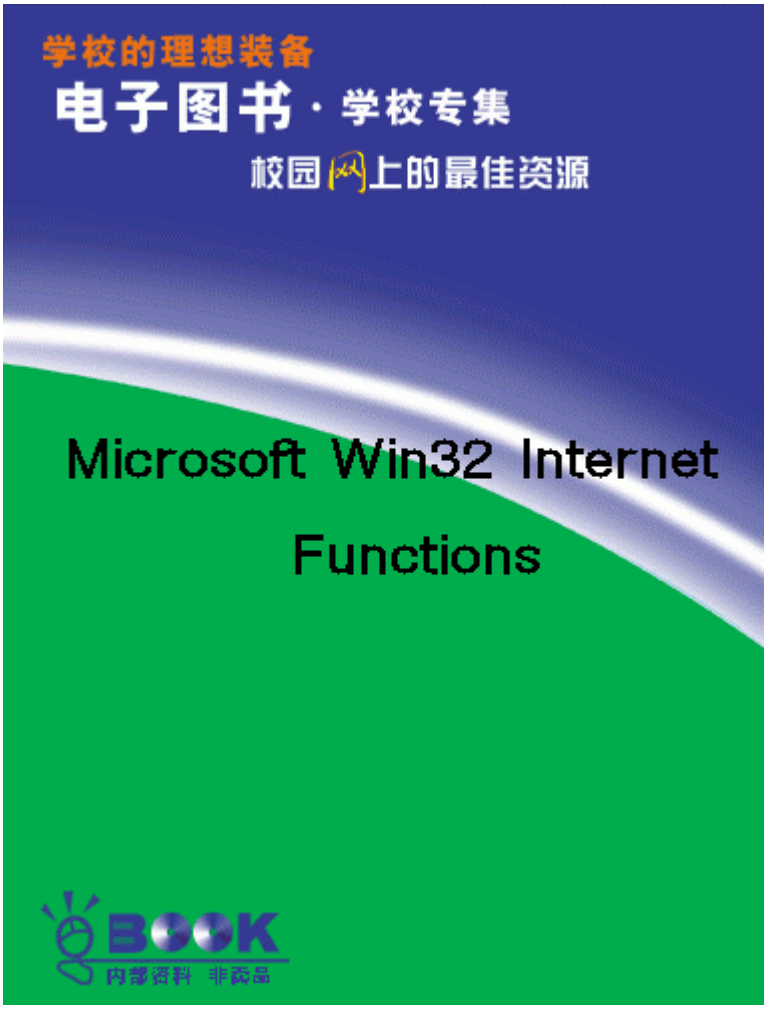
学校的理想装备
电子图书·学校专集
校园网上的最佳资源
Microsoft Win32 Internet
Functions
BOOK
内部资料 非卖品

# Microsoft Win32 Internet Functions

## Purpose and Background

The Microsoft® Win32® Internet functions provide Win32-based applications with easy access to common Internet protocols. These functions abstract the Internet's Gopher, FTP (file transfer protocol), and HTTP (hypertext transfer protocol) protocols into a high-level application programming interface (API) that is familiar to independent software vendors (ISVs) and software developers, and that provides a fast and straightforward path to making applications Internet-aware.

---

**Note** Initially, the Win32 Internet functions will be shipped as redistributables independent of operating systems through the Microsoft Developer Network (MSDN), CompuServe, and the Internet. ISVs can redistribute Wininet.dll with their applications, following the model of Win32s®. In the future, the functions described in this documentation will be folded into all Microsoft operating systems. The Win32 Internet functions are exported from Wininet.dll.

---

The Win32 Internet functions facilitate access to the Internet by:

- Eliminating the need to embed knowledge of TCP/IP and Windows® Sockets.

  By converting the Internet protocols into task-oriented functions, application developers do not need to write Windows Sockets code or be familiar with the TCP/IP protocol.
- Eliminating the need to embed knowledge of Internet protocols.

  While the concepts supported by the Internet protocols, such as FTP and HTTP, are simple, the actual implementation of these protocols can be complex. For example, FTP servers return ASCII text file directory listings, but parsing these listings requires specific knowledge of the format returned by each FTP server. By encapsulating this functionality within the Internet functions, directory parsing is solved once for all applications using the FTP protocol. This provides consistent behavior across applications.
- Providing a constant set of functions in an environment of rapidly changing and evolving protocols.

  Keeping pace with the changes in Internet protocols is a challenge when writing applications. With the set of functions designed to remain constant, application developers no longer need to update their applications every time the underlying protocol changes. Now, only

Wininet.dll needs to be changed. In addition, advanced protocols, such as HTTP version 1.1, can also be implemented without changing applications.

- Following Win32 function standards.

  The Win32 Internet functions are similar to the traditional Win32 functions in the way they treat elements such as buffer management and error returns. Application developers familiar with the Win32 function set will find that the Win32 Internet functions return information in a familiar format. Furthermore, application developers will find it easy to use the returned information in other Win32 functions.

- Providing full access to Internet protocols.

  Occasionally, applications need to access extended features of the Internet protocols. The Win32 Internet functions help provide this access.

- Enabling high-performance, multithreaded Internet applications.

  The Win32 Internet functions are fully "reentrant" and multithread safe. Multithreaded applications can make simultaneous calls into the functions from different threads without adverse effects. The Internet functions themselves complete any necessary synchronization.

- Having persistent caching support built in.

  The Win32 Internet functions provide persistent caching for all protocols, so the application developer can concentrate on obtaining the data and not worrying about managing the cache. For more information about how Win32 Internet functions use the cache functions to get proper Web behavior, see *Persistent URL Cache Functions*.

The Win32 Internet functions are intended to make Internet client applications easier to write; they are not intended to facilitate writing Internet servers. This is because servers must be able to control how the protocol is accessed and how I/O is performed in order to achieve the high performance necessary for high-traffic servers. In addition, the Win32 Internet functions are not intended to solve the general issue of access to mail and news servers.

# verview of the Win32 Internet Functions

The following table summarizes the Win32 Internet functions. Each function indicates any functions that it is dependent on. A dependent function can be called only after the related higher-level function is called. This is because the higher-level function returns a handle and sets up a state at the protocol level that is a prerequisite to the successful execution of the dependent function or functions.

| General Win32 Internet Functions | |
|---|---|
| InternetOpen | Initializes the application's use of the Win32 Internet functions and creates the root HINTERNET handle |
| InternetConnect | Opens an FTP, Gopher, or HTTP |

| | |
|---|---|
| | session with a server. This function requires a handle created by **InternetOpen** . |
| **InternetCloseHandle** | Closes any designated handle created by a Win32 Internet function and any handles derived from that handle. |
| **InternetQueryOption** | Queries the setting of an Internet option. |
| **InternetSetOption** | Sets an Internet option. |
| **InternetSetStatusCallback** | Sets a callback function that is called with status information. Assigns a callback function to the designated HINTERNET handle and all of the handles derived from it. |
| **InternetStatusCallback** | This is a placeholder for the application-defined status callback. |
| **InternetConfirmZoneCrossing** | Checks for changes between secure and non-secure URLs. |
| **InternetTimeFromSystemTime** | Formats a date and time according to the specified RFC format (as specified in the HTTP version 1.0 specification). |
| **InternetTimeToSystemTime** | Takes an HTTP time/date string and converts it to a **SYSTEMTIME** structure. |
| **InternetAttemptConnect** | Allows an application to attempt to connect to the Internet before issuing any requests. |
| **InternetReadFile** | Dowloads data from the Internet. This function requires a handle created by **InternetOpenUrl**, **FtpOpenFile**, **GopherOpenFile**, or **HttpOpenRequest**. |
| **InternetSetFilePointer** | Sets the position for the next read in a file. This function requires a handle created by **InternetOpenUrl** (on an HTTP URL only) or a handle created by **HttpOpenRequest** using the GET method. |
| **InternetFindNextFile** | Continues file enumeration or search. This function requires a handle created by **FtpFindFirstFile** or **GopherFindFirstFile**. |
| **InternetQueryDataAvailable** | Queries the amount of data available. This function requires a handle created by **FtpOpenFile**, **GopherOpenFile**, or **HttpOpenRequest**. |
| **InternetGetLastResponseInfo** | Retrieves the text of the server's response to the FTP command. This function requires a handle created by **InternetConnect**. |

