL VFS_GetDropMenu (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszFiles,
      LPVFSCONTEXTMENUDATA lpMenuData,
      DWORD dwEffects
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszFiles
[in] A double null-terminated list of files that the context menu is being shown for.

lpwfdData
[in/out] Pointer to a **VFSCONTEXTMENUDATA** structure into which you place information about the context menu items you wish to add.

dwEffects
[in] Bitmask of qualifier keys held down when the drop took place.

```
DROPEFFECT_COPY
   Control key held down – "normal" action is to copy files

DROPEFFECT_MOVE
   Shift key held down – "normal" action is to move files

DROPEFFECT_LINK
   Alt key held down – "normal" action is to create shortcuts

DROPEFFECT_NONE
   No qualifier keys held down
```

Return Values

If you wish to add items to the context menu you must set the appropriate fields in **VFSCONTEXTMENUDATA** and return **TRUE**. If you do not want to add to the context menu you should return **FALSE**.

Remarks

Custom commands you add to the context menu will invoke your **VFS_ContextVerb** function when the user selects them. See the **Context and Drop Menus** section for more information on building your context menu.

VFS_GetFileAttr

The **VFS_GetFileAttr** function is called to retrieve the attributes of a file or folder in your plugin's namespace.

```
BOOL VFS_GetFileAttr (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPDWORD lpdwAttr
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve attributes for.

lpdwAttr
[out] Points to a **DWORD** into which you must place the attributes of the specified file.

Return Values

If the file exists you should place its attributes in *lpdwAttr* and return **TRUE**. If the file does not exist or another error occurs you should return **FALSE**.

Remarks

The attribute flags are the same as those in the Windows API. See the documentation for the Windows **GetFileAttributes** function for a list of flags. You only need to support those flags that make sense in the context of your plugin.

VFS_GetFileComment

The **VFS_GetFileComment** function is called to retrieve the comment set by the user for a file or folder in your plugin's namespace.

```
BOOL VFS_GetFileComment (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPTSTR lpszComment,
      int cchCommentMax
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve the comment for.

lpszComment
[out] Points to a buffer into which you must copy the comment of the specified file.

cchCommentMax
[in] Specifies the size of the buffer in characters

Return Values

If the file has a comment set by the user you should copy it to the supplied buffer and return **TRUE**. If the file has no comment you should return **FALSE**.

Remarks

This function is not to be confused with **VFS_GetFileDescription**, which is called to obtain an automatically generated description of a file. The comment this function returns must have been set by the user.

VFS_GetFileDescription

The **VFS_GetFileDescription** function is called to retrieve an automatically generated description of the specified file.

```
int VFS_GetFileDescription (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPTSTR lpszDescription,
      int cchDescriptionMax
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve the description for.

lpszDescription
[out] Points to a buffer into which you must copy the description of the specified file.

cchDescriptionMax
[in] Specifies the size of the buffer in characters

Return Values

If you are able to generate a description for the specified file you should copy it to the supplied buffer and return **TRUE**. If your plugin cannot generate a description you should return **FALSE**.

Remarks

This function is not to be confused with **VFS_GetFileComment**, which is called to obtain a user-specified comment for a file. The description this function returns must be automatically generated by the plugin to succinctly describe the file.

VFS_GetFileIcon

The **VFS_GetFileIcon** function is called to retrieve the icon for a file or folder in your plugin's namespace.

```
BOOL VFS_GetFileIcon (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszFile,
      LPINT lpiSysIconIndex,
      HICON* phLargeIcon,
      HICON* phSmallIcon,
      LPBOOL lpfDestroyIcons,
      LPTSTR lpszCacheName,
      int cchCacheNameMax,
      LPINT lpiCacheIndex
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszFile
[in] A null-terminated string containing the full pathname of the file to retrieve the icon for.

lpiSysIconIndex
[out] Points to an integer to receive the index of the file's icon in the system image list.

phLargeIcon
[out] Points to an **HICON** to receive the large icon for the file

phSmallIcon
[out] Points to an **HICON** to receive the small icon for the file

lpfDestroyIcons
[out] Points to a **BOOL** that you can set to **TRUE** to indicate the supplied icons are to be destroyed

lpszCacheName
[out] Points to a buffer for you to supply a unique name used to cache the supplied icons

cchCacheNameMax
[in] Maximum size of buffer in characters

lpiCacheIndex
[out] Points to an integer to receive an index value used to cache the supplied icons

Return Values

If you can return an icon for the specified file you should store the icon information in the appropriate parameters and return **TRUE**. If your plugin cannot return an icon you should return **FALSE**.

Remarks

You can either use the *lpiSysIconIndex* parameter to return an index to an icon in the system image list, or *phLargeIcon* and *phSmallIcon* to return actual icons. To use the system image list, simply store the index value in *lpiSysIconIndex* and return **TRUE**.

To return icon images you must place the **HICON** handles in the supplied *phLargeIcon* and *phSmallIcon* parameters. If you want Opus to destroy the icons with **DestroyIcon** you should set *lpfDestroyIcons* to **TRUE** – otherwise, set *lpfDestroyIcons* to **FALSE** if you want to destroy the icons yourself. Note though that if you do not ask Opus to destroy them they must remain valid for as long as your namespace instance handle is valid (that is, until Opus destroys it with **VFS_Destroy**.)

Opus is able to cache icon images returned in this manner to save memory. To take advantage of this, copy a unique name into the buffer pointed to by *lpszCacheName*. For example, you could use a UUID or a combination of your plugin's name and a random number for the cache name. You can also provide a cache index in the *lpiCacheIndex* parameter.

