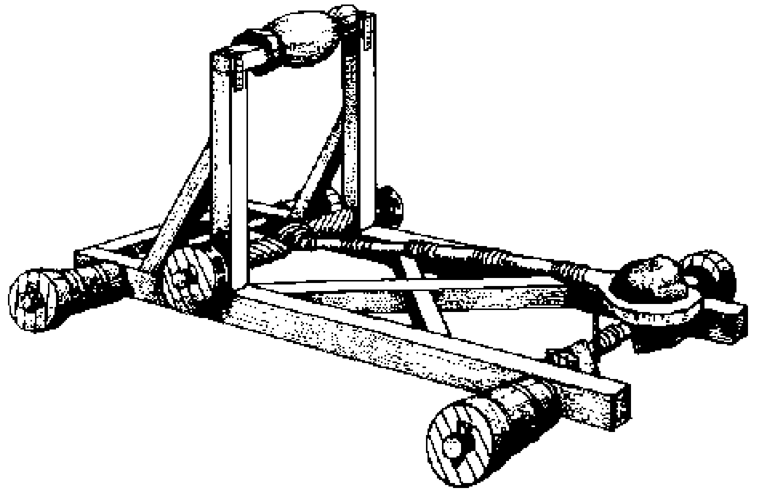
Microsoft Internet Extensions for Win32



Revision 0.54 (DRAFT)
March 23, 1995

Microsoft Internet Extensions for Win32

Table of Contents

1Background and Motivation

This specification describes the *Microsoft Internet Extensions for Win32,* an extension to the Win32 API set which allow Win32 applications easy access to common Internet protocols. The Internet Extensions for Win32 abstract the Gopher, FTP (File Transfer Protocol), Archie, and HTTP (HyperText Transfer Protocol, the basis of the World Wide Web or "WWW") protocols into easy-to-use, task focused interfaces which significantly simplify application use of the Internet. MIME support is also included to provide a single "registry" for Internet applications to access

These APIs are exported from a single DLL called **wininet.dll.** Initially, the APIs will be shipped independently of any operating system through channels such as MSDN, Compuserve, and of course on the Internet itself. ISVs will be granted rights to redistribute wininet.dll with their applications, following the model of Win32s. Over the long term, the APIs described in this specification are expected to be folded into all Microsoft operating systems.

1.1Goals

The high-level goal of the Internet Extensions for Win32 is to make it easy for applications to enable access to the Internet. This is achieved through a number of specific goals:

- **Free applications from needing to embed knowledge of TCP/IP and Windows Sockets.** By abstracting the Internet protocols into task-oriented APIs, the Internet Extensions for Win32 free applications from having to write Windows Sockets code or know about the complex TCP/IP protocol. One or several Windows Sockets API calls may be executed by each Internet API call. Applications need not be aware of this.

- **Free applications from needing to embed knowledge of Internet protocols.** While the concepts supported by the Internet protocols like FTP and HTTP are simple, the actual implementations of these protocols can be complex. For example, FTP servers return ASCII text file directory listings; parsing these listings requires specific knowledge of the format returned by each type of FTP server. By encapsulating this functionality underneath the Internet APIs, directory parsing is solved once for all applications which use the FTP protocol, thereby providing consistent behavior across applications.

- **Provide applications a constant API set in the face of rapidly changing and evolving protocols.** The Internet can move incredibly fast, and keeping up with the changes in protocols can make writing an application a difficult and frustrating experience. By defining an API set which remains constant, application writers need not change their application every time the underlying protocol changes; only wininet.dll needs to change. In addition, advanced protocol features like firewall-friendly FTP and Gopher+ can be implemented without having to change applications.

- **Follow Win32 API standards.** The Internet Extensions for Win32 look much like the traditional Win32 APIs in terms of how they treat buffer management, error returns, and the like. Application programmers familiar with the Win32 API set will find the Internet Extensions for Win32 return information in a manner which is natural to the Win32 programmer, and using the returned information in other Win32 APIs is simple.

- **Provide both ANSI and Unicode versions of APIs.** Although the underlying Internet protocols invariably use Latin 1[1] character sets, the Internet Extensions for Win32 provide both ANSI and Unicode entry points to facilitate Unicode-only applications.

- **Provide full access to Internet protocols.** The Internet Extensions for Win32 focus on providing easy access to the common functionality of the supported Internet protocols. However, in some instances applications will need to access extended features of some protocols. The Internet Extensions for Win32 make every effort to provide such access.

- **Enable high-performance, multithreaded Internet applications.** The Internet Extensions for Win32 are designed to be fully reentrant and multithread safe. Multithreaded applications can make simultaneous calls into the APIs from different threads without adverse effects. The Internet APIs themselves do any necessary synchronization.

1.2Non-Goals

The Internet Extensions for Win32 do not attempt to solve every possible Internet interfacing issue:

- **Access for Internet servers.** The Internet Extensions for Win32 are focused on making Internet client applications easier; making it easier to write Internet servers is not a goal because servers, in order to achieve the type high performance necessary in industrial-strength servers, need to be able to control how the protocol is accessed and how I/O is performed.

- **Mail, News APIs.** The Microsoft solution for mail and news APIs is MAPI. The Internet Extensions for Win32 make no effort to solve the general issue of access to mail and news servers.

[1] Latin 1 and ANSI are very similiar, but differ in a few character mappings. The ANSI versions of the Internet Extensions for Win32 do not attempt to perform any mapping between ANSI and Latin 1; they simply return exactly what they receive from the server and pass to the server exactly what they receive from the application.

2API Overview

The table below summarizes the APIs included in the Internet Extensions for Win32 in the first version. The indentation of each API shows dependencies between the APIs: indented APIs may only be called after the unindented APIs that precede them. Typically, this is because the less indented API returns a handle and sets up some state at the protocol level, all of which is required to execute successfully the later API.

InternetOpen	Initializes the application's use of the Internet Extensions for Win32.
InternetOpenUrl*	Begins retrieving an FTP, Gopher, or HTTP URL.
InternetReadFile	Reads URL data.
InternetCloseHandle	Stops reading data from the URL.
InternetSetStatusCallback*	Sets a function which is called with status information.
InternetQueryOption*	Queries the setting of an Internet option.
InternetSetOption*	Sets an Internet option.
MIME	
MimeCreateAssociation	Associates a MIME type string with an executable
MimeDeleteAssociation	Removes a MIME association
MimeGetAssociation	Returns information about MIME associations on the machine
FTP	
InternetConnect	Opens an FTP session with a server. Logs on the user.
FtpFindFirstFile	Starts file enumeration in the current directory.
InternetFindNextFile	Continues file enumeration.
InternetCloseHandle	Ends directory file enumeration.
FtpGetFile	Retrieves an entire file from the server.
FtpPutFile	Writes an entire file to the server.
FtpDeleteFile	Deletes a file on the server.
FtpRenameFile	Renames a file on the server.
FtpOpenFile	Initiates access to a file on the server for either reading or writing.
InternetReadFile	Reads data from an open file.
InternetWriteFile	Writes data to an open file.
InternetCloseHandle	Ends reading/writing to/from a file on the server.
FtpCreateDirectory	Creates a new directory on the server.
FtpRemoveDirectory	Deletes a directory on the server.
FtpSetCurrentDirectory	Changes the client's current directory on the server.
FtpGetCurrentDirectory	Returns the client's current directory on the server.
FtpCommand	Issues an arbitrary command to the FTP server.
InternetGetLastResponseInfo	Retrieves the text of the server's response to the FTP command.
InternetCloseHandle	Closes the FTP session.
Gopher	
InternetConnect	Indicates a Gopher server the application is interested in accessing

GopherFindFirstFile	Starts enumerating a Gopher directory listing
InternetFindNextFile	Continues Gopher directory enumeration
InternetCloseHandle	Terminates Gopher directory enumeration
GopherOpenFile	Starts retrieval of a Gopher object
InternetReadFile	Reads data from a Gopher object
InternetCloseHandle	Completes the reading of a Gopher object
GopherCreateLocator	Forms a Gopher locator for use in other Gopher function calls
GopherGetAttribute	Retrieves attribute information on the Gopher object
InternetCloseHandle	Indicates that the application is no longer interested in the server
HTTP (World Wide Web)	
InternetConnect	Indicates an HTTP server the application is interested in accessing
HttpOpenRequest	Opens an HTTP request handle
HttpAddRequestHeaders	Adds HTTP request headers to the HTTP request handle
HttpSendRequest	Sends the specified request to the HTTP server
InternetReadFile	Reads a block of data from an outstanding HTTP request
HttpQueryInfo	Queries information about an HTTP request
InternetCloseHandle	Closes an open HTTP request handle
InternetCloseHandle	Indicates that the application is no longer interested in the server
Archie	
ArchieFindFirstFile*	Starts an Archie query
InternetFindNextFile	Continues an Archie query
InternetCloseHandle	Ends the Archie query
InternetCloseHandle	Completes the application's use of the Internet Extensions for Win32

* **Unimplemented functions:** Not all of the functions described in this specification are implemented in the March 1995 "Alpha" release of the Internet Extensions for Win32. The functions which are not implemented are indicated by an asterisk following the name of the function wherever the name appears in this specification. These unimplemented functions are included in this specification to provide the reader with an indication of functions which may be implemented in the future.

2.1Handles

The handles returned by the Internet Extensions for Win32 are not native system handles; that is, Win32 routines like **ReadFile** and **CloseHandle** will not work on Internet handles. The Internet handles may only be used for the corresponding Internet APIs for which they are intended.