| | |
|---|---|
| InternetWriteFile | Writes data to an open file. This function requires a handle created by FtpOpenFile. |
| InternetCrackUrl | Parses a URL string into components. |
| InternetCreateUrl | Creates a URL string from components. |
| InternetCanonicalizeUrl | Converts a URL to a canonical form. |
| InternetCombineUrl | Combines base and relative URLs. |
| InternetOpenUrl | Begins retrieving a complete FTP, Gopher, or HTTP URL. This function requires a handle created by InternetOpen. |
| FTP Functions | |
| FtpFindFirstFile | Starts file enumeration or file search in the current directory. This function requires a handle created by InternetConnect. |
| FtpGetFile | Retrieves an entire file from the server. This function requires a handle created by InternetConnect. |
| FtpPutFile | Writes an entire file to the server. This function requires a handle created by InternetConnect. |
| FtpDeleteFile | Deletes a file on the server. This function requires a handle created by InternetConnect. |
| FtpRenameFile | Renames a file on the server. This function requires a handle created by InternetConnect. |
| FtpOpenFile | Initiates access to a file on the server for either reading or writing. This function requires a handle created by InternetConnect. |
| FtpCreateDirectory | Creates a new directory on the server. This function requires a handle created by InternetConnect. |
| FtpRemoveDirectory | Deletes a directory on the server. This function requires a handle created by InternetConnect. |
| FtpSetCurrentDirectory | Changes the client's current directory on the server. This function requires a handle created by InternetConnect. |
| FtpGetCurrentDirectory | Returns the client's current directory on the server. This function requires a handle created by InternetConnect. |
| Gopher Functions | |
| GopherFindFirstFile | Starts enumerating a Gopher directory listing. This function requires a handle created by |

| | |
|---|---|
| | InternetConnect. |
| GopherOpenFile | Starts retrieving a Gopher object. This function requires a handle created by InternetConnect. |
| GopherCreateLocator | Forms a Gopher locator for use in other Gopher function calls. |
| GopherGetAttribute | Retrieves attribute information on the Gopher object. This function requires a handle created by InternetConnect. |
| HTTP (World Wide Web) Functions | |
| HttpOpenRequest | Opens an HTTP request handle. This function requires a handle created by InternetConnect. |
| HttpAddRequestHeaders | Adds HTTP request headers to the HTTP request handle. This function requires a handle created by HttpOpenRequest. |
| HttpSendRequest | Sends the specified request to the HTTP server. This function requires a handle created by HttpOpenRequest. |
| HttpQueryInfo | Queries information about an HTTP request. This function requires a handle created by HttpOpenRequest. |
| InternetErrorDlg | Displays predefined dialog boxes for common Internet error conditions. Requires the handle used in the call to the function HttpSendRequest. |
| Cookie Functions | |
| InternetGetCookie | Returns cookies for the specified URL and all its parent URLs. |
| InternetSetCookie | Sets a cookie on the specified URL. |
| Cache Functions | |
| CommitUrlCacheEntry | Caches data in the specified file in the cache storage and associates it with the given URL. |
| CreateUrlCacheEntry | Allocates the requested cache storage and creates a local file name for saving the cache entry corresponding to the source name. |
| GetUrlCacheEntryInfo | Retrieves information about a cache entry. |
| ReadUrlCacheEntryStream | Reads the cached data from a stream that has been opened using RetrieveUrlCacheEntryStream. |
| RetrieveUrlCacheEntryFile | Retrieves a cache entry from the cache in the form of a file. |
| RetrieveUrlCacheEntryStream | Provides the most efficient and implementation-independent way of |

| | |
|---|---|
| | accessing the cache data. |
| SetUrlCacheEntryInfo | Sets the specified members of the INTERNET_CACHE_ENTRY_INFO structure. |
| UnlockUrlCacheEntryFile | Unlocks the cache entry that was locked while the file was retrieved for use from the cache. |
| UnlockUrlCacheEntryStream | Closes the stream that has been retrieved using RetrieveUrlCacheEntryStream. |
| DeleteUrlCacheEntry | Removes the file associated with the source name from the cache, if the file exists. |
| FindCloseUrlCache | Closes the specified enumeration handle. |
| FindFirstUrlCacheEntry | Begins the enumeration of the cache. |
| FindNextUrlCacheEntry | Retrieves the next entry in the cache. |

# ndles

The handles that are created and used by the Win32 Internet functions are opaque handle types called HINTERNETS. These handles returned by the Win32 Internet function APIs are not interchangeable with the base Win32 handles, so they cannot be used with Win32 APIs such as **ReadFile** or **CloseHandle**. Similarly, base Win32 handles cannot be used with the Win32 Internet function APIs. For example, a handle returned by **CreateFile** cannot be passed to **InternetReadFile**.

If a callback function was registered for a handle, all operations on that handle can generate status indications, provided that the context value that was supplied when the handle was created was not zero. Providing a zero context value is a method to force an operation to complete synchronously, even though INTERNET_FLAG_ASYNC was specified in **InternetOpen**.

Status indications are mainly intended to give the application feedback as to the progress of an operation, and are mainly concerned with network operations, such as resolving a host name, connecting to a server, and receiving data. Three special-purpose status indications can be made for a handle:

- INTERNET_STATUS_REQUEST_COMPLETE is indicated when an asynchronous operation completes.
- INTERNET_STATUS_HANDLE_CREATED is indicated when the handle is initially created.
- INTERNET_STATUS_HANDLE_CLOSING is the last status indication that will be made for a handle.

The application must check the INTERNET_ASYNC_RESULTS structure to determine whether the operation succeeded or failed after receiving an INTERNET_STATUS_REQUEST_COMPLETE indication.

The **InternetCloseHandle** function closes handles of type HINTERNET and all handles that descended from it in the **Handle Hierarchy** function. Note that handle values are recycled quickly; therefore, if a handle is closed and a new handle is generated immediately, there is a good chance that the new handle will have the same value as the handle just closed.

## Handle Hierarchy

Handles returned from Win32 Internet functions are maintained in a tree hierarchy. The handle returned by the **InternetOpen** function is the root node. Handles returned from the **InternetConnect** function occupy the next level. Currently, handles that are returned by open or find functions, such as **HttpOpenRequest** and **FtpFindFirstFile**, are the leaf nodes. This structure can be used by **InternetCloseHandle** to close a single handle or an entire subtree.

## Multithreaded Access

The Win32 Internet functions are "reentrant" in the sense that there can be multiple calls to an individual function from different threads. The functions complete any necessary synchronization. However, multiple simultaneous calls using the same Internet connection can 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 which call will be completed first, or which thread will receive file data first. Applications that use multiple threads for the same Internet connection are responsible for synchronization between threads to ensure a predictable return of information.

## Error Handling

The Win32 Internet functions return error information in the same way as Win32 functions. Return values tell whether the function is successful or not. For example, some Internet functions return a BOOL value that is TRUE if the function succeeded or FALSE if it failed, and others return a handle of type HINTERNET. A NULL handle indicates that the function failed, and any other value indicates that it succeeded.

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

Both **GetLastError** and **InternetGetLastResponseInfo** operate on a per-thread basis. If two threads call Internet functions at the same time, error information will be returned for each of the individual threads so that there is no conflict between the threads.

# icode Support

The Win32 Internet functions do not currently provide support for Unicode. However, support will be provided in future versions.

# I Flags

Many of the Win32 Internet functions accept a double-word array of flags as a parameter. The following is a brief description of the defined flags:

INTERNET_FLAG_RELOAD

Force a download of the requested file, object, or directory listing from the origin server, not from the cache. The GopherFindFirstFile, GopherOpenFile, FtpFindFirstFile, FtpGetFile, FtpOpenFile, FtpPutFile, HttpOpenRequest, and InternetOpenUrl functions utilize this flag.

INTERNET_FLAG_RAW_DATA

Return the data as a GOPHER_FIND_DATA structure when retrieving Gopher directory information, or as a WIN32_FIND_DATA structure when retrieving FTP directory information using the InternetOpenUrl API.

INTERNET_FLAG_EXISTING_CONNECT

Attempt to use an existing InternetConnect object if one exists with the same attributes required to make the request. Only the InternetOpenUrl function uses this flag.

INTERNET_FLAG_ASYNC

Make only asynchronous requests on handles descended from the handle returned from this function. Only the InternetOpen function uses this flag.

INTERNET_FLAG_PASSIVE

Use passive FTP semantics. Only InternetConnect and InternetOpenUrl use this flag. InternetConnect uses this flag for FTP requests, and InternetOpenUrl uses this flag for FTP files and directories.

INTERNET_FLAG_NO_CACHE_WRITE

Do not add the returned entity to the cache. This flag is used by GopherFindFirstFile, GopherOpenFile, FtpFindFirstFile, FtpGetFile, FtpPutFile, HttpOpenRequest, and InternetOpenUrl.