VFS_GetFileInformation

The **VFS_GetFileInformation** function is called to retrieve information about a specified file or folder in your plugin's namespace.

```
LPVFSFILEDATAHEADER VFS_GetFileInformation (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      HANDLE hHeap,
      DWORD dwFlags
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve information for

hHeap
[in] A handle to a memory heap for you to allocate memory from using **HeapAlloc**.

dwFlags
[in] Currently unused.

Return Values

If the file exists you should allocate a chunk of memory using the heap provided containing a **VFSFILEDATAHEADER** and a **VFSFILEDATA** structure and return the address of the allocated memory. If the file does not exist you should return **NULL**.

Remarks

This function is very similar to **VFS_ReadDirectory** – the only difference is that information for only a single file is returned. The following code is an example of how to allocate the necessary data structure:

```
LPVOID lpMem = HeapAlloc(hHeap,0,sizeof(VFSFILEDATAHEADER) +
                                 sizeof(VFSFILEDATA) );
LPVFSFILEDATAHEADER pHeader=(LPVFSFILEDATAHEADER)lpMem;
LPVFSFILEDATA pFileData=(LPVFSFILEDATA)(pHeader+1);
// initialize structures
return pHeader;
```

VFS_GetFileSize

The **VFS_GetFileSize** function is called to retrieve the size of a file in your plugin's namespace.

```
BOOL VFS_GetFileSize (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      HANDLE hFile,
      unsigned __int64* piFileSize
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve the size of.

hFile
[in] A handle to the file to retrieve the size of.

piFileSize
[out] Pointer to an unsigned 64 bit value to receive the file size.

Return Values

If the file exists you should place its size in *piFileSize* and return **TRUE**. If the file does not exist or another error occurs you should return **FALSE**.

Remarks

Directory Opus will either specify the pathname of the file in *lpszPath* or will provide a **HANDLE** to the file in *hFile* – never both. If *hFile* is specified it will be a file handle created by your **VFS_CreateFile** function. Your **VFS_GetFileSize** function should be able to handle both cases.

VFS_GetFreeDiskSpace

The **VFS_GetFreeDiskSpace** function is called to retrieve information about the free and used space in your plugin's namespace.

```
BOOL VFS_GetFreeDiskSpace (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      unsigned __int64* piFreeBytesAvailable,
      unsigned __int64* piTotalBytes,
      unsigned __int64* piTotalFreeBytes
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the path to retrieve free disk space for.

piFreeBytesAvailable
[out] Points to an unsigned 64 bit value to receive the number of free bytes available for the current user (may be NULL)

piTotalBytes
[out] Points to an unsigned 64 bit value to receive the total number of bytes on the disk (may be NULL)

piTotalFreeBytes
[out] Points to an unsigned 64 bit value to receive the total number of free bytes on the disk (may be NULL)

Return Values

You should place the requested sizes in the supplied parameters and return **TRUE**. If an error occurs you should return **FALSE**.

Remarks

Directory Opus may not request all three sizes at once – you must check that the pointers are not NULL before placing data in them.

VFS_GetLastError

The **VFS_GetLastError** function is called to retrieve a code for the last error encountered by your plugin.

```
long VFS_GetLastError (
      HANDLE hVFSData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

Return Values

You should return the code of the last error that occurred or 0 if the last call was successful.

Remarks

You should maintain a “last error” variable in your namespace data structure. Whenever Opus calls one of your functions you should set this to an appropriate error code if the function fails, or to 0 if the function is successful.

VFS_GetPathDisplayName

The **VFS_GetPathDisplayName** function is called to convert a URL-style namespace path into a “pretty path” for display to the user.

```
BOOL VFS_GetPathDisplayName (
      HANDLE hVFSData,
      LPTSTR lpszPath,
      LPTSTR lpszDisplayName,
      int cchDisplayNameMax
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpszPath
[in] A null-terminated string containing the full path in your plugin’s namespace.

lpszDisplayName
[out] Points to a buffer into which you must copy the pretty path.

cchDisplayNameMax
[in] Specifies the size of the buffer in characters

Return Values

If you are able to return a pretty path for the supplied path, should copy it to the supplied buffer and return **TRUE**. If you do not wish to change the display of the path string you should return **FALSE**.

Remarks

The Opus internal FTP support uses this method. Internally, FTP paths are stored as and handled as ***ftp://user:password@ftpsite.com/path/to/folder***, but the path displayed to the user has the user and password information removed – **ftp://ftpsite.com/path/to/folder**.

If you return **FALSE** from this function (or do not provide the function at all) then Opus will display paths as-is to the user.

VFS_GetPathParentRoot

The **VFS_GetPathParentRoot** function is called to calculate the parent or root path of a path in your plugin's namespace.

```
BOOL VFS_GetPathParentRoot (
      HANDLE hVFSData,
      LPTSTR lpszPath,
      BOOL fRoot,
      LPTSTR lpszNewPath,
      int cchNewPathMax
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpszPath
[in] A null-terminated string containing the full path to calculate the parent or root of.

fRoot
[in] Set to **TRUE** for a root or **FALSE** for a parent.

lpszNewPath
[out] Points to a buffer into which you must copy the calculated path.

cchNewPathMax
[in] Specifies the size of the buffer in characters

Return Values

If you are able to calculate the path you should copy it to the supplied buffer and return **TRUE**. If it is not possible to calculate the desired path you should return **FALSE**.

Remarks

The **VFS_GetPathParentRoot** function can be provided by your plugin if you want to do custom path processing whenever Opus needs to calculate the parent or root folder of a folder in your plugin's namespace. Opus calls this function if you set the **VFSCAPABILITY_LETMEDOPARENTS** capabilities flag.