Any of the handles returned from the Internet APIs is closed with the **InternetCloseHandle** function. If a handle is closed while there is still outstanding activity on the handle, the activity is aborted. *Failing to close valid handles returned from the Internet APIs may result in resource leaks.*

2.2MIME Integration

The HTTP and Gopher protocols return MIME (Multimedia Internet Mail Extensions) information about files. In addition to the functions for handling various Internet protocols, the Internet Extensions for Win32 include functions for accessing MIME information in a standard fashion. The existence of these MIME functions free applications from having to embed knowledge of the registry or .INI files which contain the mappings between file extensions and the associated viewer executables. This is convenient for users with multiple Internet applications from different vendors.

In addition, the MIME functions allow for mappings of MIME type information (for example, “application/winword”) to both file extensions (for example, “.DOC”) and viewer executables (“winword.exe”). This allows Internet applications which download data files to launch the standard viewer application associated with any downloaded file based on the MIME content type string returned with the file.

2.3Error Handling

The Internet Extensions for Win32 return error information in the same fashion as all Win32 APIs. Function return values indicate whether a function is successful or not, either by returning a **BOOL** where TRUE is success and FALSE indicates failure, or by returning a handle of type **HINTERNET** which is NULL in the case of failure and any other value for a successful call.

If a function fails, the application may call the Win32 **GetLastError** function to retrieve a specific error code for the failure. In addition, the FTP and Gopher protocols allow for servers to return additional textual error information. For these protocols, applications may use the **InternetGetLastResponseInfo** function to retrieve error text.

Both **GetLastError** and **InternetGetLastResponseInfo** operate on a per-thread basis. In other words, if two threads make Internet function calls at about the same time, any error information will be returned for each of the individual threads so that there is no conflict between the threads. This also means that only the thread which made a function call may retrieve the error information for the function.

2.4Multi-threaded Access

All the Internet Extensions for Win32 are “reentrant” in the sense that there may be multiple calls to an individual function from different threads. All the functions do any necessary synchronization. However, multiple simultaneous calls using the same Internet connection may lead to unpredictable results.

For example, if an application has used **FtpOpenFile** to begin downloading a file from an FTP server and two threads simultaneously make calls to **InternetReadFile**, there is no guarantee as to which thread’s call will complete first or the order of the data returned to each thread. Applications which use multiple threads for the same Internet connection are responsible for any necessary synchronization between threads to ensure predictable return of information.

2.5Canceling Requests

All of the Internet Extensions for Win32 are synchronous. In some instances, an application may want to cancel an outstanding request because of user action. To cancel a request, use the **InternetCloseHandle** function to close the handle on which the request is outstanding. Canceling a request in this fashion will typically abort the connection to the server, requiring the application to reestablish the connection if it wants to communicate further with the server.

2.6Unicode Support

All the Internet Extensions for Win32 that take string arguments on input or output have both ANSI and Unicode versions. As with all Win32 APIs, the ANSI functions have “A” as the final character of their name and the Unicode functions have “W”. On both Windows NT and Windows 95, both the ANSI and Unicode versions of the functions are implemented..

Because the underlying Internet protocols pass all information in Latin 1 (very similiar to ANSI), the Unicode versions of the Win32 Internet functions must do translations to and from ANSI. Because in some cases it is not possible to convert a Unicode string to ANSI, the Unicode functions may fail. However, this will not typically be a problem because the cases where the translation fails are where there is no ANSI equivalent, so the requested object could not exist.

None of the Win32 Internet functions translate sent or received *content*. For example, when calling a Unicode function to retrieve an file from an FTP server, the application must specify the file name in Unicode, but the file data is returned to the application exactly as the FTP server has it stored.

3General Internet APIs

3.1InternetOpen

The **InternetOpen** function initializes an application's use of the Win32 Internet APIs.

HINTERNET InternetOpen(
 LPCTSTR *lpszCallerName,*
 DWORD *dwAccessType,*
 LPVOID *lpszGatewayName*
);

Parameters

lpszCallerName

A string that identifies the name of the application or entity that is calling the Internet APIs, for example "Microsoft Internet Browser." This name is used as the user agent in the HTTP protocol.

dwAccessType

type of access required. Valid parameters are:

PRE_CONFIG_INTERNET_ACCESS - pre-configured (registry)

LOCAL_INTERNET_ACCESS (direct-to-Internet)

GATEWAY_INTERNET_ACCESS (via gateway)

lpGatewayName

Name of preferred gateway if GATEWAY_INTERNET_ACCESS is requested.

Return Value

InternetOpen returns a valid handle that the application passes to subsequent Internet Extensions for Win32. If **InternetOpen** fails, it returns NULL and the application may retrieve a specific error code with the **GetLastError** function.

Remarks

InternetOpen is the first Win32 Internet function called by an application. It tells the Internet DLL to initialize internal data structures and prepare for future calls from the application. When the application is done using the Internet APIs, it should call **InternetCloseHandle** to free the **HINTERNET** and any other resources.

See Also

InternetCloseHandle

3.2InternetCloseHandle

The **InternetCloseHandle** function is used to close any Internet handle opened by an application.

BOOL InternetCloseHandle(
 HINTERNET *hInet*
);

Parameters

hInet

A valid Internet handle to be closed.

Return Value

TRUE if the handle was closed successfully, or FALSE if there was an error, in which case the application may call **GetLastError** to retrieve a specific error code.

Remarks

InternetCloseHandle is used to close all Internet handles (type **HINTERNET**) and free any resources associated with the handle. If there are any pending operations on the handle they are aborted and any outstanding data is discarded. If a thread is blocking in a call to the Internet DLL, another thread in the application may call **InternetCloseHandle** on the Internet handle in use by the first thread to cancel the operation unblock the first thread.

When an application has finished using the Internet DLL, it should close the **HINTERNET** returned from **InternetOpen** by calling this function.

See Also

InternetOpen, InternetConnect, FtpFindFirstFile, FtpOpenFile, GopherFindFirstFile, HttpOpenRequest

3.3InternetConnect

The **InternetConnect** function opens an FTP, Gopher, or HTTP session for the specified site.

HINTERNET InternetConnect(
 HINTERNET *hInternetSession*,
 LPCTSTR *lpszServerName*,
 INTERNET_PORT *nServerPort,*
 LPCTSTR *lpszUsername*,
 LPCTSTR *lpszPassword*,
 DWORD *dwService*,
 DWORD *dwFlags*,
 DWORD *dwContext*
);

Parameters

hInternetSession

Handle to the current Internet session returned by **InternetOpen**.

lpszServerName

Points to a null-terminated string that specifies the host name of an Internet server. Alternately, the string may contain the IP number of the site in ASCII dotted-decimal format (e.g. “11.0.1.45”).

nServerPort

The TCP/IP port on the server to connect to. If *nServerPort* is INVALID_PORT_NUMBER (0), then the default port for the specified service is used.

lpszUsername

Points to a null-terminated string that specifies the name of the user to log in. If NULL, an appropriate default is used. For the FTP protocol, the default is “anonymous”.

lpszPassword

Points to a null-terminated string that specifies the password to use during login. If both *lpszPassword* and *lpszUsername* are NULL, the default anonymous password is used. In the case of FTP, the default anonymous password is the user’s email name. If *lpszPassword* is NULL (or the empty string) but *lpszUsername* is not NULL, a blank password is used. The following table describes the behavior for the four possible settings of *lpszUsername* and *lpszPassword*:

lpszUsername	*lpszPassword*	Username sent to FTP server	Password sent to FTP server
NULL or “”	NULL or “”	“anonymous”	User’s email name
Non-NULL String	NULL or “”	*lpszUsername*	“”
NULL	Non-NULL String	ERROR	ERROR
Non-NULL String	Non-NULL String	*lpszUsername*	*lpszPassword*

dwService

The type of service to access. May be one of INTERNET_SERVICE_ARCHIE, INTERNET_SERVICE_FTP, INTERNET_SERVICE_GOPHER, or INTERNET_SERVICE_HTTP.

dwFlags

Specifies flags specific to the service used.

Possible values:

dwService	*dwFlags* supported	
INTERNET_SERVICE_FTP	INTERNET_CONNECT_FLAG_PASSIVE	Use passive mode in all data connections for this FTP session

dwContext

An application-defined value that is used to identify the application context for the returned handle in callbacks.

Return Value

If the connection is successfully, a valid handle to the FTP, Gopher, or HTTP session is returned. If the connection attempt fails, NULL is returned and the application may call **GetLastError** to retrieve a specific error code. Applications may use **InternetGetLastResponseInfo** to determine why access to the service was denied.

Remarks

The **InternetConnect** function is required before communicating with any Internet service. For some protocols **InternetConnect** actually establishes a connection with the server, while for others such as Gopher the actual connection is not established until the application requests a specific transaction. The motivation for having a connect function for all protocols, even those which do not use persistent connections, is that it allows the application to communicate common information about several requests with a single function call and allows for future versions of Internet protocols which do not require the high cost of connection establishment for every request the client performs.

For maximum efficiency, applications using the Gopher and HTTP protocols should try to minimize calls to **InternetConnect** and not call it for every transaction requested by the user. One mechanism for doing this is to keep a small cache of handles returned from **InternetConnect** so that when the user makes a request to a previously accessed server, the session handle is still available.

Applications interested in displaying any multi-line text information sent by an FTP server may use **InternetGetLastResponseInfo** to retrieve it.

For FTP connections, if *lpszUsername* is NULL, **InternetConnect** will send the string "anonymous" as the username. If *lpszPassword* is NULL, **InternetConnect** attempts to use the user's email name as the password..

To close the handle returned from **InternetConnect**, the application should call **InternetCloseHandle**. **InternetCloseHandle** will disconnect the client from the server and free all resources associated with the connection.

See Also

InternetCloseHandle

3.4InternetOpenUrl*

The **InternetOpenUrl*** function begins reading an FTP, Gopher, or HTTP URL (Universal Resource Locator).