INTERNET_FLAG_MAKE_PERSISTENT

Add the returned entity to the cache as a persistent entity. This means that standard cache cleanup, consistency checking, or garbage collection cannot remove this item from the cache. This flag is used by GopherFindFirstFile, GopherOpenFile, FtpFindFirstFile, FtpGetFile, FtpOpenFile, FtpPutFile, HttpOpenRequest, and InternetOpenUrl.

INTERNET_FLAG_OFFLINE

Do not make network requests. All entities are returned from the cache. If the requested item is not in the

cache, a suitable error, such as ERROR_FILE_NOT_FOUND, is returned. Only the **InternetOpen** function uses this flag.

INTERNET_FLAG_SECURE

Use secure transaction semantics. This translates to using Secure Sockets Layer/Private Communications Technology (SSL/PCT) and is only meaningful in HTTP requests. This flag is used by **HttpOpenRequest** and **InternetOpenUrl**, but this is made redundant if "https://" appears in the URL.

INTERNET_FLAG_KEEP_CONNECTION

Use keep-alive semantics, if available, for the connection. This flag is used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_NO_AUTO_REDIRECT

Do not automatically handle redirection in **HttpSendRequest**. This flag can also be used by **InternetOpenUrl** for HTTP requests.

INTERNET_FLAG_IGNORE_CERT_CN_INVALID

Disable Win32 Internet function checking of SSL/PCT-based certificates that are returned from the server against the host name given in the request. Win32 Internet functions use a simple check against certificates by comparing for matching host names and simple wildcarding rules. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_IGNORE_CERT_DATE_INVALID

Disable Win32 Internet function checking of SSL/PCT-based certificates for proper validity dates. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTPS

Disable the ability of the Win32 Internet functions to detect this special type of redirect. When this flag is used, Win32 Internet functions transparently allow redirects from HTTP to HTTPS URLs. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTP

Disable the ability of the Win32 Internet functions to detect this special type of redirect. When this flag is used, Win32 Internet functions transparently allow redirects from HTTPS to HTTP URLs. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_READ_PREFETCH

Download the entity without requiring the application to initiate reads from the network. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_NO_COOKIES

Do not automatically add cookie headers to requests, and do not automatically add returned cookies to the cookie database. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_NO_AUTH

Do not attempt authentication automatically. This flag can be used by **HttpOpenRequest** and **InternetOpenUrl** (for HTTP requests).

INTERNET_FLAG_TRANSFER_ASCII

Transfer file as ASCII (FTP only). This flag can be used by **FtpOpenFile**, **FtpGetFile**, and **FtpPutFile**.

INTERNET_FLAG_TRANSFER_BINARY

Transfer file as binary (FTP only). This flag can be used by **FtpOpenFile**, **FtpGetFile**, and **FtpPutFile**.

INTERNET_FLAG_MUST_CACHE_REQUEST

Cause operation to fail if the downloaded file cannot be cached. This flag can be used by **GopherFindFirstFile**, **GopherOpenFile**, **FtpFindFirstFile**, **FtpGetFile**, **FtpOpenFile**, **FtpPutFile**, **HttpOpenRequest**, and **InternetOpenUrl**.

INTERNET_FLAG_RESYNCHRONIZE

Perform a conditional download of the file. This flag can be used by **GopherFindFirstFile**, **GopherOpenFile**, **FtpFindFirstFile**, **FtpGetFile**, **FtpOpenFile**, **FtpPutFile**, **HttpOpenRequest**, and **InternetOpenUrl**.

INTERNET_FLAG_HYPERLINK

Force a reload if there was no Expires time and no Last-Modified time returned from the server when determining whether to reload the item from the network. This flag can be used by **GopherFindFirstFile**, **GopherOpenFile**, **FtpFindFirstFile**, **FtpGetFile**, **FtpOpenFile**, **FtpPutFile**, **HttpOpenRequest**, and **InternetOpenUrl**.

## ntext Values

Many of the Win32 Internet functions that create a handle can also accept an application-defined context value. This context value is associated with the handle until it is closed. For example, you can specify a context value to the **HttpOpenRequest** function that will be used in all callbacks made for requests against this handle. If the INTERNET_FLAG_ASYNC flag is specified, supplying a zero context value forces the request to be synchronous.

## ynchronous Support

By default, the Win32 Internet functions operate synchronously. An application can request asynchronous operation by setting the INTERNET_FLAG_ASYNC flag in the call to the **InternetOpen** function. All future calls made against handles derived from the handle returned from **InternetOpen** will be made asynchronously.

The rationale for asynchronous versus synchronous operation is to allow a single-threaded application to maximize its utilization of the CPU without having to wait for network I/O to complete. Therefore, depending on the request, the operation may complete synchronously or asynchronously. The application should check the return code. If a function returns FALSE or NULL, and **GetLastError** returns ERROR_IO_PENDING, the request has been made asynchronously, and the application will be called back with INTERNET_STATUS_REQUEST_COMPLETE when the function has completed.

For an application to be able to make requests asynchronously, it must set the INTERNET_FLAG_ASYNC flag in the call to **InternetOpen**, it must register a valid callback function, and it must supply a non-zero context value.

# Persistent Caching

Win32 Internet functions have built-in caching support that is simple yet flexible. Any data that is retrieved from the network is cached on the hard disk and retrieved for subsequent requests. The caller has the option of controlling the caching on a per-request basis. In the case of HTTP, most headers received from the server are also cached. When an HTTP request is satisfied from the cache, the cached headers are also returned to the caller. This makes data download from Win32 Internet functions seamless, whether it is coming from the cache or from the wire.

# Buffers and Buffer Parameters

For the APIs that return strings, there is an input *lpszBuffer* parameter and an *lpdwBufferLength* parameter. The *lpszBuffer* can be NULL, and *lpdwBufferLength* must be a valid pointer to a DWORD variable. If the input buffer pointed to by *lpszBuffer* is either too small to hold the output string or NULL, a failure indication will be returned by the API and **GetLastError** will return ERROR_INSUFFICIENT_BUFFER. The variable pointed to by *lpdwBufferLength* will contain a number that represents the number of bytes that are required by the function to return the requested string, which includes the NUL terminator. The application should allocate a buffer of this size, set the variable pointed to by *lpdwBufferLength* to this value, and resubmit the request. If the size of the buffer is sufficient to receive the requested string, the string is copied to the output buffer with a NUL terminator and a success indication is returned by the API. The variable pointed to by *lpdwBufferLength* will now contain the number of characters stored in the buffer, excluding the NUL terminator.

# General Win32 Internet Functions

The general Win32 Internet functions perform basic Internet file manipulations.

## InternetAttemptConnect

```
DWORD InternetAttemptConnect(
    DWORD dwReserved
```

```
);
```

Attempts to make a connection to the Internet. This function allows an application to first attempt to connect before issuing any requests. If the connection fails, the application can enter off-line mode.

- Returns ERROR_SUCCESS if successful, or a Win32 error value otherwise.

*dwReserved*
Reserved; must be zero.

## InternetCanonicalizeUrl

```
BOOL InternetCanonicalizeUrl(
    IN LPCTSTR lpszUrl,
    OUT LPTSTR lpszBuffer,
    IN OUT LPDWORD lpdwBufferLength,
    IN DWORD dwFlags
);
```

Converts a URL to a canonical form, which includes converting unsafe characters into escape sequences.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. Possible errors include:

| | |
|---|---|
| ERROR_INVALID_PARAMETER | Bad string, buffer, buffer size, or flags parameter. |
| ERROR_INSUFFICIENT_BUFFER | Canonicalized URL is too large to fit in the buffer provided. The **lpdwBufferLength* parameter is set to the size, in bytes, of the buffer required to hold the resultant, canonicalized URL. |
| ERROR_BAD_PATHNAME | The URL could not be canonicalized. |
| ERROR_INTERNET_INVALID_URL | The format of the URL is invalid. |

*lpszUrl*
Address of the input URL to canonicalize.

*lpszBuffer*
Address of the buffer that receives the resulting canonicalized URL.

*lpdwBufferLength*
Length, in bytes, of the *lpszBuffer* buffer. If the function succeeds, this parameter receives the length of the *lpszBuffer* buffer裡he length does not include the terminating null. If the function fails, this

parameter receives the required length, in bytes, of the *lpszBuffer* buffer裡he required length includes the terminating null.