If you do not provide this function then Opus applies standard path parsing rules to calculate the desired path.

VFS_GetPrefixList

The **VFS_GetPrefixList** function is called to retrieve a list of URL-style prefixes supported by your plugin.

```
BOOL VFS_GetPrefixList (
      LPTSTR lpszPrefix,
      int cchPrefixMax
);
```

Parameters

lpszPrefix
[out] Points to a buffer into which you must copy your list of prefixes.

cchPrefixMax
[in] Specifies the size of the buffer in characters

Return Values

You should copy your prefix list into the buffer as a double null-terminated list of strings, and return **TRUE** for success. On failure you should return **FALSE**.

Remarks

This function is called if you specify the **VFSF_MULTIPLEFORMATS** plugin flag. Opus calls this function to retrieve a list of the URL-style prefixes your plugin supports. You must supply the prefix list as a double null-terminated list of strings.

VFS_Init

The **VFS_Init** function is called immediately after Opus first loads your plugin DLL. It lets you perform initialization that may not be safe to perform inside the **DllMain** function (eg, opening other DLLs.)

```
BOOL VFS_Init (
            LPVFSINITDATA lpInitData
);
```

Parameters

lpInitData
[in] Points to a **VFSINITDATA** structure.

Return Values

You should perform any initialization your plugin requires and then return **TRUE** to allow Opus to use your plugin. If you return **FALSE** Opus will not access your plugin again this session.

Remarks

The **hwndDOpusMsgWindow** field of the **VFSINITDATA** structure provides the window handle of the Plugin Support manager, which can be used to send several notification messages to Opus (documented in the Plugin Support API SDK.) The **dwOpusVerMajor** and **dwOpusVerMinor** fields provide the current version of Opus, and **pszLanguageName** provides the name of the currently selected language.

This function may seem similar to the **VFS_Identify** function, however Opus guarantees to only call **VFS_Init** once, immediately after your plugin DLL is loaded. Additionally, Opus will call the matching **VFS_Uninit** function when your plugin is released. You should use **VFS_Init** to perform any required initialization that may not be safe when placed in **DllMain**.

Note that if you return **FALSE**, Opus does **not** call **VFS_Uninit**.

VFS_Identify

The **VFS_Identify** function is called when Opus first initializes your plugin to retrieve information about the plugin itself.

```
BOOL VFS_Identify (
            LPVFSPLUGININFO lpVFSInfo
);
```

Parameters

lpVFSInfo
[out] Points to a **VFSPLUGININFO** structure for you to initialize.

Return Values

You should initialize the plugin info structure and return **TRUE**. If you return **FALSE** Opus will not access your plugin again this session.

Remarks

The **cbSize** field of the **VFSPLUGININFO** structure is used to maintain compatibility between different versions of this API and Opus. You should check the size of the structure specified and make sure you don't attempt to access fields beyond the size specified.

See the **Plugin Identification** section for more information on the **VFS_Identify** function.

VFS_MoveFile

The **VFS_MoveFile** function is called to rename a file or folder in your plugin's namespace, or to move a file or folder from one location to another.

```
BOOL VFS_MoveFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszOldPath,
      LPTSTR lpszNewPath
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszOldPath
[in] Null-terminated string representing the full path of the file to move or rename

lpszNewPath
[in] Null-terminated string representing the new pathname of the file

Return Values

The function must return **TRUE** if the file was successfully moved or renamed, or **FALSE** on failure. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError**.

Remarks

You must check the paths provided to determine whether this is a simple rename or a move operation. If you don't support moving of files from one folder to another via a rename you should return **FALSE** and set the error code to **VFSERR_NOT_SUPPORTED** – Opus will then fallback to moving the file via copy and delete.

VFS_Properties

The **VFS_Properties** function is called when the user wants to display a Properties dialog for one or more files in your plugin's namespace.

```
HWND VFS_Properties (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      HWND hwndParent,
      LPTSTR lpszFiles
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

hwndParent
[in] The parent window handle

lpszFiles
[in] Double null-terminated list of string representing the files to show Properties for

Return Values

For a modeless dialog, your function must return the **HWND** of the dialog window. For a modal dialog you should return **TRUE**. On failure you should return **0**.

Remarks

If you specify the **VFSCAPABILITY_COMBINEDPROPERTIES** flag and the user selects multiple files, *lpszFiles* will be a double null-terminated list of filenames. If only a single file is selected you can treat *lpszFiles* as an ordinary string. If you are showing combined properties your Properties dialog should combine the common properties of the selected files as best you can.

Your Properties dialog should resemble the standard Windows file Properties dialog as much as possible, to avoid confusing the user and to create a more seamless environment.

VFS_PropGet

The **VFS_PropGet** function is called to retrieve several properties for files or folders in your plugin namespace, or for the namespace itself.

```
BOOL VFS_PropGet (
      HANDLE hVFSData,
      vfsProperty propId,
      LPVOID lpPropData,
      LPVOID lpData1,
      LPVOID lpData2,
      LPVOID lpData3
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

propId
[in] The type of property to retrieve

lpPropData
[in/out] The meaning of this parameter changes depending on the property type

lpData1
[in/out] The meaning of this parameter changes depending on the property type

lpData2
[in/out] The meaning of this parameter changes depending on the property type

lpData3
[in/out] The meaning of this parameter changes depending on the property type

Return Values

The function must return **TRUE** if the property was successfully retrieved, or **FALSE** on failure. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError**.

Remarks

Generally, *lpPropData* will point to a variable or buffer for you to return the necessary property information, and *lpData1*, *lpData2* and *lpData3* will be used to provide additional information about the property request. However this is not always the case and you should read the documentation in the **Property Reference** section for a full listing of the property types supported by this function.

VFS_QueryPath

The **VFS_QueryPath** function is called to discover if your plugin is able to create and display a namespace for a specific file extension, or for a URL-style path prefix.