HINTERNET InternetOpenUrl(
 HINTERNET *hInternetSession*,
 LPCTSTR *lpszUrl*,
 LPCTSTR *lpszHeaders*,
 DWORD *dwHeadersLength*,
 DWORD *dwFlags*,
 DWORD *dwContext*
);

Parameters

hInternetSession

Handle to the current Internet session returned by **InternetOpen**.

lpszUrl

The name of the URL to begin reading. Only URLs beginning with “ftp:”, “gopher:”, or “http:” are supported.

lpszHeaders

Pointer to a string containing the headers to be sent to the http server. See the lpszAdditional parameter to **HttpSendRequest** for more details.

dwHeadersLength

The length (in characters) of the additional headers. If this is -1L and lpszHeaders is non-NULL, then lpszHeaders is assumed to be zero terminated (ASCIIZ) and the length is calculated.

dwFlags

Flags describing how to handle this session. Valid flags are:

dwContext

Application defined value that is passed with the returned handle in any callbacks.

Return Value

If the connection is successfully established, a valid handle to the FTP, Gopher, or HTTP URL is returned. If the open attempt fails, NULL is returned and the application may call **GetLastError** to retrieve a specific error code. Applications may use **InternetGetLastResponseInfo** to determine why access to the service was denied.

Remarks

InternetOpenUrl* is a general function useful for retrieving data over any of the protocols supported by the Internet Extensions for Win32. It is useful when the calling application does not care about accessing the particulars of a protocol but simply wants to retrieve the data corresponding to a URL. **InternetOpenUrl*** parses the URL string, establishes a connection to the server, and gets ready to download the data identified by the URL. The application should use **InternetReadFile** to retrieve the URL data.

Applications need not call **InternetConnect** prior to **InternetOpenUrl***.

Use **InternetCloseHandle** to close the handle returned from **InternetOpenUrl*.** Closing the handle before all the URL data has been read results in aborting the connection.

See Also

HttpSendRequest, **InternetOpen, InternetReadFile, InternetCloseHandle**

3.5InternetReadFile

The **InternetReadFile** function reads data from a handle opened by the **FtpOpenFile**, **GopherOpenFile**, or **HttpOpenRequest** functions.

BOOL InternetReadFile(
 HINTERNET *hFile*,
 LPVOID *lpBuffer*,
 DWORD *dwNumberOfBytesToRead*,
 LPDWORD *lpdwNumberOfBytesRead*
);

Parameters

hFile

A valid handle returned from a previous call to **InternetOpenUrl*, FtpOpenFile, GopherOpenFile**, or **HttpOpenRequest.**.

lpBuffer

Points to the buffer that receives the data read.

dwNumberOfBytesToRead

Specifies the number of bytes to read.

lpNumberOfBytesRead

Points to the number of bytes read by this call. **InternetReadFile** sets this value to zero before doing any work or error checking

Return Value

If the function succeeds, the return value is TRUE; otherwise, it is FALSE. To get extended error information, use the **GetLastError** function, and the **InternetGetLastResponseInfo** function when appropriate.

Remarks

If the return value is TRUE and the number of bytes read is zero, then the transfer has completed and there are no more bytes to be read on the handle. This is analogous to reaching EOF in a local file. The application should then call **InternetCloseHandle**.

The buffer pointed to by *lpBuffer* is not always filled by calls to **InternetReadFile**, as sufficient data may not have arrived from the server to do so. Unless the transfer has completed, however, at least one byte will always be placed in the buffer.

See Also

InternetOpenUrl*, FtpOpenFile, GopherOpenFile, **HttpOpenRequest, InternetCloseHandle**

3.6InternetWriteFile

The **InternetWriteFile** function writes data to an open Internet file.

BOOL InternetWriteFile(
 HINTERNET *hFile*,
 LPVOID *lpBuffer*,
 DWORD *dwNumberOfBytesToWrite*,
 LPDWORD *lpNumberOfBytesWritten*
);

Parameters

hFile

A valid handle returned from a previous call to **FtpOpenFile.**.

lpBuffer

Points to the buffer containing the data to be written to the file.

dwNumberOfBytesToWrite

Specifies the number of bytes to write to the file.

lpNumberOfBytesWritten

Points to the number of bytes written by this call. **InternetWriteFile** sets this value to zero before doing any work or error checking.

Return Value

If the function succeeds, the return value is TRUE; otherwise, it is FALSE. To get extended error information, use the **GetLastError** function, and the **InternetGetLastResponseInfo** function when appropriate.

Remarks

When the application is done sending data, it must call the **InternetCloseHandle** function to end the data transfer.

See Also

FtpOpenFile, InternetCloseHandle

3.7InternetFindNextFile

InternetFindNextFile continues a file search from a previous call to **FtpFindFirstFile, GopherFindFirstFile,** or **ArchieFindFirstFile***.

BOOL InternetFindNextFile(
 HINTERNET *hSearchHandle*,
 LPVOID *lpFindFileData,*
);

Parameters

hSearchHandle

A valid handle returned from one of the **XxxFindFirstFile** functions.

lpFindFileData

Points to a buffer that receives information about the found file or directory. The format of the information placed in the buffer depends on the protocol in use. For example, the FTP protocol returns a WIN32_FIND_DATA structure while the Gopher protocol returns a GOPHER_FIND_DATA structure.

Return Value

If the function succeeds, the return value is TRUE; otherwise, it is FALSE. To get extended error information, use the **GetLastError** function. If no matching files can be found, the **GetLastError** function returns ERROR_NO_MORE_FILES.

See Also

FtpFindFirstFile, GopherFindFirstFile, ArchieFindFirstFile*

3.8InternetQueryOption*

The **InternetQueryOption*** function queries an Internet option on the specified handle.

BOOL
InternetQueryOption(
 HINTERNET *hInternetSession*,
 DWORD *dwOption*,
 LPVOID *lpBuffer*,
 LPDWORD *lpBufferLength*
);

Parameters

hInternetSession

The Internet handle on which to query information.

dwOption

The Internet option to query. The following options are defined:

Value	**Meaning**
INTERNET_OPTION_CALLBACK	Returns the address of the callback function defined for this handle
INTERNET_OPTION_CONNECT_TIMEOUT	A timeout value in milliseconds to use for Internet connection requests. If any connection request takes longer than this timeout then the request is canceled. The default timeout is infinite.
INTERNET_OPTION_CONNECT_RETRIES	A retry count to use for Internet connection requests. If any connection attempt still fails after this many tries then the request is canceled. The default is 5 retries.
INTERNET_OPTION_CONNECT_BACKOFF	A delay value in milliseconds to wait between connection retries.
INTERNET_OPTION_CONTROL_SEND_TIMEOUT	A timeout value in milliseconds to use for non-data (control) Internet send requests. If any non-data send request takes longer than this timeout then the request is canceled. The default timeout is infinite. Currently this value only has meaning for ftp sessions.
INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT	A timeout value in milliseconds to use for non-data (control) Internet receive requests. If any non-data receive request takes longer than this timeout then the request is canceled. The default timeout is infinite. Currently this value only has meaning for ftp sessions.
INTERNET_OPTION_DATA_SEND_TIMEOUT	A timeout value in milliseconds to use for data Internet send requests. If any data send request takes longer than this timeout then the request is canceled. The default timeout is infinite.
INTERNET_OPTION_DATA_RECEIVE_TIMEOUT	A timeout value in milliseconds to use for data Internet receive requests. If any data receive request takes longer than this timeout then the request is canceled. The default timeout is

	infinite.
INTERNET_OPTION_HANDLE_TYPE	Returns the type of the Internet handle passed in. Possible return values include: INTERNET_HANDLE_TYPE_INTERNET INTERNET_HANDLE_TYPE_CONNECT_ARCHIE INTERNET_HANDLE_TYPE_CONNECT_FTP INTERNET_HANDLE_TYPE_CONNECT_GOPHER INTERNET_HANDLE_TYPE_CONNECT_HTTP INTERNET_HANDLE_TYPE_ARCHIE_FIND INTERNET_HANDLE_TYPE_FTP_FIND INTERNET_HANDLE_TYPE_FTP_FIND_HTML INTERNET_HANDLE_TYPE_FTP_FILE INTERNET_HANDLE_TYPE_FTP_FILE_HTML INTERNET_HANDLE_TYPE_GOPHER_FIND INTERNET_HANDLE_TYPE_GOPHER_FIND_HTML INTERNET_HANDLE_TYPE_GOPHER_FILE INTERNET_HANDLE_TYPE_GOPHER_FILE_HTML INTERNET_HANDLE_TYPE_HTTP_REQUEST
INTERNET_OPTION_CONTEXT_VALUE	Returns the context value associated with this Internet handle.

lpBuffer

A buffer which receives the option setting.

lpBufferLength

A pointer to a DWORD containing the length of *lpBuffer*. On return this contains the length of the data placed into *lpBuffer*.

Return Value

TRUE if the operation was successful, or FALSE if there was an error, in which case the application may call **GetLastError** to retrieve a specific error code.

Remarks

See Also

3.9InternetSetOption*

The **InternetSetOption*** function sets an Internet option on the specified handle.

BOOL
InternetSetOption(
 HINTERNET *hInternetSession*,
 DWORD *dwOption*,
 LPVOID *lpBuffer*,
 DWORD *dwBufferLength*
);

Parameters

hInternetSession

The Internet handle on which to set information.

dwOption

The Internet option to set. See **InternetQueryOption** for a list of possible options.

lpBuffer

A buffer which contains the option setting.

dwBufferLength

The length of *lpBuffer*.

Return Value

TRUE if the operation was successful, or FALSE if there was an error, in which case the application may call **GetLastError** to retrieve a specific error code.

Remarks

See Also

3.10InternetGetLastResponseInfo

The **InternetGetLastResponseInfo** function retrieves error text from the last Win32 Internet function which failed.