*dwFlags*

Flags that control canonicalization. Can be one of these values:

| Value | Meaning |
|---|---|
| ICU_DECODE | Convert %XX escape sequences to characters. |
| ICU_NO_ENCODE | Do not convert unsafe characters to escape sequence. |
| ICU_NO_META | Do not remove meta sequences (such as "." and "..") from the URL. |
| ICU_ENCODE_SPACES_ONLY | Encode spaces only. |
| ICU_BROWSER_MODE | Do not encode or decode characters after #' or '?', and do not remove trailing white space after '?'. If this value is not specified, the entire URL is encoded and trailing white space is removed. |

If no flags are specified, the function converts all unsafe characters and meta sequences (such as \.,\ .., and \...) to escape sequences.

**InternetCanonicalizeUrl** always encodes by default, even if the ICU_DECODE flag has been specified. To decode without re-encoding, use ICU_DECODE | ICU_NO_ENCODE. If the ICU_DECODE flag is used without ICU_NO_ENCODE, the URL is decoded before being parsed; unsafe characters then are re-encoded after parsing. This function will handle arbitrary protocol schemes, but to do so it must make inferences from the unsafe character set.

## InternetCloseHandle

```
BOOL InternetCloseHandle(
    IN HINTERNET hInet
);
```

Closes a single Internet handle or a subtree of Internet handles.

- Returns TRUE if the handle is successfully closed, or FALSE otherwise. To get extended error information, call **GetLastError**.

*hInet*

Valid Internet handle to be closed.

This function can be used to close any Internet handle or subtree of handles of the type HINTERNET and free any associated resources. The function terminates any pending operations on the handle and discards any outstanding data. If a thread is blocking a call to Wininet.dll, another thread in the application can call **InternetCloseHandle** on the Internet handle being used by the first thread to cancel the operation and unblock the first thread.

**InternetCloseHandle** should be used to close the handle returned from **InternetOpen** when the application has finished using the Internet DLL.

If there is a status callback registered for the handle being closed and the handle was created with a non-NULL context value, an INTERNET_STATUS_HANDLE_CLOSING callback will be made. This indication will be the last callback made from a handle and indicates that the handle is being destroyed.

If asynchronous requests are pending for the handle or any of its child handles, the handle cannot be closed immediately, but it will be invalidated. Any new requests attempted using the handle will return with an ERROR_INVALID_HANDLE notification. The asynchronous requests will complete with INTERNET_STATUS_REQUEST_COMPLETE. Applications must be prepared to receive any INTERNET_STATUS_REQUEST_COMPLETE indications on the handle before the final INTERNET_STATUS_HANDLE_CLOSING indication is made, which indicates that the handle is completely closed.

An application can call **GetLastError** to determine if requests are pending. If **GetLastError** returns ERROR_IO_PENDING, there were outstanding requests when the handle was closed.

See also **FtpFindFirstFile**, **FtpOpenFile**, **GopherFindFirstFile**, **HttpOpenRequest**, **InternetConnect**, **InternetOpen**

## InternetCombineUrl

```
BOOL InternetCombineUrl(
    IN LPCTSTR lpszBaseUrl,
    IN LPCTSTR lpszRelativeUrl,
    OUT LPTSTR lpszBuffer,
    IN OUT LPDWORD lpdwBufferLength,
    IN DWORD dwFlags
);
```

Combines a base and relative URL into a single URL. The resultant URL will be canonicalized (see **InternetCanonicalizeUrl**).

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call **GetLastError**. Possible error codes include:

  ERROR_INVALID_PARAMETER
  Bad string, buffer, buffer size, or flags parameter.

  ERROR_INSUFFICIENT_BUFFER
  The **lpdwBufferLength* parameter, in bytes, of the buffer required to hold the resultant, combined URL.

  ERROR_BAD_PATHNAME
  The URLs could not be combined.

  ERROR_INTERNET_INVALID_URL
  The format of the URL is invalid.

*lpszBaseUrl*
Address of the base URL to be combined.

*lpszRelativeUrl*
Address of the relative URL to be combined.

*lpszBuffer*
Address of a buffer that receives the resulting URL.

*lpdwBufferLength*
Size, in bytes, of the *lpszBuffer* buffer. If the function succeeds, this parameter receives the length, in characters, of the resultant combined URL裡he length does not include the null terminator. If the function fails, this parameter receives the length, in bytes, of the required buffer裡he length includes the null terminator.

*dwFlags*
Flags controlling the operation of the function. For a description of the flags, see **InternetCanonicalizeUrl**.

## InternetConnect

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

Opens an FTP, Gopher, or HTTP session for a given site.

- Returns a valid handle to the FTP, Gopher, or HTTP session if the connection is successful, or NULL otherwise. To get extended error information, call **GetLastError**. An application can also use **InternetGetLastResponseInfo** to determine why access to the service was denied.

*hInternetSession*
Handle of the current Internet session. The handle must have been returned by a previous call to **InternetOpen**.

*lpszServerName*
Address of a null-terminated string that contains the host name of an Internet server. Alternately, the string can contain the IP number of the site in ASCII dotted-decimal format (for example, 11.0.1.45).

*nServerPort*
Number of the TCP/IP port on the server to connect to. Can be one of the values in the following list. If this parameter is set to INTERNET_INVALID_PORT_NUMBER, the function uses the default port for the specified service. These values do not cause the function to use this

protocol. The value sets the port to be used. A flag must be used to set the service.

| Value | Meaning |
|---|---|
| INTERNET_DEFAULT_FTP_PORT | Use the default port for FTP servers (port 21). |
| INTERNET_DEFAULT_GOPHER_PORT | Use the default port for Gopher servers (port 70). |
| INTERNET_DEFAULT_HTTP_PORT | Use the default port for HTTP servers (port 80). |
| INTERNET_DEFAULT_HTTPS_PORT | Use the default port for HTTPS servers (port 443). |

*lpszUsername*

Address of a null-terminated string that contains the name of the user to log on. If this parameter is NULL, the function uses an appropriate default, except for HTTP. A NULL parameter in HTTP causes the server to return an error. For the FTP protocol, the default is anonymous.

*lpszPassword*

Address of a null-terminated string that contains the password to use to log on. If both *lpszPassword* and *lpszUsername* are NULL, the function uses the default anonymous password. In the case of FTP, the default anonymous password is the user's e-mail name. If *lpszPassword* is NULL, but *lpszUsername* is not NULL, the function uses a blank password. The following table describes the behavior for the four possible settings of *lpszUsername* and *lpszPassword*:

| *lpszUsername* | *lpszPassword* | User name sent to FTP server | Password sent to FTP server |
|---|---|---|---|
| NULL | NULL | "anonymous" | User's e-mail name |
| Non-NULL string | NULL | *lpszUsername* | "" |
| NULL | Non-NULL string | ERROR | ERROR |
| Non-NULL string | Non-NULL string | *lpszUsername* | *lpszPassword* |

*dwService*

Type of service to access. Can be one of these values:

| Value | Meaning |
|---|---|
| INTERNET_SERVICE_FTP | FTP service. |
| INTERNET_SERVICE_GOPHER | Gopher service. |
| INTERNET_SERVICE_HTTP | HTTP service. |

*dwFlags*

Flags specific to the service used. Can be one of these values:

| If *dwService* is: | *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.

The **InternetConnect** function is required before communicating with any Internet service.

Having a connect function for all protocols, even those that do not use persistent connections, lets an application communicate common information about several requests using a single function call. In addition, this allows for future versions of Internet protocols that do not require a connection to be established for every client request.

For FTP sites, **InternetConnect** actually establishes a connection with the server; for others, such as Gopher, the actual connection is not established until the application requests a specific transaction.

For maximum efficiency, applications using the Gopher and HTTP protocols should try to minimize calls to **InternetConnect** and avoid calling this function for every transaction requested by the user. One way to accomplish this is to keep a small cache of handles returned from **InternetConnect**; when the user makes a request to a previously accessed server, that session handle is still available.

An application that needs to display multiline text information sent by an FTP server can use **InternetGetLastResponseInfo** to retrieve the text.

For FTP connections, if *lpszUsername* is NULL, **InternetConnect** sends the string "anonymous" as the user name. If *lpszPassword* is NULL, **InternetConnect** attempts to use the user's e-mail name as the password.

To close the handle returned from **InternetConnect**, the application should call **InternetCloseHandle**. This function disconnects the client from the server and frees all resources associated with the connection.

See also **InternetCloseHandle**, **InternetOpen**

## InternetConfirmZoneCrossing

```
DWORD InternetConfirmZoneCrossing(
    IN HWND hWnd,
    IN LPSTR szUrlPrev,
    IN LPSTR szUrlNew,
    IN BOOL bPost
);
```

Checks for changes between secure and non-secure URLs. When a change occurs in security between two URLs, an application should allow the user to acknowledge this change, typically by displaying a dialog box.