```
BOOL VFS_QueryPath (
      LPTSTR lpszPath,
      BOOL fPrefix,
      LPGUID pGUID
);
```

Parameters

lpszPath
[in] Pointer to a string containing the prefix or file extension to test

fPrefix
[in] Set to **TRUE** if *lpszPath* is a URL-style path prefix

pGUID
[out] Pointer to a **GUID** structure for you to return a unique identifier for this namespace type

Return Values

The function must place a **GUID** in the *pGUID* parameter and return **TRUE** if the path or filename extension can be handled, or **FALSE** if it cannot.

Remarks

This function is used if you specify the **VFSF_MULTIPLEFORMATS** plugin flag. The **GUID** you return must be unique for the type of namespace you will use to handle this file or prefix. This must be different from the GUID you supplied in response to the **VFS_Identify** function.

See the **Namespace Identification** section for more information on this function.

VFS_ReadDirectory

The **VFS_ReadDirectory** function is called to read the contents of a directory in your plugin's namespace.

```
BOOL VFS_ReadDirectory (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPVFSREADDIRDATA lpReadDirData
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpReadDirData
[in/out] Pointer to a **VFSREADDIRDATA** structure

Return Values

The function must return **TRUE** if the directory was read successfully, or **FALSE** if it could not.

Remarks

The directory contents are returned in a variable length linked list of variable size arrays of **VFSFILEDATA** structures. Each array must be preceded in memory with a **VFSFILEDATAHEADER** structure. All memory returned by this function must be allocated with **HeapAlloc** using the memory heap provided. A pointer to the first **VFSFILEDATAHEADER** structure must be placed in the *lpFileData* field of the **VFSREADDIRDATA** structure.

Several read operations are defined that do **not** require you to return any directory contents. *lpFileData* should be set to 0 in these cases.

See the **Reading a Directory** section for more information on this function.

VFS_ReadFile

The **VFS_ReadFile** function is called to read data from an open file.

```
BOOL VFS_ReadFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      HANDLE hFile,
      LPVOID lpData,
      DWORD dwSize,
      LPDWORD lpdwReadSize
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

hFile
[in] File handle as returned by your **VFS_CreateFile** function

lpData
[out] Pointer to a buffer to receive the data

dwSize
[in] Size of the buffer

lpdwReadSize
[out] Pointer to a DWORD to receive the amount of data read

Return Values

The function must return **TRUE** if data was successfully read from the file, or **FALSE** on failure or end-of-file. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError** – the error code should be set to 0 for an end-of-file condition.

Remarks

When this function is called you should read as much data as possible up to and including *dwSize* bytes from the file into the buffer provided. You should not block or wait for additional data to become available – only return as much data as you can (up to the provided buffer size). The amount of data returned must be placed in *lpdwReadSize*.

VFS_RemoveDirectory

The **VFS_RemoveDirectory** function is called to delete a sub-directory from your plugin's namespace.

```
BOOL VFS_RemoveDirectory (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      DWORD dwFlags
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] Null-terminated string representing the full path of the directory to delete

dwFlags
[in] Currently unused.

Return Values

The function must return **TRUE** if the folder was successfullydeleted, or **FALSE** on failure. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError**.

Remarks

Opus may occasionally attempt to delete non-empty directories via this function. If your plugin supports this you should perform the delete as normal and return **TRUE**. If you don't support deleting non-empty directories you should set the error code to **ERROR_DIR_NOT_EMPTY** and return **FALSE**.

VFS_SeekFile

The **VFS_SeekFile** function is called to set the file pointer within an open file.

```
BOOL VFS_SeekFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      HANDLE hFile,
      __int64 iPos,
      DWORD dwMethod,
      DWORD dwFlags,
      unsigned __int64* piNewPos
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

hFile
[in] File handle as returned by your **VFS_CreateFile** function

iPos
[in] Position to seek (either relative or absolute)

dwMethod
[in] Method of seeking in the file

```
FILE_BEGIN
   iPos is relative to the start of the file

FILE_CURRENT
   iPos is relative to the current position

FILE_END
   iPos is relative to the end of the file
```

dwFlags
[in] Flags describing this seek operation

```
VFSSEEKF_RESUME
   The seek is occurring as part of a copy resume operation
```

piNewPos
[out] Pointer to an unsigned 64 bit value to receive the new file pointer or NULL

Return Values

The function must return **TRUE** if seek was successful, or **FALSE** on failure. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError**. If *piNewPos* is non-NULL you should return the new position of the file pointer in this parameter.

Remarks

It is valid for this function to be called with *iPos* set to 0 and *dwMethod* set to **FILE_CURRENT**. In this case you should simply return the current file position in *piNewPos* and return **TRUE**.

If your function does not support random seeking you should return an error of **VFSERR_NOT_SUPPORTED** when appropriate. You should still endeavor to support forward seeking if possible.

VFS_SetFileAttr

The **VFS_SetFileAttr** function is called to set the attributes of a file or folder in your plugin's namespace.