BOOL
InternetGetLastResponseInfo(
LPDWORD *lpdwError*,
 LPTSTR *lpszBuffer,*
 LPDWORD *lpdwBufferLength*
);

Parameters

lpdwError

Receives an error code pertaining to the operation that failed.

lpszBuffer

A buffer which receives the error text.

lpdwBufferLength

On input, the size of the buffer pointed to by *lpszBuffer.* On output, the size of the string written to the buffer.

Return Value

TRUE if error text was successfully written to the buffer, or FALSE if there was an error, in which case the application may call **GetLastError** to retrieve a specific error code. If the buffer is too small to hold all the error text, **GetLastError** returns ERROR_BUFFER_OVERFLOW and *lpdwBufferLength* contains the minimum buffer size required to return all the error text.

Remarks

The FTP and Gopher protocols return textual information along with most errors. Applications interested in retrieving this extended error information may use the **InternetGetLastResponseInfo** function whenever a **GetLastError**, following an unsuccessful function call, returns ERROR_INTERNET_EXTENDED_ERROR.

The buffer pointed to by *lpszBuffer* must be large enough to hold both the error string and a zero terminator at the end of the string. However, the value returned in *lpdwBufferLength* does not include the terminating zero.

See Also

3.11InternetSetStatusCallback*

The **InternetSetStatusCallback*** function sets up a callback function that the Win32 Internet functions call when progress is made during an operation.

INTERNET_STATUS_CALLBACK InternetSetStatusCallback(
INTERNET_STATUS_CALLBACK *lpInetProc*
);

Parameters

lpInetProc

Pointer to a callback function to call whenever progress is made.

Return Value

Returns the previously defined status callback function for the thread, NULL if the thread had no previously defined status callback, or INVALID_INTERNET_STATUS_CALLBACK if the callback is not valid.

Remarks

Many of the Win32 Internet functions perform several operations on the network. Each of these operations can take some time to complete, and each has the possibility for failure. In some circumstances it is convenient for an application to display processing status during a long-term operation. Setting up an Internet status callback function enables this.

InternetSetStatusCallback* operates on a per-thread basis. This allows different threads within an application process to use different callback functions if desired. However, this also means that applications must call **InternetSetStatusCallback*** for every thread which makes Win32 Internet calls for which status information is required.

After calling **InternetSetStatusCallback***, the callback is called from within any Win32 Internet function performing network operations which may take an extended period of time. The function is defined as follows:

BOOL lpInetProc(
 DWORD *dwContext,*
 DWORD *dwInternetStatus*,
 LPVOID *lpStatusInformation,*
 DWORD *dwStatusInformationLength*
);

dwInternetStatus indicates the operation the Internet DLL is performing. The contents of *lpStatusInformation* depends on the value of *dwInternetStatus*, and *dwStatusInformationLength* indicates the length of the data included in *lpStatusInformation.* The following status values for *dwInternetStatus* are defined:

Value	**Meaning**
INTERNET_STATUS_RESOLVING_NAME	Looking up the IP address of the name contained in *lpStatusInformation.*
INTERNET_STATUS_NAME_RESOLVED	Successfully found the IP address of the name contained in *lpStatusInformation.*
INTERNET_STATUS_CONNECTING_TO_SERVER	Connecting to the socket address

	(SOCKADDR) pointed to by *lpStatusInformation.*
INTERNET_STATUS_CONNECTED_TO_SERVER	Successfully connected to the socket address (SOCKADDR) pointed to by *lpStatusInformation.*
INTERNET_STATUS_SENDING_REQUEST	Sending the information request to the server. *LpStatusInformation* is NULL.
INTERNET_STATUS_ REQUEST_SENT	Successfully sent the information request to the server. *lpStatusInformation* is NULL.
INTERNET_STATUS_RECEIVING_RESPONSE	Waiting for the server to respond to a request. *LpStatusInformation* is NULL.
INTERNET_STATUS_RESPONSE_RECEIVED	Successfully received a response from the server. *LpStatusInformation* is NULL.
INTERNET_STATUS_CLOSING_CONNECTION	Closing the connection to the server. *LpStatusInformation* is NULL.
INTERNET_STATUS_CONNECTION_CLOSED	Successfully closed the connection to the server. *LpStatusInformation* is NULL
INTERNET_STATUS_HANDLE_CREATED	Used by **InternetConnect** to indicate that it has created the new handle. This allows the application to call **InternetCloseHandle** from another thread if the connect is taking too long.

Returning FALSE from the **lpInetProc** cancels the operation and causes the Win32 Internet function to fail.

See Also

InternetCloseHandle, InternetConnect

4Mime APIs

4.1MimeCreateAssociation

The **MimeCreateAssociation** function creates an association between a MIME content type string and the application(s) which support the type.

BOOL MimeCreateAssociation(
 LPCTSTR *lpszContentType,*
 LPCTSTR *lpszExtensions,*
 LPCTSTR *lpszViewer,*
 LPCTSTR *lpszViewerFriendlyName,*
 LPCTSTR *lpszCommandLine,*
 DWORD *dwOptions*
);

Parameters

lpszContentType

The MIME content-type string, for example "application/winword".

lpszExtensions

The file name extension for the viewer, for example ".doc". The caller must include the period in the extension.

lpszViewer

The name of the executable which may be used to view the file type, for example "winword.exe".

lpszViewerFriendlyName

Friendly name for the viewer executable, for example "Word.Document.6".

lpszCommandLine

The command line which should be used to launch the viewer application, for example "C:\WINWORD\WINWORD.EXE /w".

dwOptions

Controls the creation operation. The following flags are possible:

Value	**Meaning**
MIME_OVERWRITE_EXISTING	If the association already exists, overwrite it.
MIME_FAIL_IF_EXISTING	Fail if the association already exists. This is the default behavior.

Return Value

TRUE if the function is successful, FALSE otherwise. Extended error information is available via the **GetLastError** API.

Remarks

MimeCreateAssociation sets up the information required to launch viewers based on the MIME content-type strings returned in many protocols such as HTTP (World Wide Web) and Gopher. This function is similar to the **File-->Associate** functionality of the file manager except that it adds the MIME content-type string. The information is written to the same spot in the registry as with the file manager with the addition of the MIME-specific information.

If MIME_OVERWRITE_EXISTING is specified, then any existing associations for the MIME content-type and file name extension are overwritten. This includes both associations created with **MimeCreateAssociation** and associations created with the file manager.

Viewer applications should call **MimeCreateAssociation** at setup time so that any Internet browsing application can successfully launch them when needed to view retrieved information.

See Also

4.2MimeDeleteAssociation

The **MimeDeleteAssociation** function removes an association created by **MimeCreateAssociation**.

BOOL MimeDeleteAssociation(
 LPCTSTR *lpszContentType*
);

Parameters

lpszContentType

The MIME content-type string as specified to **MimeCreateAssociation**. For example, "application/winword".

Return Value

TRUE if the function is successful, FALSE otherwise. Extended error information is available via the **GetLastError** API.

Remarks

This function only removes associations created by **MimeCreateAssociation**. Associations created in the file manager may not be removed with **MimeDeleteAssociation**.

MimeDeleteAssociation removes all information about an association from the local system.

See Also

4.3MimeGetAssociation

The **MimeGetAssociation** function returns information about MIME content-type and file-extension associations.

BOOL MimeGetAssociation(
 DWORD *dwFilterType,*
 LPCTSTR *lpszFilter,*
 MIME_ENUMERATOR *lpEnumerator*
);

Parameters

dwFilterType

Determines what the *lpszFilter* parameter refers to. May be one of:

Value	**Meaning**
MIME_CONTENT_TYPE	The filter is a MIME content-type string such as "application/winword."
MIME_EXTENSION	The filter is a file name extension like ".doc".
MIME_VIEWER	The filter is a viewer executable name like "winword.exe".
MIME_ALL	Read all associations.

lpszFilter

The filter to use in returning MIME associations.

lpEnumerator

Points to an application-defined callback function. For each association that matches, the MIME_ENUMERATOR callback function is called once.

Return Value

TRUE if the function is successful, FALSE otherwise. Extended error information is available via the **GetLastError** API.

Remarks

The MIME_ENUMERATOR callback function is defined as follows:

BOOL CALLBACK EnumMimeAssociationsProc(
LPCTSTR *lpszContentType,*
 LPCTSTR *lpszExtensions,*
 LPCTSTR *lpszViewer,*
 LPCTSTR *lpszViewerFriendlyName,*
 LPCTSTR *lpszCommandLine*
);

Since a single viewer may handle several types of file extensions and MIME content-type strings, the *lpEnumerator* function may be called multiple times. To stop enumeration, the **EnumMimeAssociationsProc** may return FALSE at any time.

See Also

5FTP APIs

5.1FtpFindFirstFile

The **FtpFindFirstFile** function begins searching the current directory of the given FTP session. File and directory entries are returned to the application in the WIN32_FIND_DATA structure.

HINTERNET FtpFindFirstFile(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszSearchFile*,
 LPWIN32_FIND_DATA *lpFindFileData*,
 DWORD *dwContext*
);

Parameters

hFtpSession

A valid handle to an FTP session returned from **InternetConnect**.

lpszSearchFile

Points to a null-terminated string that specifies a valid directory path or file name for the FTP server's file system.

lpFindFileData

Points to the WIN32_FIND_DATA structure that receives information about the found file or directory.

dwContext

Application-defined value that associates this search with any application data. This would only be used if the calling thread has already set up a Status Callback with **InternetSetStatusCallback**.

Return Value

A valid handle for the request if the directory enumeration was started successfully, or NULL if there was an error, in which case an application may call **GetLastError** to get a specific error code. If no matching files can be found, the **GetLastError** function returns ERROR_NO_MORE_FILES.