- Returns ERROR_SUCCESS if the user confirmed that it was okay to continue or there was no user input needed, ERROR_CANCELLED if the user canceled, or ERROR_NOT_ENOUGH_MEMORY if there is not enough memory to carry out the request.

*hWnd*
Handle of the parent window for any needed dialog box.

*szUrlPrev*
URL that was viewed before the current request was made.

*szUrlNew*
New URL that the user has requested to view.

*bPost*
TRUE if a post is being made in this request. This flag is ignored in this release.

## InternetCrackUrl

```
BOOL InternetCrackUrl(
    IN LPCSTR lpszUrl,
    IN DWORD dwUrlLength,
    IN DWORD dwFlags,
    IN OUT LPURL_COMPONENTS lpUrlComponents
);
```

Cracks a URL into its component parts.

- Returns TRUE if the function succeeds, or FALSE otherwise. To get extended error information, call GetLastError.

*lpszUrl*
Address of a string that contains the canonical URL to crack.

*dwUrlLength*
Length of the *lpszUrl* string, or zero if *lpszUrl* is an ASCIIZ string.

*dwFlags*
Flags controlling the operation. Can be one of these values:

| | |
|---|---|
| ICU_ESCAPE | Convert unsafe characters in the URL-path component to escape sequences. |
| ICU_DECODE | Convert encoded characters back to their normal form. This can be used only if the user provides buffers to copy the components into. |
| ICU_USERNAME | Not currently implemented. |

*lpUrlComponents*
Address of a URL_COMPONENTS structure that receives the URL components.

The required components are indicated by members of the URL_COMPONENTS structure. Each component has a pointer to the value, and has a member that stores the length of the stored value. If both the value and the length for a component are equal to zero, that component is not returned. If the pointer to the value of the component is NULL and the value of its corresponding length member is non-zero, the address of the first character of the corresponding component in the *lpszUrl* string is stored in

the pointer, and the length of the component is stored in the length member. Otherwise, the pointer contains the address of the user-supplied buffer where the component is copied, and the length member is updated with the length of the copied component, minus 1 for the trailing string terminator.

See also **FtpOpenFile**, **InternetCloseHandle**, **InternetFindNextFile**, **InternetSetStatusCallback**

## InternetCreateUrl

```
BOOL
    IN LPURL_COMPONENTS lpUrlComponents,
    IN DWORD dwFlags,
    OUT LPSTR lpszUrl,
    IN OUT LPDWORD lpdwUrlLength
);
```

Creates a URL from its component parts.

- Returns TRUE if the function succeeds, or FALSE otherwise. To get extended error information, call **GetLastError**. If the function finds no matching files, **GetLastError** returns ERROR_NO_MORE_FILES.

***lpUrlComponents***
Address of a URL_COMPONENTS structure that contains the components from which to create the URL.

***dwFlags***
Flags that control the operation of this function. Can be a combination of these values:

| Value | Meaning |
|---|---|
| ICU_ESCAPE | Convert unsafe characters in the URL-path component to escape sequences. |
| ICU_USERNAME | When adding the user name, use the name that was specified at login time. |

***lpszUrl***
Address of a buffer that receives the URL.

***lpdwUrlLength***
Length, in bytes, of the *lpszUrl* buffer. When the function returns, this parameter receives the length, in bytes, of the URL string, minus 1 for the terminating character. If **GetLastError** returns ERROR_INSUFFICIENT_BUFFER, this parameter receives the number of bytes required to hold the created URL.

## InternetErrorDlg

```
DWORD InternetErrorDlg(
```

```
    IN HWND hWnd,
    IN OUT HINTERNET hInternet,
    IN DWORD dwError,
    IN DWORD dwFlags,
    IN OUT LPVOID *lppvData
);
```

Displays a dialog box that explains why an error occurred with an **HttpSendRequest** Win32 Internet function. An application can call this function for several different error codes, and can do bookkeeping based on the user's response to the dialog box.

- Returns ERROR_SUCCESS, ERROR_CANCELLED, or ERROR_INTERNET_FORCE_RETRY.

***hWnd***
Handle of the parent window for any needed dialog box. This parameter can be NULL if no dialog box is needed.

***hInternet***
Handle of the Internet connection used in the call to **HttpSendRequest**.

***dwError***
Error value for which to display a dialog box. Can be one of these values:

ERROR_INTERNET_HTTP_TO_HTTPS_ON_REDIR
: Notifies the user of the zone crossing to and from a secure site.

ERROR_INTERNET_INCORRECT_PASSWORD
: Displays a dialog box for obtaining the user's name and password. (On Windows 95, the function attempts to use the network caching user interface and disk cache.)

ERROR_INTERNET_INVALID_CA
: Notifies the user that the Win32 Internet function does not have a certificate for this SSL site.

ERROR_INTERNET_POST_IS_NON_SECURE
: Displays a warning about posting data to the server through a non-secure connection.

ERROR_INTERNET_SEC_CERT_CN_INVALID
: Displays an Invalid SSL Common Name dialog box, and lets the user view the incorrect certificate. Also allows the user to select a certificate in response to a server request.

ERROR_INTERNET_SEC_CERT_DATE_INVALID
: Tells the user that the SSL certificate has expired.

***dwFlags***
Action flags. Can be a combination of these values:

FLAGS_ERROR_UI_FLAGS_CHANGE_OPTIONS
: If the function succeeds, store the results of the dialog box in the Internet handle.

FLAGS_ERROR_UI_FLAGS_GENERATE_DATA

Query the Internet handle for needed information. The function constructs the appropriate data structure for the error. (For example, for Cert CN failures, the function will grab the certificate.)

FLAGS_ERROR_UI_FILTER_FOR_ERRORS

Scan the returned headers for errors. Call after every **HttpSendRequest**. The function detects any hidden errors, such as an authentication error.

*lppvData*

Address of a yet-to-be-defined structure. The structure may be different for each error that needs to be handled

## InternetFindNextFile

```
BOOL InternetFindNextFile(
    IN HINTERNET hFind,
    OUT LPVOID lpvFindData
);
```

Continues a file search started as a result of a previous call to **FtpFindFirstFile** or **GopherFindFirstFile**.

- Returns TRUE if the function succeeds, or FALSE otherwise. To get extended error information, call **GetLastError**. If the function finds no matching files, **GetLastError** returns ERROR_NO_MORE_FILES.

*hFind*

Valid handle returned from either **FtpFindFirstFile** or **GopherFindFirstFile**.

*lpvFindData*

Address of the 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. The FTP protocol returns a **WIN32_FIND_DATA** structure, and the Gopher protocol returns a **GOPHER_FIND_DATA** structure.

See also **FtpFindFirstFile**, **GopherFindFirstFile**

## InternetGetLastResponseInfo

```
BOOL InternetGetLastResponseInfo(
    OUT LPDWORD lpdwError,
    OUT LPTSTR lpszBuffer OPTIONAL,
    IN OUT LPDWORD lpdwBufferLength
);
```

Retrieves the last Win32 Internet function error description or server response on the thread calling this API.

- Returns TRUE if error text was successfully written to the buffer, or FALSE otherwise. To get extended error information, call GetLastError. If the buffer is too small to hold all the error text, GetLastError returns ERROR_INSUFFICIENT_BUFFER, and the *lpdwBufferLength* parameter contains the minimum buffer size required to return all the error text.

*lpdwError*
Address of a variable that receives an error code pertaining to the operation that failed.

*lpszBuffer*
Address of a buffer that receives the error text.

*lpdwBufferLength*
Size of the *lpszBuffer* buffer. When the function returns, this parameter contains the size of the string written to the buffer.

The FTP and Gopher protocols can return additional text information along with most errors. This extended error information can be retrieved by using the **InternetGetLastResponseInfo** function whenever **GetLastError** returns ERROR_INTERNET_EXTENDED_ERROR (occurring after an unsuccessful function call).

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, note that the value returned in *lpdwBufferLength* does not include the terminating zero.

**InternetGetLastResponseInfo** can be called multiple times until another Win32 Internet function API is called on this thread. When another API is called, the internal buffer that is storing the last response information is cleared.

## InternetOpen

```
HINTERNET InternetOpen(
    IN LPCTSTR lpszAgent,
    IN DWORD dwAccessType,
    IN LPCTSTR lpszProxyName OPTIONAL,
    IN LPCSTR lpszProxyBypass OPTIONAL,
    IN DWORD dwFlags
);
```

Initializes an application's use of the Win32 Internet functions.

- Returns a valid handle that the application passes on to subsequent Win32 Internet functions. If **InternetOpen** fails, it returns NULL. To get a specific error code, call **GetLastError**.

*lpszAgent*
Address of a string that contains the name of the application or entity calling the Internet functions (for example, Microsoft Internet Explorer). This name is used as the user agent in the HTTP protocol.