```
BOOL VFS_SetFileAttr (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      DWORD dwAttr,
      BOOL fForDelete
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve attributes for.

dwAttr
[in] Specifies the new attributes for the file.

fForDelete
[in] Set to **TRUE** if this is being called to unprotect a file for a delete operation

Return Values

If the attributes were successfully changed you should return **TRUE**. If the file does not exist or another error occurs you should return **FALSE**.

Remarks

The attribute flags are the same as those in the Windows API. See the documentation for the Windows S**etFileAttributes** function for a list of flags. You only need to support those flags that make sense in the context of your plugin.

VFS_SetFileComment

The **VFS_SetFileComment** function is called to allow the user to assign an arbitrary comment to a file or folder in your plugin's namespace.

```
BOOL VFS_SetFileComment (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPTSTR lpszComment
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to retrieve the comment for.

lpszComment
[in] Points to a string containing the new comment for the file.

Return Values

If you can successfully store the comment you should return **TRUE**, otherwise set an appropriate error code and return **FALSE**.

Remarks

If *lpszComment* is NULL or an empty string you should remove any previously assigned comment.

VFS_SetFileTime

The **VFS_SetFileTime** function is called to change one or more timestamps for a file in your plugin's namespace.

```
BOOL VFS_SetFileTime (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      LPTSTR lpszPath,
      LPFILETIME lpCreateTime,
      LPFILETIME lpAccessTime,
      LPFILETIME lpWriteTime,
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

lpszPath
[in] A null-terminated string containing the full pathname of the file to set the timestamps of.

lpCreateTime
[in] A pointer to the **FILETIME** representing the new creation time, or NULL.

lpAccessTime
[in] A pointer to the **FILETIME** representing the new last access time, or NULL.

lpWriteTime
[in] A pointer to the **FILETIME** representing the new last modification time, or NULL.

Return Values

If the timestamp was set successfully you should return **TRUE**. If the file does not exist or another error occurs you should set an appropriate error code and return **FALSE**.

Remarks

Not all of the three filetimes are necessarily provided in each call to this function – you must check that they are non-NULL before attempting to access them. Directory Opus generally does not call this function to change the last access time of a file. Last modification time is the most common timestamp changed, followed by creation time. If you only support one type of time stamp you should treat that as last modification time and ignore any other timestamps provided.

VFS_Uninit

The **VFS_Uninit** function is called immediately before Opus unloads your plugin DLL. It lets you perform cleanup of any initialization that was performed in the **VFS_Init** function.

```
void VFS_Uninit ( void );
```

Parameters

None

Return Values

None.

Remarks

VFS_Uninit is called immediately before Opus calls **FreeLibrary** on your plugin DLL, *unless* you returned **FALSE** in **VFS_Init**.

VFS_USBSafe

The **VFS_USBSafe** function is called when the user is performing an export of their Opus installation to a USB drive. It lets you indicate that your plugin is safe to run off USB and also lets you specify other files that must be exported alongside your plugin.

```
BOOL VFS_USBSafe (
      LPOPUSUSBSAFEDATA lpUSBSafeData
);
```

Parameters

lpUSBSafeData
[in,out] A pointer to an **OPUSUSBSAFEDATA** structure

Return Values

The function must return **TRUE** if the plugin is to be allowed to be exported to a USB device. If you return **FALSE**, or do not export this function at all, the user will be unable to export your plugin to USB drives.

Remarks

You should not indicate that you are USB safe unless that really is the case. Generally, this simply means storing all configuration data in XML files using the functions provided by the Plugin Support API, rather than writing to the registry. Opus plugins should not make any changes to the host system when running in USB mode.

You can use the provided **OPUSUSBSAFEDATA** structure to indicate other files that must be exported alongside your plugin. If your plugin requires additional support files (eg, the unrar plugin requires the unrar.dll library), copy them into the provided **pszOtherExports** field. They must be provided as a double-null terminated list of filenames, relative to the Opus VFS Plugins folder. The **cchOtherExports** field specifies the size of the provided buffer.

Note that there is no ANSI version of this function.

VFS_WriteFile

The **VFS_WriteFile** function is called to write data to an open file.

```
BOOL VFS_WriteFile (
      HANDLE hVFSData,
      LPVFSFUNCDATA lpFuncData,
      HANDLE hFile,
      LPVOID lpData,
      DWORD dwSize,
      BOOL fFlush,
      LPDWORD lpdwWriteSize
);
```

Parameters

hVFSData
[in] The handle to the namespace instance data

lpFuncData
[in] unused

hFile
[in] File handle as returned by your **VFS_CreateFile** function

lpData
[in] Pointer to a buffer containing the data to write

dwSize
[in] Size of the data in bytes

fFlush
[in] Set to **TRUE** if you should flush any write buffers to disk after completing this write operation

lpdwWriteSize
[out] Pointer to a DWORD to receive the amount of data written successfully

Return Values

The function must return **TRUE** if data was successfully written to the file, or **FALSE** on failure or end-of-file. You should set an appropriate error code on failure that Opus can retrieve via **VFS_GetLastError** – the error code should be set to 0 for an end-of-file condition.

Remarks

You must write as much data as possible up to and including *dwSize* bytes to your file from the buffer provided. The amount of data actually written must be placed in *lpdwWriteSize*.

Capabilities Flags Reference

These flags are used in the *dwCapabilities* field of the **VFSPLUGININFO** structure, and as a return value for the **VFS_GetCapabilities** function.

VFSCAPABILITY_MOVEBYRENAME
Indicates that your plugin is able to move files by renaming them. If this flag is specified and the user attempts to move a file from one folder to another, Opus will call your VFS_MoveFile function to perform the operation. If this flag is not specified, moving a file will involve creating a copy of it and then deleting the original.