Remarks

FtpFindFirstFile is similar to the Win32 function **FindFirstFile**. However, an important difference is that *only one **FtpFindFirstFile** may occur at a time within a given FTP session, and the enumerations are therefore correlated with the FTP session handle instead*. This is because the FTP protocol allows only a single directory enumeration on a session. After calling **FtpFindFirstFile**, and until calling **InternetCloseHandle**, the application may not call **FtpFindFirstFile** again on a given FTP session handle. In that situation, calls to the **FtpFindFirstFile** function will fail with error code ERROR_FTP_TRANSFER_IN_PROGRESS.

After beginning a directory enumeration with **FtpFindFirstFile**, use the **InternetFindNextFile** function to continue the enumeration.

Use the **InternetCloseHandle** function to close the handle returned from **FtpFindFirstFile**. **InternetCloseHandle** the handle before **InternetFindNextFile** fails with ERROR_NO_MORE_FILES results in the directory enumeration being aborted.

Because the FTP protocol provides no standard means of enumerating some common information pertaining to files (such as file creation date, file size, etc.,) may not always be available or correct. In these situations, **FtpFindFirstFile** and **InternetFindNextFile** fill unavailable information in with a "best guess" based on information that is available. For example, creation and last access dates will often be the same as the file's modification date.

This function will enumerate both files and directories.

The application may not call **FtpFindFirstFile** between calls to **FtpOpenFile** and **InternetCloseHandle**.

See Also

FtpOpenFile, InternetCloseHandle, InternetFindNextFile, InternetSetStatusCallback

5.2 FtpGetFile

The **FtpGetFile** function retrieves a file from the FTP server, and stores it under the specified file name, creating a new local file in the process.

BOOL FtpGetFile(
HINTERNET *hFtpSession*,
LPCTSTR *lpszRemoteFile*,
LPCTSTR *lpszNewFile*,
BOOL *fFailIfExists*,
DWORD *dwFlagsAndAttributes*,
DWORD *dwFlags*,
DWORD *dwContext*
);

Parameters

hFtpSession

A valid handle to an FTP session

lpszRemoteFile

A pointer to a null-terminated string that contains the name of the file to retrieve from the remote system.

lpszNewFile

A pointer to a null-terminated string that contains the name of the file to create on the local system.

fFailIfExists

A Boolean flag that indicates whether the function should proceed if a local file of the specified name already exists. If *fFailIfExists* is TRUE and the local file exists, **FtpGetFile** fails.

dwFlagsAndAttributes

Specifies the file attributes and flags for the new file. May be any combination of FILE_ATTRIBUTE_* file attributes. See **CreateFile** for further information on FILE_ATTRIBUTE_* attributes.

dwFlags

Specifies the conditions under which the transfer occurs. May be any of the FTP_TRANSFER_TYPE_* constants. For further information on the FTP_TRANSFER_TYPE_* constants, see **FtpOpenFile.**

dwContext

Application-defined value to associate this search with any application data. This would only be used if the calling thread has already set up a Status Callback with **InternetSetStatusCallback**.

Return Value

TRUE if the file was retrieved successfully, or FALSE if there was an error, in which case an application may call **GetLastError** to get a specific error code.

Remarks

FtpGetFile is a high-level routine that handles all the bookkeeping and overhead associated with reading a file from an FTP server and storing it locally. Applications which want to retrieve file data only or which want to have careful control over the file transfer should use the **FtpOpenFile** and **InternetReadFile** functions.

If the *dwTransferType* specifies FILE_TRANSFER_TYPE_ASCII, translation of the file data will convert control and formatting characters to local equivalents. The default transfer is binary mode, where the file is downloaded in the exactly same format as it is stored on the server.

Both *lpszRemoteFile* and *lpszNewFile* may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpGetFile** translates the directory name separators to the appropriate character before using it.

See Also

5.3FtpPutFile

The **FtpPutFile** function stores a file on the FTP server.

BOOL FtpPutFile(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszLocalFile*,
 LPCTSTR *lpszNewRemoteFile*,
 DWORD *dwTransferType*
);

Parameters

hFtpSession

A valid handle to an FTP session.

lpscLocalFile

A pointer to a null-terminated string that contains the name of the file to send from the local system.

lpszNewRemoteFile

A pointer to a null-terminated string that contains the name of the file to create on the remote system.

dwTransferType

Specifies the conditions under which the transfer occurs. May be any combination of FTP_TRANSFER_* defined constants. For further information on the FTP_TRANSFER_* constants, see **FtpOpenFile**.

Return Value

TRUE if the file was stored successfully, or FALSE if there was an error, in which case an application may call **GetLastError** to get a specific error code.

Remarks

FtpPutFile is a high-level routine that handles all the bookkeeping and overhead associated with reading a file from an FTP server and storing it locally. Applications which want to send file data only or which want to have careful control over the file transfer should use the **FtpOpenFile** and **InternetWriteFile** functions.

If the *dwTransferType* specifies FILE_TRANSFER_TYPE_ASCII, translation of the file data will convert control and formatting characters to local equivalents.

Both *lpszNewRemoteFile* and *lpszLocalFile* may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpPutFile** translates the directory name separators to the appropriate character before using it.

See Also

5.4FtpDeleteFile

The **FtpDeleteFile** function deletes a file stored on the FTP server.

BOOL FtpDeleteFile(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszFile*
);

Parameters

hFtpSession

A valid handle to an FTP session.

lpszFile

A pointer to a null-terminated string that contains the name of the file to delete on the remote system.

Return Value

TRUE if the file was deleted successfully, or FALSE if there was an error, in which case an application may call **GetLastError** to get a specific error code.

Remarks

lpszFile may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpDeleteFile** translates the directory name separators to the appropriate character before using it.

See Also

5.5FtpRenameFile

FtpRenameFile renames a file stored on the FTP server.

BOOL FtpRenameFile(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszExisting*,
 LPCTSTR *lpszNew*
);

Parameters

hFtpSession

A valid handle to an FTP session.

lpszExisting

A pointer to a null-terminated string that contains the name of the file which will have its name on the remote FTP server changed.

lpszNew

A pointer to a null-terminated string that contains the new name for the remote file.

Return Value

TRUE if the file was renamed successfully, or FALSE if there was an error, in which case an application may call **GetLastError** to get a specific error code.

Remarks

lpszExisting and *lpszNew* may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpRenameFile** translates the directory name separators to the appropriate character before using it.

See Also

5.6FtpOpenFile

The **FtpOpenFile** function initiates access to a remote file for either writing or reading.

HINTERNET FtpOpenFile(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszFileName*,
 DWORD *fdwAccess*,
 DWORD *dwTransferType*
);

Parameters

hFtpSession

A valid handle to an FTP session

lpszFileName

A pointer to a null-terminated string that contains the name of the file to access on the remote system.

fdwAccess

Determines how the file will be accessed: either GENERIC_READ or GENERIC_WRITE, but not both

dwTransferType

Specifies the conditions under which subsequent transfers occur. May be any of the FTP_TRANSFER_* constants:

Value	**Meaning**
FTP_TRANSFER_TYPE_ASCII	The file will be transferred using FTP's ASCII (Type "A") transfer method. Control and formatting information will be converted to local equivalents.
FTP_TRANSFER_TYPE_BINARY	The file will be transferred using FTP's Image (Type "I") transfer method. The file is transferred exactly as it exists with no changes. This is the default transfer method.

Return Value

A valid handle for the request if the file was opened successfully, or NULL if there was an error, in which case an application may call **GetLastError** to get a specific error code.

Remarks

The **FtpOpenFile** function should be used in these cases:

• An application has data which it wants to send to an FTP server to be created as a file on the FTP server. The application does not have a local file containing the data. After InternetOpen the file with **FtpOpenFile**, the application will use **InternetWriteFile** to send the FTP file data to the server.

• An application wants to retrieve a file from the server into application-controlled memory rather than writing the file to disk. The application uses **InternetReadFile** after **InternetOpen** the file.

• An application wants a fine level of control over a file transfer. For example, the application may want to display a “thermometer” when downloading a file to give the user an indication that the file transfer is proceeding correctly (or not).

After calling the **FtpOpenFile** function, and until calling **InternetCloseHandle**, the application may only call **InternetReadFile** or **InternetWriteFile, InternetCloseHandle,** the **FtpFindFirstFile** function. Calls to other FTP functions on the same FTP session will fail and set the error code to FTP_ETRANSFER_IN_PROGRESS.

Only one file may be open in a single FTP session. Therefore, no file handle is returned, and the application merely uses the FTP session handle when appropriate.

lpszFile may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash (“\”) or forward slash (“/”) may be used as the directory separator for either name; **FtpOpenFile** translates the directory name separators to the appropriate character before using it.

Use the **InternetCloseHandle** function to handle returned from **FtpOpenFile**. InternetCloseHandle the handle before all the data has been transferred results in the transfer being aborted.

See Also

5.7FtpCreateDirectory

The **FtpCreateDirectory** function creates a new directory on the FTP server.

BOOL FtpCreateDirectory(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszDirectory*
);

Parameters

hFtpSession

A valid handle to an FTP session.

lpszDirectory

A pointer to a null-terminated string that contains the name of the directory to create on the remote system. This may be either a fully qualified path name or a name relative to the current directory.

Return Value

If the function succeeds, **FtpCreateDirectory** returns TRUE. Otherwise, an error occurred and the application may use **GetLastError** to retrieve a specific error code. If the error code indicates that the FTP server denied the request to create a directory, **InternetGetLastResponseInfo** may be useful in determining why.

Remarks

Applications should use **FtpGetCurrentDirectory** to determine the remote site's current working directory, rather than assuming that the remote system uses a hierarchical naming scheme for directories.

lpszDirectory may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpCreateDirectory** translates the directory name separators to the appropriate character before using it.