*dwAccessType*
Type of access required. Can be one of these values:

INTERNET_OPEN_TYPE_DIRECT
Resolve all host names locally.

INTERNET_OPEN_TYPE_PROXY
Pass requests to the proxy unless a proxy bypass list is supplied and the name to be resolved bypasses the proxy. In this case, the function proceeds as for INTERNET_OPEN_TYPE_DIRECT.

INTERNET_OPEN_TYPE_PRECONFIG
Retrieve the proxy or direct configuration from the registry.

*lpszProxyName*
Address of a string that contains the name of the proxy server (or servers) to use if proxy access was specified. If this parameter is NULL, the function reads proxy information from the registry. For more information about this parameter, see the comments below.

*lpszProxyBypass*
Address of an optional list of host names or IP addresses, or both, that are known locally. Requests to these names are not routed through the proxy. The list can contain wildcards, such as "157.55.* *int*", meaning any IP address starting with 157.55, or any name containing the substring "int", will bypass the proxy.

If this parameter specifies the "<local>" macro as the only entry, the function bypasses any host name that does not contain a period. For example, "www.microsoft.com" would be routed to the proxy, whereas "internet" would not.

If this parameter is NULL, the function reads the bypass list from the registry.

*dwFlags*
Flag that indicates various options affecting the behavior of the function. Can be a combination of these values:

INTERNET_FLAG_OFFLINE
Satisfy download operations on this handle through the persistent cache only. If the item does not exist in the cache, the function returns an appropriate error code.

INTERNET_FLAG_ASYNC
Future operations on this handle may fail with ERROR_IO_PENDING. A status callback will be made with INTERNET_STATUS_REQUEST_COMPLETE. This callback will be on a thread other than the one for the original request. A status callback routine must be registered or the functions will be completed synchronously.

This function 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 finishes using the Internet functions, it should call **InternetCloseHandle** to free the handle and any associated resources.

If *dwFlags* includes INTERNET_FLAG_ASYNC, all handles derived from this handle will have asynchronous behavior as long as a status callback routine is registered. For a function to be completed synchronously, *dwContext* must be set to zero for that call.

By default, the function assumes that the proxy specified by *lpszProxyName* is a CERN proxy. For example, "proxy" defaults to a CERN proxy called "proxy" that listens at port 80 (decimal). An application can specify more than one proxy, including different proxies for the different protocols. For example, if you specify "ftp=ftp://ftp-gw gopher=http://jericho:99 proxy", FTP requests are made through the FTP proxy "ftp-gw", which listens at port 21 (default for FTP), and Gopher requests are made through a CERN proxy called "jericho", which listens at port 99. All other requests (for example, HTTP requests) are made through the CERN proxy called "proxy", which listens at port 80. Note that if the application is only using FTP, for example, it would not need to specify "ftp=ftp://ftp-gw:21". It could specify just "ftp-gw". An application must specify the protocol names only if it will be using more than one protocol per handle returned by **InternetOpen**.

The application can make any number of calls to **InternetOpen**, although a single call is normally sufficient. The application may need to have separate behaviors defined for each **InternetOpen** instance, such as different proxy servers configured for each.

See also **InternetCloseHandle**

## InternetOpenUrl

```
HINTERNET InternetOpenUrl(
    IN HINTERNET hInternetSession,
    IN LPCTSTR lpszUrl,
    IN LPCTSTR lpszHeaders OPTIONAL,
    IN DWORD dwHeadersLength,
    IN DWORD dwFlags,
    IN DWORD dwContext
);
```

Begins reading a complete FTP, Gopher, or HTTP Universal Resource Locator (URL). Use **InternetCanonicalizeUrl** first if the URL being used contains a relative URL and a base URL separated by blank spaces.

- Returns a valid handle to the FTP, Gopher, or HTTP URL if the connection is successfully established, or NULL if the connection fails. To get a specific error code, call **GetLastError**. To determine why access to the service was denied, call **InternetGetLastResponseInfo**.

*hInternetSession*

Handle of the current Internet session. The handle must have been returned by a previous call to **InternetOpen**.

*lpszUrl*
Address of a string that contains the URL to begin reading. Only URLs beginning with ftp:, gopher:, http:, or https: are supported.

*lpszHeaders*
Address of a string that contains the headers to be sent to the HTTP server. (For more information, see the description of the *lpszHeaders* parameter to **HttpSendRequest**.)

*dwHeadersLength*
Length, in characters, of the additional headers. If this parameter is -1L and *lpszHeaders* is not NULL, *lpszHeaders* is assumed to be zero-terminated (ASCIIZ) and the length is calculated.

*dwFlags*
Action flags. Can be one of these values:

INTERNET_FLAG_RELOAD
Get the data from the wire even if it is locally cached.

INTERNET_FLAG_DONT_CACHE
Do not cache the data, either locally or in any gateways.

INTERNET_FLAG_RAW_DATA
Return raw data (**WIN32_FIND_DATA** structures for FTP, and **GOPHER_FIND_DATA** structures for Gopher). If this flag is not specified, **InternetOpenUrl** returns HTML formatted for directories.

INTERNET_FLAG_SECURE
Request secure transactions on the wire with Secure Sockets Layer or PCT. This flag applies to HTTP requests only.

INTERNET_FLAG_EXISTING_CONNECT
If possible, reuse the existing connections to the server for new requests generated by **InternetOpenUrl** instead of creating a new session for each request.

*dwContext*
An application-defined value that is passed, along with the returned handle, to any callback functions.

This is a general function that an application can use to retrieve data over any of the protocols that the Win32 Internet functions support. This function is particularly useful when the application does not need to access the particulars of a protocol, but only requires the data corresponding to a URL. The **InternetOpenUrl** function parses the URL string, establishes a connection to the server, and prepares to download the data identified by the URL. The application can then use **InternetReadFile** to retrieve the URL data. It is not necessary to call **InternetConnect** before **InternetOpenUrl**.

**InternetOpenUrl** disables Gopher on ports less than 1024, except for port 70 (the standard Gopher port) and port 105 (typically used for Central Services Organization [CSO] name searches).

Use **InternetCloseHandle** to close the handle returned from **InternetOpenUrl**. However, note that closing the handle before all the URL data has been read results in the connection being terminated.

See also **HttpSendRequest**, **InternetCloseHandle**, **InternetOpen**, **InternetReadFile**

## InternetQueryDataAvailable

```
BOOL
    IN HINTERNET hFile,
    OUT LPDWORD lpdwNumberOfBytesAvailable,
    IN DWORD dwFlags,
    IN DWORD dwContext
);
```

Queries the amount of data available.

- Returns TRUE if the function succeeds, or FALSE otherwise. To get extended error information, call **GetLastError**. If the function finds no matching files, the **GetLastError** function returns ERROR_NO_MORE_FILES.

*hFile*
Valid Internet file handle, as returned by **FtpOpenFile**, **GopherOpenFile**, or **HttpOpenRequest**.

*lpdwNumberOfBytesAvailable*
Address of a variable that receives the number of available bytes.

*dwFlags*
Reserved; must be zero.

*dwContext*
Reserved; must be zero.

This function returns the number of bytes of data that are available to be read immediately by a subsequent call to **InternetReadFile**. If there is currently no data available and the end of the file has not been reached, the request waits until data becomes available. The amount of data remaining will not be recalculated until all of the available data indicated by the call to **InternetQueryDataAvailable** is read.

See also **FtpFindFirstFile**, **GopherFindFirstFile**

## InternetQueryOption

```
BOOL InternetQueryOption(
    IN HINTERNET hInternet OPTIONAL,
    IN DWORD dwOption,
    OUT LPVOID lpBuffer OPTIONAL,
        IN OUT LPDWORD lpdwBufferLength
    );
```

Queries an Internet option on the specified handle.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call GetLastError.

*hInternet*
Internet handle on which to query information.

*dwOption*
Internet option to query. Can be one of these values:

INTERNET_OPTION_CALLBACK
Returns the address of the callback function defined for this handle.

INTERNET_OPTION_CONNECT_TIMEOUT
Returns the time-out value in milliseconds to use for Internet connection requests. If a connection request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_CONNECT_RETRIES
Returns the retry count to use for Internet connection requests. If a connection attempt still fails after the specified number of tries, the request is canceled. The default is five retries.

INTERNET_OPTION_CONNECT_BACKOFF
Returns the delay value, in milliseconds, to wait between connection retries. (This flag is currently not implemented.)