VFSCAPABILITY_COPYINDEFINITESIZES
Indicates that the file sizes reported by your plugin are not necessarily accurate. This is used when the user copies files from your plugin's namespace. If this flag is set Opus will ignore the stated size of a file and continue to read data until you return an end-of-file indicator. If this flag is not set Opus will only read the number of bytes you have reported for the size of the file.

VFSCAPABILITY_CANRESUMECOPIES
Indicates that your plugin can resume interrupted file transfers (or file copy operations). If this flag is set and the user attempts to resume a file transfer either to or from your plugin's namespace, Opus will call your VFS_SeekFile function to position the file pointer appropriately.

VFSCAPABILITY_TRIGGERRESUME
Set this flag if you want your plugin to be the trigger for a resume of an interrupted copy. If you set this flag and the user attempts to copy a file to or from your namespace that already exists, Opus will give them the option of resuming the transfer. As an example of how this flag differs from VFSCAPABILITY_CANRESUMECOPIES, the internal FTP namespace in Opus sets both flags, whereas the standard file system namespace only sets VFSCAPABILITY_CANRESUMECOPIES. This means that a user copying an existing file between two file system folders will not be given the option of resuming, but a user copying from or to an FTP namespace will be asked if they wish to resume or not. The standard file system namespace supports resume but only the FTP namespace triggers the option.

VFSCAPABILITY_POSTCOPYREREAD
If this flag is set Opus will automatically trigger a refresh of the destination Lister whenever files are copied to it or removed from it.

VFSCAPABILITY_CASESENSITIVE
Set this flag if filenames in your namespace are case-sensitive.

VFSCAPABILITY_RANDOMSEEK
Set this flag if you support random seeking within files. If you only support sequential seeking, or do not support seeking at all, do not set this flag. Opus does not generally use or require random seeking, but some viewer plugins may require the ability.

VFSCAPABILITY_FILEDESCRIPTIONS
Set this flag if you want your plugin to be able to provide description strings for files in its namespace. Opus will call your VFS_GetFileDescription function to retrieve descriptions for files (for example, if the user displays the Description field in a Lister). Note that the strings returned by this function are not (necessarily) user supplied comments – they can contain any information you desire. A separate function pair (VFS_GetFileComment / VFS_SetFileComment) is used to implement for user-editable comments.

VFSCAPABILITY_ALLOWMUSICCOLUMNS
Set this flag if you want the music-related Lister information fields to be available in your plugin's namespace. Note that Opus does not call your plugin to provide this information – it uses its own routines to open files in your namespace and parse them for the needed information. You can use the VFS_GetCustomColumns function to provide your own information columns for display in Listers.

VFSCAPABILITY_ALLOWIMAGECOLUMNS
Similar to the VFSCAPABILITY_ALLOWMUSICCOLUMNS flag, this flag indicates that you want the image-related Lister information fields to be available in your plugin's namespace.

VFSCAPABILITY_ALLOWEXTRADATECOLUMNS
This flag indicates that you want the last accessed and creation date fields to be available in your plugin's namespace. If this flag is not set the only date fields available will be for last modified date.

VFSCAPABILITY_LETMEDOPARENTS
If this flag is set then Opus will call your VFS_GetPathParentRoot function whenever it needs to calculate the parent or root of a path in your plugin's namespace. If not specified, Opus will apply standard parsing rules to calculate the desired path.

VFSCAPABILITY_COMBINEDPROPERTIES
Set this flag if your plugin is able to display a combined Properties sheet for multiple files. If this flag is set and the user requests the properties of multiple files at once in your plugin's namespace, Opus will call your VFS_Properties function with a double-null terminated list of files to display properties for. If this flag is not set, Opus will call your VFS_Properties function once for each selected file.

VFSCAPABILITY_COMPARETIMENOSECONDS
Set this flag if your plugin does not report or preserve seconds in file times. If this flag is set Opus will discard or ignore the seconds of any file times it needs to compare with times provided by your plugin.

VFSCAPABILITY_GETBATCHFILEINFO
Set this flag if your plugin is able to handle asynchronous requests for file information (which may involve opening and reading file data). If this flag is set, Opus will launch a background thread to read all required or desired file information for all files in the folder whenever a directory is read. If this flag is not set, Opus will only request file information when needed.

VFSCAPABILITY_SLOW
Set this flag if your plugin represents a slow device or media. This flag is passed to viewer plugins as an indication that accesses to your plugin's namespace may take longer than expected. Additionally, Opus will not attempt to determine file type by reading the contents of the file on a slow device, and may also refrain from attempting to extract some file information in some cases.

VFSCAPABILITY_MULTICREATEDIR
Set this flag if your plugin supports the creation of multiple directories simultaneously. If set, your VFS_CreateDirectory function should be able to handle a comma-separated list of folders to create.

VFSCAPABILITY_ALLOWFILEHASH
Set this flag if you want your plugin to allow the hashing of files within its namespace. The actual hashing is performed by Directory Opus (currently using MD5 functions) – all that is required of your plugin if this flag is set is the ability to read sequentially from files within your namespace. If access to your files is particularly slow you may wish to disable the hash functionality.

VFSCAPABILITY_READONLY
This flag should be set if your plugin is read-only – that is, if you do not support the creation of, writing to or deleting of files within your plugin's namespace. The example unrar plugin sets this flag as the required support library does not support the creation of rar archives.