See Also

5.8FtpRemoveDirectory

The **FtpRemoveDirectory** function removes the specified directory on the FTP server.

BOOL FtpRemoveDirectory(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszDirectory*
);

Parameters

hFtpSession

A valid handle to an FTP session.

lpszDirectory

A pointer to a null-terminated string that contains the name of the directory to remove on the remote system. This may be either a fully qualified path name or a name relative to the current directory.

Return Value

If the function succeeds, the return value is TRUE. Otherwise, an error occurred and the application may use **GetLastError** to retrieve a specific error code. If the error code indicates that the FTP server denied the request to remove a directory, **InternetGetLastResponseInfo** may be useful in determining why.

Remarks

Applications should use **FtpGetCurrentDirectory** to determine the remote site's current working directory, rather than assuming that the remote system uses a hierarchical naming scheme for directories.

lpszDirectory may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpRemoveDirectory** translates the directory name separators to the appropriate character before using it.

See Also

5.9FtpSetCurrentDirectory

The **FtpSetCurrentDirectory** function changes to a different working directory on the FTP server.

BOOL FtpSetCurrentDirectory(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszDirectory*
);

Parameters

hFtpSession

A valid handle to an FTP session.

lpszDirectory

A pointer to a null-terminated string that contains the name of the directory to change to on the remote system. This may be either a fully qualified path name or a name relative to the current directory.

Return Value

If the function succeeds, the return value is TRUE. Otherwise, an error occurred and the application may use **GetLastError** to retrieve a specific error code. If the error code indicates that the FTP server denied the request to change to a directory, **InternetGetLastResponseInfo** may be useful in determining why.

Remarks

Applications should use **FtpGetCurrentDirectory** to determine the remote site's current working directory, rather than assuming that the remote system uses a hierarchical naming scheme for directories.

lpszDirectory may be either partially qualified file names relative to the current directory or fully qualified. Either a backslash ("\") or forward slash ("/") may be used as the directory separator for either name; **FtpSetCurrentDirectory** translates the directory name separators to the appropriate character before using it.

See Also

5.10FtpGetCurrentDirectory

The **FtpGetCurrentDirectory** function retrieves the current directory for the specified FTP session.

BOOL FtpGetCurrentDirectory(
 HINTERNET *hFtpSession*,
 LPCTSTR *lpszCurrentDirectory,*
 LPDWORD *lpdwCurrentDirectoty*

);

Parameters

hFtpSession

A valid handle to an FTP session.

lpszCurrentDirectory

Points to the buffer for the current directory string. This null-terminated string specifies the absolute path to the current directory.

lpdwCurrentDirectory

Specifies the length, in characters, of the buffer for the current directory string. The buffer length must include room for a terminating null character. Using a length of MAX_PATH will be sufficient for all pathnames.

Return Value

If the function succeeds, the return value is TRUE. Otherwise, an error occurred and the application may use **GetLastError** to retrieve a specific error code. If the error code indicates that the FTP server denied the request to change to a directory, **InternetGetLastResponseInfo** may be useful in determining why.

Remarks

If the *lpszCurrentDirectory* buffer is not large enough, then on return *lpdwCurrentDirectory* contains the number of bytes requires to retrieve the full current directory name.

See Also

5.11FtpCommand

The **FtpCommand** function issues an arbitrary command on the FTP server referred to by the session handle.

BOOL FtpCommand(
 HINTERNET *hFtpSession*,
 BOOL *fExpectResponse*,
 DWORD *dwTransferType*,
 LPCTSTR *lpszCommand*
);

Parameters

hFtpSession

A valid handle to an FTP session.

fExpectResponse

A Boolean value indicating whether the command is expected to cause the FTP server to open a data connection on which to send a reply. If TRUE, **FtpCommand** creates a connection endpoint and negotiates for a connection with the FTP server. The application must use **InternetReadFile** and **InternetCloseHandle** to read from and close this data connection.

dwTransferType

One of the FTP_TRANSFER_TYPE_* constants. If fExpectResponse is TRUE, this value governs the type of connection constructed with the FTP server.

lpszCommand

A pointer to a null-terminated format string containing the command to send to the FTP server.

Return Value

If the function succeeds, the return value is TRUE. Otherwise, an error occurred and the application may use **GetLastError** to retrieve a specific error code. If the error code indicates that the FTP server denied the request to change to a directory, **InternetGetLastResponseInfo** may be useful in determining why

Remarks

The application must use **InternetGetLastResponseInfo** to determine the FTP server's response to the command.

The **FtpCommand** function ensures that the text sent contains only ASCII printable characters and filters the string appropriately, adding the necessary end-of-line characters.

If *fExpectResponse* is TRUE, **FtpCommand** will issue the appropriate "PORT" or "PASV" command from the FTP protocol.

See Also

6Gopher APIs

6.1GopherCreateLocator

The **GopherCreateLocator** function creates a Gopher or Gopher+ locator string from its component parts.

BOOL GopherCreateLocator(
 LPCTSTR *lpszHost*,
 INTERNET_PORT *nPort*,
 LPCTSTR *lpszDisplayString*,
 LPCTSTR *lpszSelector*,
 DWORD *dwGopherType*,
 LPCSTR *lpszLocator,*
 LPDWORD *lpdwBufferLength*
);

Parameters

lpszHost

String identifying the name of the host, or a dotted-decimal IP address like “198.105.232.1”.

nPort

Port on which the Gopher server at *Host* lives, in host-byte order. If *nPort* is INVALID_PORT_NUMBER, then the default gopher port is read from the \etc\services file..

lpszDisplayString

The Gopher document or directory to be displayed. *lpszDisplayString* may be NULL, in which case the default directory for the Gopher server will be returned.

lpszSelector

The selector string to be sent to the gopher server in order to retrieve information. May be NULL.

dwGopherType

Specifying whether *lpszSelector* refers to a directory or document, and whether the request is Gopher+ or Gopher. See GOPHER_FIND_DATA *Attributes*.

lpszLocator

Pointer to the buffer where the locator string will be returned. If *lpszLocator* is NULL, *lpdwBufferLength* receives the needed buffer length but no other processing occurs.

lpdwBufferLength

On input, the length of *lpszLocator*. On output, the number of bytes written to *lpszLocator*, or, if the *lpszLocator* buffer is too small, the number of bytes required to form the locator successfully.

Return Value

TRUE if success, else FALSE, in which case more information will be available via **GetLastError** and **InternetGetLastResponseInfo**.

Remarks

In order to retrieve information from a Gopher server, an application must first get a Gopher “locator” from the Gopher server. The locator, which the application should treat as an opaque token, is typically used for calls to the **GopherFindFirstFile** function to retrieve a specific piece of information.

See Also

6.2GopherGetLocatorType

The **GopherGetLocatorType** function parses a Gopher locator and determines its attributes.

BOOL GopherGetLocatorType(
 LPCTSTR *lpszLocator,*
 LPDWORD *lpdwGopherType*
);

Parameters

lpszLocator

A Gopher locator string to parse.

lpdwGopherType

Receives the type of the locator. This is a bitmask of any of the following:

Value	**Meaning**
GOPHER_TYPE_TEXT_FILE	The item is an ASCII text file.
GOPHER_TYPE_DIRECTORY	The item is a directory of additional Gopher items.
GOPHER_TYPE_CSO	The item is a CSO phone book server.
GOPHER_TYPE_ERROR	
GOPHER_TYPE_MAC_BINHEX	The item is a Macintosh file in BINHEX format.
GOPHER_TYPE_DOS_ARCHIVE	The file is a DOS archive file.
GOPHER_TYPE_UNIX_UUENCODED	The item is a UUENCODED file.
GOPHER_TYPE_INDEX_SERVER	The item refers to an index server.
GOPHER_TYPE_TELNET	The item refers to a telnet server.
GOPHER_TYPE_BINARY	The item is a binary file.
GOPHER_TYPE_REDUNDANT	
GOPHER_TYPE_TN3270	The item is a TN3270 server.
GOPHER_TYPE_GIF	The item is a GIF graphics file.
GOPHER_TYPE_IMAGE	The item is an image file.
GOPHER_TYPE_BITMAP	The item is a bitmap file.
GOPHER_TYPE_MOVIE	The item is a movie file.
GOPHER_TYPE_SOUND	The item is a sound file.
GOPHER_TYPE_GOPHER_PLUS	The item is a Gopher+ item

.
Remarks

GopherGetLocatorType returns information about the item referenced by a Gopher locator. Note that it is possible for multiple attributes to be set on a file. For example, both GOPHER_TYPE_TEXT_FILE and GOPHER_TYPE_GOPHER_PLUS are set for a text file stored on a Gopher+ server..

See Also

6.3GopherFindFirstFile

Given a Gopher locator and some search criteria, the **GopherFindFirstFile** function creates a session with the server and locate the requested documents, binary files, index servers or directory trees.

HINTERNET GopherFindFirstFile(
 HINTERNET *hGopherSession*,
 LPCTSTR *lpszLocator*,
 LPCTSTR *lpszSearchString*,
 LPGOPHER_FIND_DATA *lpGopherFindData*
);

Parameters

hGopherSession

Handle to a Gopher session returned by **InternetConnect.**

lpszLocator

Name of the item to locate. This parameter may be any of the following:

- a Gopher locator returned in the *lpGopherFindData.Locator* field from a previous call to this function or **InternetFindNextFile**;
- a NULL pointer or zero-length string indicating that we are returning the top-most information from a Gopher server;
- a locator created by the **GopherCreateLocator** function.

lpszSearchString

If this request is to an index server, *lpszSearchString* specifies the strings for which to search. If the request is not to an index server, *lpszSearchString* should be NULL.

lpGopherFindData

Pointer to application-supplied buffer which will be filled with a GOPHER_FIND_DATA structure.