INTERNET_OPTION_CONTROL_SEND_TIMEOUT
Returns the time-out value, in milliseconds, to use for non-data (control) Internet send requests. If a non-data send request takes longer than this time-out, the request is canceled. The default time-out is infinite. Currently, this value has meaning only for FTP sessions.

INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT
Returns the time-out value, in milliseconds, to use for non-data (control) Internet receive requests. If a non-data receive request takes longer than this time-out value, the request is canceled. The default time-out is infinite. Currently, this value has meaning only for FTP sessions.

INTERNET_OPTION_DATA_SEND_TIMEOUT
Returns the time-out value, in milliseconds, to use for Internet data send requests. If a data send request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_DATA_RECEIVE_TIMEOUT
Returns the time-out value, in milliseconds, to use for Internet data receive requests. If a data receive request takes longer than this time-out value, the request is canceled. The default time-out value 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_FTP

INTERNET_HANDLE_TYPE_CONNECT_GOPHER

INTERNET_HANDLE_TYPE_CONNECT_HTTP

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.

INTERNET_OPTION_READ_BUFFER_SIZE

Returns the size of the read buffer (for example, the buffer used by **FtpGetFile**).

INTERNET_OPTION_WRITE_BUFFER_SIZE

Returns the size of the write buffer (for example, the buffer used by **FtpPutFile**).

INTERNET_OPTION_ASYNC_PRIORITY

Returns the priority of this download if it is an asynchronous download.

INTERNET_OPTION_PARENT_HANDLE

Returns the parent handle of this handle.

INTERNET_OPTION_KEEP_CONNECTION

Returns an indication whether this handle uses

persistent connections. Can be one of these values:

INTERNET_KEEP_ALIVE_UNKNO
WN

INTERNET_KEEP_ALIVE_ENABL
ED

INTERNET_KEEP_ALIVE_DISAB
LED

INTERNET_OPTION_USERNAME

Returns the user name associated with a handle returned by **InternetConnect**.

INTERNET_OPTION_PASSWORD

Returns the password associated with the handle returned by **InternetConnect**.

INTERNET_OPTION_REQUEST_FLAGS

Returns special status flags about the current download in progress. The only flag that is returned at this time is INTERNET_REQFLAG_FROM_CACHE. This is the way for the caller to find out whether a request is being satisfied from the cache.

INTERNET_OPTION_EXTENDED_ERROR

Returns the Windows Sockets error code that was mapped to the ERROR_INTERNET_ error codes last returned in this thread context.

INTERNET_OPTION_SECURITY_CERTIFICATE_STRUCT

Returns the certificate for an SSL/PCT server into the INTERNET_CERTIFICATE_INFO structure.

INTERNET_OPTION_SECURITY_CERTIFICATE

Returns the certificate for an SSL/PCT server into a formatted string.

INTERNET_OPTION_SECURITY_KEY_BITNESS

Returns the bit size of the encryption key. The larger the number, the greater the encryption strength being used.

INTERNET_OPTION_OFFLINE_MODE

Not currently implemented.

INTERNET_OPTION_CACHE_STREAM_HANDLE

Returns the file handle being used to write the cached data.

INTERNET_OPTION_ASYNC

Not currently implemented.

INTERNET_OPTION_SECURITY_FLAGS

Returns the security flags for a handle. Can be a combination of these values:

SECURITY_FLAG_128BIT

SECURITY_FLAG_40BIT

SECURITY_FLAG_56BIT

SECURITY_FLAG_IETFSSL4

SECURITY_FLAG_IGNORE_CERT
_CN_INVALID

SECURITY_FLAG_IGNORE_CERT
_DATE_INVALID

SECURITY_FLAG_IGNORE_REDI
RECT_TO_HTTP

SECURITY_FLAG_IGNORE_REDI
RECT_TO_HTTPS

SECURITY_FLAG_NORMALBITNE
SS

SECURITY_FLAG_PCT

SECURITY_FLAG_PCT4

SECURITY_FLAG_SECURE

SECURITY_FLAG_SSL

SECURITY_FLAG_SSL3

SECURITY_FLAG_UNKNOWNBIT

INTERNET_OPTION_DATAFILE_NAME

Returns the name of the file backing a downloaded entity.

INTERNET_OPTION_URL

Returns the full URL of a downloaded entity.

INTERNET_OPTION_REFRESH

Returns TRUE if variables are allowed to be re-read from the registry for a handle.

INTERNET_OPTION_PROXY

Returns the proxy information on an existing **InternetOpen** handle when the process handle is not NULL. If the process handle is NULL, the API sets or queries the global proxy information. The *lpBuffer* parameter is an INTERNET_PROXY_INFO structure that contains the proxy information.

INTERNET_OPTION_VERSION

Returns the version number of Wininet.dll. The *lpBuffer* parameter is the address of an INTERNET_VERSION_INFO structure.

INTERNET_OPTION_USER_AGENT

Returns the user agent string on handles supplied by **InternetOpen** and used in a subsequent **HttpSendRequest**, so long as it is not overridden by a header added by **HttpAddRequestHeaders** or **HttpSendRequest**.

*lpBuffer*

Address of a buffer that receives the option setting.

*lpdwBufferLength*
Address of a variable that contains the length of *lpBuffer*. When the function returns, this parameter receives the length of the data placed into *lpBuffer*. If GetLastError returns ERROR_INSUFFICIENT_BUFFER, this parameter receives the number of bytes required to hold the created URL.

See also FtpGetFile, FtpPutFile, InternetConnect, InternetOpen, InternetSetOption

## InternetReadFile

```
BOOL InternetReadFile(
    IN HINTERNET hFile,
    IN LPVOID lpBuffer,
    IN DWORD dwNumberOfBytesToRead,
    OUT LPDWORD lpNumberOfBytesRead
);
```

Reads data from a handle opened by the InternetOpenUrl, FtpOpenFile, GopherOpenFile, or HttpOpenRequest function.

- Returns TRUE if successful or FALSE otherwise. To get extended error information, call GetLastError. An application can also use InternetGetLastResponseInfo when necessary.

*hFile*
Valid handle returned from a previous call to InternetOpenUrl, FtpOpenFile, GopherOpenFile, or HttpOpenRequest.

*lpBuffer*
Address of a buffer that receives the data read.

*dwNumberOfBytesToRead*
Number of bytes to read.

*lpNumberOfBytesRead*
Address of a variable that receives the number of bytes read. The InternetReadFile function sets this value to zero before doing any work or error checking.

If the return value is TRUE and the number of bytes read is zero, the transfer has been completed and there are no more bytes to read on the handle. This is analogous to reaching EOF in a local file. Call InternetCloseHandle to free up the connection to the server.

The buffer pointed to by *lpBuffer* is not always filled by calls to InternetReadFile (sufficient data may not have arrived from the server). When reading HTML data, for the first read posted, the buffer must be large enough to hold the HTML headers. When reading HTML-encoded directory entries, the buffer must be large enough to hold at least one complete entry.

When the item being read is also being cached by a Win32 Internet function, the application must ensure that a read is made for end-of-file so the cache file is committed correctly.

This function always fulfills the user's request. If more data is requested than is available, the function waits until enough data to complete the request is available. The only time that less data is returned than requested is when the end of the file has been reached.

This function can also be used to retrieve FTP and Gopher directory listings as an HTML document on a handle opened by **InternetOpenUrl**. The INTERNET_FLAG_RAW_DATA value must not have been specified in the call to **InternetOpenUrl**.

See also **FtpOpenFile**, **GopherOpenFile**, **HttpOpenRequest**, **InternetCloseHandle**, **InternetOpenUrl**

## InternetSetFilePointer

```
BOOL InternetSetFilePointer(
    IN HINTERNET hFile,
    IN LONG lDistanceToMove,
    IN PVOID pReserved,
    IN DWORD dwMoveMethod,
    IN DWORD dwContext
);
```

Sets a file position for **InternetReadFile**. This is a synchronous call; however, subsequent calls to **InternetReadFile** may block or return pending if the data is not available from the cache and the server does not support random access.

- Returns the current file position if the function succeeds, or -1 otherwise.

*hFile*
Valid handle returned from a previous call to **FtpOpenFile**, **GopherOpenFile**, **InternetOpenUrl** on an HTTP URL, or to **HttpOpenRequest** (using the GET or HEAD method and passed to **HttpSendRequest**). This handle must not have been created with the INTERNET_FLAG_DONT_CACHE or INTERNET_FLAG_NO_CACHE_WRITE value set.

*lDistanceToMove*
Number of bytes to move the file pointer. A positive value moves the pointer forward in the file, and a negative value moves it backward.

*pReserved*
Reserved; must be NULL.