VFSCAPABILITY_CHECKAVAILONDIRCHANGE

If this flag is set Opus will call your VFS_PropGet function to retrieve the VFSPROP_FUNCAVAILABILITY property and update the state of any toolbar buttons whenever a new folder is read within your plugin's namespace. This allows you to selectively enable or disable file functions on a per-folder basis and have the user interface reflect this automatically.

Property Reference

The following property values are used with **VFS_PropGet**. Note that it is possible for a parameter which is documented as containing a path to be NULL so you should always check the value before attempting to access it.

Property: **VFSPROP_ISEXTRACTABLE**
Parameters: **lpPropData** = LPBOOL *pfIsExtractable*
lpData1 = LPTSTR *lpszFileName*
lpData2 = DWORD *dwAttr*
Description: Query whether a specified file or folder is *extractable* - that is, whether it can be extracted from the plugin's namespace to a local folder using the **VFS_ExtractFiles** function. *lpszFileName* provides the full pathname of the file in question, and *dwAttr* the file attributes (so you can check for the **FILE_ATTRIBUTE_DIRECTORY** attribute if needed.) *pfIsExtractable* points to a **BOOL** value – you should set this value to **TRUE** if the file can be extracted.

Property: **VFSPROP_USEFULLRENAME**
Parameters: **lpPropData** = LPBOOL *pfUseFullRename*
lpData1 = LPTSTR *lpszPath*
Description: Query whether the specified folder supports the "full rename" function. *lpszPath* provides the path of the folder in question, and *pfUseFullRename* points to a **BOOL** value. Set this value to **FALSE** if you only want the user to be able to use the "simple rename" function in your namespace.

Property: **VFSPROP_SHOWFULLPROGRESSBAR**
Parameters: **lpPropData** = LPDWORD *pdwShowFullProgress*
lpData1 = LPTSTR *lpszPath*
lpData2 = LPTSTR *lpszOtherPath*
lpData3 = BOOL *fIsDest*
Description: Query whether the "full progress" bar should be shown for a copy operation involving the two specified paths. *lpszPath* is the path in your plugin's namespace. *lpszOtherPath* is the other path involved in the operation (although this may be NULL). *fIsDest* indicates if your plugin is the destination or source of the copy operation.

pdwShowFullProgress is a pointer to a DWORD. You should set the LOWORD of this DWORD to **TRUE** if you want a full progress bar displayed, or **FALSE** if you do not. You should set the HIWORD to **TRUE** if you want to prevent a full progress bar from being displayed (otherwise, the other namespace involved in the copy may also enable the full progress bar.)

Property: **VFSPROP_CANDELETETOTRASH**
Parameters: **lpPropData** = LPBOOL *pfCanDeleteToTrash*
lpData1 = LPTSTR *lpszPath*
Description: Query whether the file specified by *lpszPath* can be deleted to the recycle bin. If it can you should return **TRUE** in *pfCanDeleteToTrash*, otherwise you should return **FALSE**.

Property: **VFSPROP_CANDELETESECURE**
Parameters: **lpPropData =** LPBOOL *pfCanDeleteSecure*
lpData1 = LPTSTR *lpszPath*
Description: Query whether the specified file can be deleted securely or not. *lpszPath* provides the file or folder name. If your **VFS_DeleteFile** function can delete it securely you should return **TRUE** in *pfCanDeleteSecure*, otherwise you should return **FALSE**.

Property: **VFSPROP_COPYBUFFERSIZE**
Parameters: **lpPropData** = LPDWORD *pdwBufferSize*
lpData1 = LPTSTR *lpszPath*
lpData2 = unsigned __int64* *puiFileSize*
Description: Query the recommended buffer size for a copy operation involving the specified path. If *lpszPath* is NULL you should just return your preferred buffer size. *puiFileSize* may be NULL, or it may point to an unsigned 64 bit integer specifying the size of the file to be copied. You should return your recommended buffer size (in bytes) in *pdwBufferSize*.

Property: **VFSPROP_DRAGEFFECTS**
Parameters: **lpPropData** = LPDWORD *pdwDragEffectsAvailable*
lpData1 = LPTSTR *lpszPath*
Description: Query the drag effects available when dragging from the path specified by *lpszPath*. You should return the effects value in *pdwDragEffectsAvailable*. Valid effects are **DROPEFFECT_COPY**, **DROPEFFECT_MOVE**, **DROPEFFECT_LINK** or **DROPEFFECT_NONE**.

Property: **VFSPROP_SHOWTHUMBNAILS**
Parameters: **lpPropData** = LPBOOL *pfShowThumbnails*
lpData1 = LPTSTR *lpszPath*
Description: Query whether thumbnails should be allowed for the path specified in *lpszPath*. You should return **TRUE** in *pfShowThumbnails* if you want to allow thumbnails.

Property: **VFSPROP_SHOWFILEINFO**
Parameters: **lpPropData** = LPBOOL *pfShowFileInfo*
lpData1 = LPTSTR *lpszPath*
Description: Query whether extractable file information should be supported for the pathname specified in *lpszPath*. You should return **TRUE** to allow the extraction of file information.

Property: **VFSPROP_ALLOWTOOLTIPGETSIZES**
Parameters: **lpPropData** = LPBOOL *pfAllowToolTipGetSizes*
lpData1 = LPTSTR *lpszPath*
Description: Query whether tooltips for the specified folder can be allowed to trigger a **GetSizes** command. Return **TRUE** to allow this or **FALSE** to prevent it.

Property: **VFSPROP_CANSHOWSUBFOLDERS**
Parameters: **lpPropData** = LPBOOL *pfCanShowSubFolders*
lpData1 = LPTSTR *lpszPath*
Description: Query whether the showing of sub-folders (flat view) should be allowed for the specified path. Return **TRUE** in *pfCanShowSubFolders* to allow this or **FALSE** to prevent it.