Return Value

If successful a valid search handle, else NULL, in which case more information will be available from the **GetLastError** and **InternetGetLastResponseInfo** functions.

Comments

GopherFindFirstFile closely resembles the Win32 function **FindFirstFile**. It creates a connection with a Gopher server and then returns a single structure containing information about the first Gopher object referenced by the locator string.

After calling **GopherFindFirstFile**, to get the first Gopher object in an enumeration, use the **InternetFindNextFile** to retrieve subsequent Gopher objects.

Use the **InternetCloseHandle** function to close the handle returned from **GopherFindFirstFile.** If there are any pending operations described by the handle when the application calls **InternetCloseHandle**, they are canceled or marked close-pending. Any open sessions will be terminated, and any data waiting to be indicated to the caller will be discarded. Any allocated buffers will be freed.

See Also

InternetFindNextFile

6.4 GopherOpenFile

The **GopherOpenFile** function starts reading a Gopher data file from a Gopher server.

HINTERNET GopherOpenFile(
 HINTERNET *hGopherSession*,
 LPCTSTR *lpszLocator*,
 LPCTSTR *lpszView*
);

Parameters

hInternetSession

Handle to a Gopher session returned by **InternetConnect.**

lpszLocator

String identifying the file to "open". Typically, this locator will have been returned from a call to **GopherFindFirstFile** or **InternetFindNextFile**. Since the Gopher protocol has no concept of a "current directory," the locator is always fully qualified.

lpszView

If several views of the file exist at the server, this parameter describes which file view to open. If *lpszView* is NULL, the default file view is used.

Return Value

NULL if the file cannot be opened. Use **GetLastError** or **InternetGetLastResponseInfo** to determine the cause of the error.

Remarks

"Opens" a file at a Gopher server. Since a file cannot be actually opened or locked at a server, this call simply associates location information with a handle that can be used for file-based operations, such as **InternetReadFile** or **GopherGetAttribute**.

Use the **InternetCloseHandle** function to close the handle returned from **GopherOpenFile.** If there are any pending operations described by the handle when the application calls **InternetCloseHandle**, they are canceled or marked close-pending. Any open sessions will be terminated, and any data waiting to be indicated to the caller will be discarded. Any allocated buffers will be freed.

See Also

InternetReadFile, **GopherGetAttribute**

6.5GopherGetAttribute

The **GopherGetAttribute** function allows an application to retrieve specific attribute information from the server.

BOOL GopherGetAttribute(
HINTERNET *hGopherSession*,
LPCTSTR *lpszLocator*,
LPCTSTR *lpszInformation*,
LPBYTE *lpBuffer*,
DWORD *dwBufferLength*,
LPDWORD *lpdwBytesReturned,*
GOPHER_ATTRIBUTE_ENUMERATOR *lpfnEnumerator*
);

Parameters

hInternetSession

Handle to an Gopher session returned by **InternetConnect.**

lpszLocator

String identifying the item at Gopher server about which to return attribute information.

lpszInformation

Pointer to space-delimited string specifying names of attributes to return.

lpBuffer

Pointer to user-supplied buffer into which attribute information is retrieved.

dwBufferLength

Size of *lpBuffer* in bytes.

lpdwBytesReturned

Number of bytes read into *lpBuffer*.

lpfnEnumerator

Points to a callback enumeration function which is called for each attribute of the locator. This parameter is optional; if NULL, then all the Gopher attribute information is placed into *lpBuffer*. If *lpfnEnumerator* is specified, then the callback function is called once for each attribute of the object. The callback function is passed a pointer to a single GOPHER_ATTRIBUTE_TYPE structure for each call. The enumeration callback allows the application to avoid having to parse the Gopher attribute information.

Return Value

TRUE if request is satisfied, else FALSE. Use **GetLastError** or **InternetGetLastResponseInfo** to discover more information about the error.

Remarks

Typically, applications will make this call following a **GopherFindFirstFile** or **InternetFindNextFile** call which will therefore request cached information.

The GOPHER_ATTRIBUTE_ENUMERATOR function has the following syntax:

BOOL GopherAttributeEnumerator(
 LPGOPHER_ATTRIBUTE_TYPE *lpAttributeInformation*,
 DWORD *dwError*
);

lpAttributeInformation points to a buffer which contains a single GOPHER_ATTRIBUTE_TYPE structure. The *lpBuffer* parameter to **GopherGetAttributes** is used for storing this structure. *dwError* is NO_ERROR (zero) if the attribute was parsed and written to the buffer successfully, or an error code if a problem was encountered. Returning FALSE from this function stops the enumeration immediately.

See Also

7HTTP APIs

7.1HttpOpenRequest

The **HttpOpenRequest** function opens an HTTP request handle.

HINTERNET HttpOpenRequest(
 HINTERNET *hInternetSession*,
 LPCTSTR *lpszVerb*,
 LPCTSTR *lpszObjectName*,
 LPCTSTR *lpszVersion*,
 LPCTSTR *lpszReferer*,
 LPCTSTR FAR * *lplpszAcceptTypes*,
);

Parameters

hInternetSession

Handle to an HTTP session returned by **InternetConnect.**

lpszVerb

The verb to use in the request.

lpszObjectName

The target object of the specified verb. This is typically a file name, an executable module, or a search specifier.

lpszVersion

The HTTP version for the request. The high word defines the minor version and the low word defines the major version. Must be HTTP_VERSION in version 1.0 of the Internet Extensions for Win32.

lpszReferer

Specifies the address (URI) of the document from which the URI in the request (*lpszObjectName*) was obtained. May be NULL, in which case no referer is specified.

lplpszAcceptTypes

Points to a NULL-terminated array of LPCTSTR pointers to content-types accepted by the client. This value may be NULL in which case no accept types are. Servers interpret a lack of accept types to indicate that the client only accepts documents of type “text/*”, that is, only text documents, not pictures or other binary files.

Return Value

A valid (non-NULL) HTTP request handle if successful, NULL otherwise. Extended error information is available via the **GetLastError** API.

Remarks

This function creates a new HTTP request handle and stores the specified parameters in that handle.

An HTTP request handle encapsulates a request to be sent to an HTTP server. The HTTP request handle contains all RFC822/MIME/HTTP headers to be sent as part of the request.

Use the **InternetCloseHandle** function to close the handle returned by **HttpOpenRequest.** InternetCloseHandle the handle cancels all outstanding I/O on the handle.

The *lpszCallerApplicationName* parameter to **InternetOpen** is used as the referrer for the HTTP request.

See Also

InternetOpen, **InternetCloseHandle**, **HttpSendRequest**, **HttpAddRequestHeaders**, **InternetReadFile**, **HttpQueryInfo**.

7.2HttpAddRequestHeaders

The **HttpAddRequestHeaders** function adds one or more HTTP request headers to the HTTP request handle.

BOOL HttpAddRequestHeaders(
 HINTERNET *hHttpRequest*,
 LPCTSTR *lpszHeaders*,
 DWORD *dwHeadersLength*
);

Parameters

hHttpRequest

An open HTTP request handle returned by **HttpOpenRequest**.

lpszHeaders

The headers to append to the request. Each header must be terminated by a CR/LF pair.

dwHeadersLength

The length (in characters) of *lpszHeaders*. If this is -1L, then *lpszHeaders* is assumed to be zero terminated (ASCIIZ) and the length is computed.

Remarks

This function appends additional "free format" headers to the HTTP request handle. This API is intended for use by sophisticated clients that need detailed control over the exact request sent to the HTTP server.

Return Value

TRUE if the function is successful, FALSE otherwise. Extended error information is available via the **GetLastError()** API.

See Also

HttpOpenRequest, **HttpSendRequest**.

7.3HttpSendRequest

The **HttpSendRequest** function sends the specified request to the HTTP server.

BOOL HttpSendRequest(
 HINTERNET *hHttpRequest*,
 LPCTSTR *lpszHeaders*,
 DWORD *dwHeadersLength*,
 LPVOID *lpOptional*,
 DWORD *dwOptionalLength*
);

Parameters

hHttpRequest

An open HTTP request handle returned by **HttpOpenRequest**.

lpszAdditional

Additional headers to be appended to the request. This may be NULL if there are no additional headers to append.

dwAdditionalLength

The length (in characters) of the additional headers. If this is -1L and lpszAdditional is non-NULL, then lpszAdditional is assumed to be zero terminated (ASCIIZ) and the length is calculated.

lpOptional

Any optional data to send immediately after the request headers. This is typically used for POST and PUT operations. This may be NULL if there is no optional data to send.

dwOptionalLength

The length (in BYTEs) of the optional data. This may be zero if there is no optional data to send.

Remarks

This function sends the specified request to the HTTP server. This function allows the client to specify additional RFC822/MIME/HTTP headers to send along with the request.

This function also allows the client to specify “optional” data to send to the HTTP server immediately following the request headers. This feature is typically used for “write” operations such as PUT and POST.

After the request is sent, this function reads the status code and response headers from the HTTP server. These headers are maintained internally to the request handle, and are available to client applications via the **HttpQueryInfo** API.

Return Value

TRUE if the function is successful, FALSE otherwise. Extended error information is available via the **GetLastError** API.

See Also

HttpOpenRequest, **InternetReadFile**, **HttpQueryInfo**.

7.4HttpQueryInfo

The **HttpQueryInfo** function queries information about an HTTP request.