Property: **VFSPROP_FUNCAVAILABILITY**
Parameters: **lpPropData** = unsigned __int64* *pdwFuncAvailFlags*
lpData1 = LPTSTR *lpszPath*
Description: Query the availability of Opus functions for the specified path. *pdwFuncAvailFlags* is supplied with a mask of the functions that Opus is querying for – on return you should clear the flags for those functions you do not support. Opus uses this value to disable functions from the user. Defined function flags are:

```
VFSFUNCAVAIL_COPY
VFSFUNCAVAIL_MOVE
VFSFUNCAVAIL_DELETE
VFSFUNCAVAIL_GETSIZES
VFSFUNCAVAIL_MAKEDIR
VFSFUNCAVAIL_PRINT
VFSFUNCAVAIL_PROPERTIES
VFSFUNCAVAIL_RENAME
VFSFUNCAVAIL_SETATTR
VFSFUNCAVAIL_SHORTCUT
VFSFUNCAVAIL_SELECTALL
VFSFUNCAVAIL_SELECTNONE
VFSFUNCAVAIL_SELECTINVERT
VFSFUNCAVAIL_VIEWLARGEICONS
VFSFUNCAVAIL_VIEWSMALLICONS
VFSFUNCAVAIL_VIEWLIST
VFSFUNCAVAIL_VIEWDETAILS
VFSFUNCAVAIL_VIEWTHUMBNAIL
VFSFUNCAVAIL_CLIPCOPY
VFSFUNCAVAIL_CLIPCUT
VFSFUNCAVAIL_CLIPPASTE
VFSFUNCAVAIL_CLIPPASTESHORTCUT
VFSFUNCAVAIL_UNDO
VFSFUNCAVAIL_SHOW
VFSFUNCAVAIL_DUPLICATE
VFSFUNCAVAIL_SPLITJOIN
VFSFUNCAVAIL_SELECTRESELECT
VFSFUNCAVAIL_SELECTALLFILES
VFSFUNCAVAIL_SELECTALLDIRS
VFSFUNCAVAIL_PLAY
VFSFUNCAVAIL_SETTIME
VFSFUNCAVAIL_VIEWTILE
VFSFUNCAVAIL_SETCOMMENT
```

Property: **VFSPROP_SUPPORTFILEHASH**
Parameters: **lpPropData** = LPBOOL *pfSupportFileHash*
lpData1 = LPTSTR *lpszPath*
Description: Query whether file hashing should be allowed for the specified path. Return **TRUE** in *pfSupportFileHash* to allow this or **FALSE** to prevent it.

Property: **VFSPROP_GETFOLDERICON**
Parameters: **lpPropData** = undefined
lpData1 = HICON* *phLargeIcon*
lpData2 = HICON* *phSmallIcon*
lpData3 = LPBOOL pfDestroyIcons
Description: Retrieve an icon to use to represent the plugin namespace itself. The icon will be displayed in the location field and in Favorites/Recent lists etc. If not provided a default icon is used instead. You should return the handle to large and small icons in *phLargeIcon* and *phSmallIcon*. Set *pfDestroyIcons* to **TRUE** to have Opus destroy these icon handles via **DestroyIcon** or set to **FALSE** if you do not want Opus to destroy them.

Property: **VFSPROP_SUPPORTPATHCOMPLETION**
Parameters: **lpPropData** = LPBOOL *pfSupportCompletion*
lpData1 = LPTSTR *lpszPath*
Description: Query whether Opus should support path completion for the specified path. Return **TRUE** in *pfSupportCompletion* to allow it or **FALSE** to prevent it.

Property: **VFSPROP_BATCHOPERATION**
Parameters: **lpPropData** = LPDWORD *pdwBatchResult*
lpData1 = LPTSTR *lpszPath*
lpData2 = LPTSTR *lpszDestPath*
lpData2 = UINT *uiOperation*
Description: Query whether Opus should call **VFS_BatchOperation** to perform a batch file operation. *lpszPath* specifies the path in your plugin's namespace, and *lpszDestPath* specifies the destination path for an extraction operation. *uiOperation* specifies the desired operation type. You should place one of the **VFS_BatchOperation** return codes in *pdwBatchResult* to indicate if the batch operation is to be handled. If any value other than **VFSBATCHRES_HANDLED** is returned, **VFS_BatchOperation** will not be called.

Property: **VFSPROP_GETVALIDACTIONS**
Parameters: **lpPropData** = **LPDWORD** *pdwActions*
lpData1 = **LPDWORD** *lpszFilePath*
Description: Query if certain shell actions are valid for a file or folder in your plugin's namespace. When called, the value pointed to by *pdwActions* contains a bitmask of the actions that Opus wishes to query. When you return you should clear the flags that are not supported for the object passed in *lpszFilePath*. The shell flags queried for are **SFGAO_CANRENAME**, **SFGAO_CANDELETE**, **SFGAO_CANCOPY**, **SFGAO_CANMOVE**, and **SFGAO_HASPROPSHEET**.

Property: **VFSPROP_SHOWPICTURESDIRECTLY**
Parameters: **lpPropData** = **LPDWORD** *pdwShowPictures*
Description: Query whether a plugin supports the direct viewing of pictures in the Opus viewer. Return **TRUE** in *pdwShowPictures* if pictures can be viewed directly (this implies that your plugin exports the **VFS_CreateFile** and **VFS_ReadFile** functions) or **FALSE** if images in your plugin's namespace must be extracted to local disk before they can be viewed.