BOOL HttpQueryInfo(
 HINTERNET *hHttpRequest***,**
 DWORD *dwInfoLevel***,**
 LPVOID *lpBuffer***,**
 LPDWORD *lpdwBufferLength*
);

Parameters

hHttpRequest

An open HTTP request handle returned by **HttpOpenRequest()**.

dwInfoLevel

One of the HTTP_QUERY_* values indicating the attribute to query.

lpBuffer

Pointer to the buffer that will receive the information.

lpdwBufferLength

On entry, points to a value containing the length (in BYTEs) of the data buffer. On exit, points to a value containing the length (in BYTEs) of the information written to the buffer.

Remarks

Most info levels return simple values. HTTP_QUERY_CONTENT_LENGTH, for example, returns an ASCII string representing the size (in BYTEs) of the returned object.

HTTP_QUERY_RAW_HEADERS is not so simple. This info level allows a client to access the raw RFC822/MIME/HTTP headers returned by the HTTP server. On entry to this API, *lpBuffer* points to the field name of the header to retrieve (i.e. "Accept:"). If lpBuffer points to an empty string (i.e. ""), then *all* headers are returned.

Return Value

TRUE if the function is successful, FALSE otherwise. Extended error information is available via the **GetLastError** API.

See Also

HttpOpenRequest.

8Archie APIs

8.1ArchieFindFirstFile*

The **ArchieFindFirstFile*** function processes an Archie query on given set of hosts, searching for instances specified by search string and returns a handle to the result.

HINTERNET ArchieFindFirstFile(
 HINTERNET *hArchieSession,*
 LPCTSTR **lplpszHosts,*
 LPCTSTR *lpszSearchString,*
 DWORD *dwMaxHits***,**
 DWORD *dwOffset,*
 DWORD *dwPriority,*
 ARCHIE_SEARCH_TYPE *SearchType,*
 LPARCHIE_FIND_DATA *lpFindData,*
 LPDWORD *lpdwNumberFound*,
 DWORD *dwContext*
);

Parameters

hArchieSession

A handle to an Internet session returned from **InternetOpen.**

lplpszHosts

Pointer to an array of null terminated character strings. This list is to be terminated with a string with two nulls in it. These strings contain the name of archie hosts to use for conducting the search. If *plpszHosts* is NULL, all the default Archie servers are used for the search.

lpszSearchString

Points to the search string. The search string may be a plain string or may contain standard regular expression wildcard symbols (wildcards like '*', '.' and optional parentheses). This is used in conjunction with *SearchType*..

dwMaxHits

Specifies the maximum number of entries to retrieve per archie server among all matched entries. The parameter is used with *dwOffset* specified to index into the list of matched entries. This limits the number of entries pulled for certain worst cases like regular expression or substring search with a single character. If unspecified (negative), a default of 30 is assumed.

dwOffset

Offset in the list of entries on each server from where to start collecting matched entries. The default is offset 0.

dwPriority

Gives the Archie server an indication of the priority to place on the search. It can be one of ARCHIE_PRIORITY_LOW, ARCHIE_PRIORITY _MEDIUM or ARCHIE_PRIORITY _HIGH.

SearchType

Specifies type of search. The following values are possible:

Value	**Meaning**
ArchieExact	Matches all filenames that are exactly same as search string.
ArchieRegexp	Matches filenames using regular expression given in search string.
ArchieExactOrRegexp	Matches filenames exact or based on regular expression.
ArchieSubstring	Matches all filenames with given search string as a substring.
ArchieExactOrSubstring	Matches all filenames exact or substring based.
ArchieCaseSubstring	Matches all filenames like *ArchiestSubstring* but is case sensitive.
ArchieExactOrCaseSubstring	Matches all filenames exact or like *ArchiestSubstring* but case sensitive.

lpFindData

Points to an ARCHIE_FIND_DATA structure that is filled with the first entry from the query.

lpdwNumberFound

This parameter on successful return contains the number of matches found for given keyword.

dwContext

Application-defined value that associates this search with any application data. This would only be used if the calling thread has already set up a Status Callback with **InternetSetStatusCallback**.

Return Value
On success, this function returns a handle to the archie client result. Otherwise returns NULL. Applications should use **GetLastError** to retrieve the error code.

Remarks

This function is a blocking call. It performs a vectored search across the supplied list of hosts or all the hosts for given search string with given search criteria. It collects the responses from the hosts, parses them and constructs a list of internal objects containing the location and attribute information for files. The application can use the returned handle to access the file information. Threads of a single process do not share any data across calls to **ArchieFindFirstFile***.

Use the **InternetFindNextFile** to continue enumerating the information returned from the Archie servers.

Use the **InternetCloseHandle** function to close the handle returned from **ArchieFindFirstFile***. **InternetCloseHandle** the handle frees any resources allocated for the search.

See Also

InternetCloseHandle, InternetFindNextFile, InternetSetStatusCallback

9Structure Definitions

9.1GOPHER_FIND_DATA

```
typedef struct {
    TCHAR cDisplayString[MAX_GOPHER_DISPLAY_TEXT + 1];
    DWORD dwGopherType;
    DWORD dwSizeLow;
    DWORD dwSizeHigh;
    FILETIME ftLastModificationTime;
    TCHAR cLocator[MAX_GOPHER_LOCATOR_LENGTH + 1];
} GOPHER_FIND_DATA, FAR *LPGOPHER_FIND_DATA;
```

The **GopherFindFirstFile** and **InternetFindNextFile** functions return GOPHER_FIND_DATA structures. The fields of GOPHER_FIND_DATA have the following meanings:

Members

cDisplayString

A friendly name that identifies the object. Display this string to the user for selection.

dwGopherType

A mask of flags which describe the item returned.

dwFileSizeLow

The low 32 bits of the file size.

dwFileSizeHigh

The high 32 bits of the file size.

ftLastModificationTime

The time the file was last modified.

cLocator

A locator that identifies the file. Pass this locator to GopherOpenFile or GopherFindFirstFile.

See Also

GopherFindFirstFile

9.2GOPHER_ATTRIBUTE_TYPE

```
typedef struct {
    LPCTSTR lpszCategoryName
    DWORD dwAttributeName
    union {
        GOPHER_ADMIN_ATTRIBUTE Admin;
        GOPHER_MOD_DATE_ATTRIBUTE ModDate;
        GOPHER_SCORE_ATTRIBUTE Score;
        GOPHER_SCORE_RANGE_ATTRIBUTE ScoreRange;
        GOPHER_SITE_ATTRIBUTE Site;
        GOPHER_ORGANIZATION_ATTRIBUTE Organization;
        GOPHER_LOCATION_ATTRIBUTE Location;
        GOPHER_GEOGRAPHICAL_LOCATION_ATTRIBUTE GeographicalLocation;
        GOPHER_TIMEZONE_ATTRIBUTE TimeZone;
        GOPHER_PROVIDER_ATTRIBUTE Provider;
        GOPHER_VERSION_ATTRIBUTE Version;
        GOPHER_ABSTRACT_ATTRIBUTE Abstract;
        GOPHER_VIEW_ATTRIBUTE View;
        GOPHER_VERONICA_ATTRIBUTE Veronica;
        GOPHER_UNKNOWN_ATTRIBUTE Unknown;
    } AttributeType;
} GOPHER_ATTRIBUTE_TYPE, *LPGOPHER_ATTRIBUTE_TYPE;
```

The GOPHER_ATTRIBUTE_TYPE structure contains the relevant information for a single Gopher attribute for an object.

Members

lpszCategoryName

The Gopher name for the attribute, for example “+ADMIN”.

dwAttribute

Defines the structure that is contained in the **AttributeType** member of GOPHER_ATTRIBUTE_TYPE, for example GOPHER_ATTRIBUTE_ADMIN.

AttributeType

The actual setting for the Gopher attribute. The specific value of **AttributeType** depends on the **dwAttribute** member. The definitions of the various attribute structures is available in wininet.h.

See Also

GopherGetAttributes

9.3ARCHIE_FIND_DATA

```
typedef struct {
    DWORD dwAttributes;
    DWORD dwSize;
    FILETIME ftLastFileModTime;
    FILETIME ftLastHostModTime;
    DWORD dwTransferType;
    ARCHIE_ACCESS_METHOD AccessMethod;
    TCHAR cHostType[ARCHIE_MAX_HOST_TYPE_LENGTH];
    TCHAR cHostName[ARCHIE_MAX_HOST_NAME_LENGTH];
    TCHAR cHostAddr[ARCHIE_MAX_HOST_ADDR_LENGTH];
    TCHAR cFileName[ARCHIE_MAX_PATH_LENGTH];
    TCHAR cUserName[ARCHIE_MAX_USERNAME_LENGTH];
    TCHAR cPassword[ARCHIE_MAX_PASSWORD_LENGTH];
} ARCHIE_FIND_DATA, *LPARCHIE_FIND_DATA;
```

The ARCHIE_FIND_DATA structure is the information returned from all Archie requests. The fields are defined as follows:

Members

dwAttributes

The file attributes, defined in Win32 file attribute bits.

dwSize

The file size in bytes.

ftLastFileModTime

The time the file was last modified.

ftLastHostModTime

The time the file was last modified.

TransferType

The type of transfer to use when retrieving the file. The following transfer types are possible:

ArchieTransferUnknown	Unknown transfer type
ArchieTransferBinary	Binary transfer type
ArchieTransferAscii	Text-mode (Ascii) transfer type

AccessMethod

The method to use to retrieve the file. The following access methods are possible:

ArchieError	
ArchieAftp	Anonymous FTP
ArchieFtp	Regular FTP
ArchieNfs	NFS File System
ArchieKnfs	Kerberized NFS
ArchiePfs	Andrew File System

cHostType

A string that identifies the type of host that has the file.

cHostName

The name of the host that has the file.

cHostAddr

The host’s Internet address.

cFileName

A fully qualified path to the file.

cUserName

A user name to use when the file must be accessed with non-anonymous FTP.

cPassword

A password to use when using non-anonymous FTP to access the file.

See Also

ArchieFindFirstFile