*dwMoveMethod*
Starting point for the file pointer move. Can be one of these values:

| | |
|---|---|
| FILE_BEGIN | The starting point is zero or the beginning of the file. If FILE_BEGIN is specified, *lDistanceToMove* is interpreted as an unsigned location for the new file pointer. |
| FILE_CURRENT | The current value of the file pointer is the starting point. |

FILE_END — The current end-of-file position is the starting point. This method fails if the content length is unknown.

*dwContext*
Reserved; must be zero.

See also FtpOpenFile, GopherOpenFile, HttpOpenRequest, HttpSendRequest, InternetOpenUrl, InternetReadFile

## InternetSetOption

```
BOOL InternetSetOption(
    IN HINTERNET hInternet OPTIONAL,
    IN DWORD dwOption,
    IN LPVOID lpBuffer,
    IN DWORD dwBufferLength
);
```

Sets an Internet option on the specified handle.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call GetLastError.

*hInternet*
Internet handle on which to set information.

*dwOption*
Internet option to set. Can be a combination of these values:

INTERNET_OPTION_CALLBACK
Sets the address of the callback function defined for this handle.

INTERNET_OPTION_CONNECT_TIMEOUT
Sets the time-out value, in milliseconds, to use for Internet connection requests. If a connection request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_CONNECT_RETRIES
Sets the retry count to use for Internet connection requests. If a connection attempt still fails after the specified number of tries, the request is canceled. The default is five retries.

INTERNET_OPTION_CONNECT_BACKOFF
Sets the delay value, in milliseconds, to wait between connection retries.

INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT
Sets the time-out value, in milliseconds, to use for non-data (control) Internet receive requests. If a non-data receive request takes longer than this time-out value, the request is canceled. The default time-out is infinite. Currently, this value has meaning

only for FTP sessions.

INTERNET_OPTION_CONTROL_SEND_TIMEOUT

Sets the time-out value, in milliseconds, to use for non-data (control) Internet send requests. If a non-data send request takes longer than this time-out value, the request is canceled. The default time-out value is infinite. Currently, this value has meaning only for FTP sessions.

INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT

Sets the time-out value, in milliseconds, to use for non-data (control) Internet receive requests. If a non-data receive request takes longer than this time-out value, the request is canceled. The default time-out value is infinite. Currently, this value has meaning only for FTP sessions.

INTERNET_OPTION_DATA_SEND_TIMEOUT

Sets the time-out value, in milliseconds, to use for data Internet send requests. If a data send request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_DATA_RECEIVE_TIMEOUT

Sets the time-out value, in milliseconds, to use for data Internet receive requests. If a data receive request takes longer than this time-out value, the request is canceled. The default time-out value is infinite.

INTERNET_OPTION_ASYNC_PRIORITY

Sets the priority of this download if it is an asynchronous download. This option has not been implemented.

INTERNET_OPTION_CONTEXT_VALUE

Sets the context value associated with this Internet handle. Currently, this sets the context value to the address stored in the pointer, DWORD(lpBuffer). For Internet Explorer 4.0, this will be corrected so that the value stored in the buffer will be used, and this flag will be reassigned a new value. The old value, 10, will be preserved so that applications that are written for the old behavior will still be supported.

INTERNET_OPTION_REFRESH

Allows variables to be re-read from the registry for a handle.

INTERNET_OPTION_PROXY

Sets the proxy information on an existing **InternetOpen** handle when the process handle is not NULL. If the process handle is NULL, the API sets or queries the global proxy information. The *lpBuffer* parameter is an INTERNET_PROXY_INFO structure that contains the proxy information.

INTERNET_OPTION_USER_AGENT

Sets the user agent string on handles supplied by **InternetOpen** and used in subsequent

HttpSendRequest functions, as long as it is not overridden by a header added by HttpAddRequestHeaders or HttpSendRequest.

INTERNET_OPTION_USERNAME

Sets the user name associated with a handle returned by InternetConnect.

INTERNET_OPTION_PASSWORD

Sets the password associated with a handle returned by InternetConnect.

INTERNET_OPTION_READ_BUFFER_SIZE

Sets the size, in bytes, of the buffer to use to read the data.

INTERNET_OPTION_WRITE_BUFFER_SIZE

Sets the size of the write buffer (for example, the buffer used by FtpPutFile).

INTERNET_OPTION_WRITE_DATA

Sets the size, in bytes, of the buffer to use while writing out the data.

*lpBuffer*

Address of a buffer that contains the option setting.

*dwBufferLength*

Length of the *lpBuffer* buffer.

See also FtpGetFile, FtpPutFile, InternetConnect, InternetOpen, InternetQueryOption

## InternetSetOptionEx

```
BOOL InternetSetOptionEx(
    IN HINTERNET hInternet OPTIONAL,
    IN DWORD dwOption,
    IN LPVOID lpBuffer,
    IN DWORD dwBufferLength,
    IN DWORD dwFlags
);
```

Sets an Internet option on the specified handle.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call GetLastError.

*hInternet*

Internet handle on which to set information.

*dwOption*

Internet option to set. For a list of possible values, see InternetSetOption.

*lpBuffer*

Address of a buffer that contains the option setting.

*dwBufferLength*
Length of the *lpBuffer* buffer.

*dwFlags*
Action flags. Can be one of these values:

| Value | Meaning |
|---|---|
| ISO_GLOBAL | Modify the option globally. |
| ISO_REGISTRY | Write the option to the registry (where applicable). |

## InternetSetStatusCallback

```
INTERNET_STATUS_CALLBACK InternetSetStatusCallback(
    IN HINTERNET hInternet,
    IN INTERNET_STATUS_CALLBACK lpfnInternetCallback
);
```

Sets up a callback function that Win32 Internet functions can call as progress is made during an operation.

- Returns the previously defined status callback function if successful, NULL if there was no previously defined status callback function, or INTERNET_INVALID_STATUS_CALLBACK if the callback function is not valid.

*hInternet*
Handle for which the callback is to be set.

*lpfnInternetCallback*
Address of the callback function to call when progress is made, or to return NULL to remove the existing callback function. For more information about the callback function, see **InternetStatusCallback**.

Both synchronous and asynchronous functions use the callback function to indicate the progress of the request, such as resolving a name, connecting to a server, and so on. The callback function is required for an asynchronous operation. The asynchronous request will call back to the application with INTERNET_STATUS_REQUEST_COMPLETE to indicate the request has been completed.

A callback function can be set on any handle, and is inherited by derived handles. A callback function can be changed using **InternetSetStatusCallback**, providing there are no pending requests that need to use the previous callback value. Note, however, that changing the callback function on a handle does not change the callbacks on derived handles, such as that returned by **InternetConnect**. You must change the callback function at each level.

Many of the Win32 Internet functions perform several operations on the network. Each operation can take time to complete, and each can fail.

It is sometimes desirable to display status information during a long-term operation. You can display status information by setting up an Internet status callback function that cannot be removed as long as any callbacks or any asynchronous functions are pending.

After initiating InternetSetStatusCallback, it can be accessed from within any Win32 Internet function for monitoring time-intensive network operations.

## InternetStatusCallback

```
VOID InternetStatusCallback(
    IN HINTERNET hInternet,
    IN DWORD dwContext,
    IN DWORD dwInternetStatus,
    IN LPVOID lpvStatusInformation OPTIONAL,
    IN DWORD dwStatusInformationLength
);
```

This is a placeholder for the application-defined status callback

- No return value.

*hInternet*
Handle for which the callback function is being called.

*dwContext*
Application-defined context value associated with *hInternet*

*dwInternetStatus*
Status code that indicates why the callback function is being called. Can be one of these values:

INTERNET_STATUS_RESOLVING_NAME
Looking up the IP address of the name contained in *lpvStatusInformation*.

INTERNET_STATUS_NAME_RESOLVED
Successfully found the IP address of the name contained in *lpvStatusInformation*.

INTERNET_STATUS_CONNECTING_TO_SERVER
Connecting to the socket address (SOCKADDR) pointed to by *lpvStatusInformation*.

INTERNET_STATUS_CONNECTED_TO_SERVER
Successfully connected to the socket address (SOCKADDR) pointed to by *lpvStatusInformation*.

INTERNET_STATUS_SENDING_REQUEST
Sending the information request to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_ REQUEST_SENT
Successfully sent the information request to the server. The *lpvStatusInformation* parameter points to a DWORD containing the number of bytes sent.

INTERNET_STATUS_RECEIVING_RESPONSE
Waiting for the server to respond to a request. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_RESPONSE_RECEIVED

Successfully received a response from the server. The *lpvStatusInformation* parameter points to a DWORD containing the number of bytes received.

INTERNET_STATUS_REDIRECT

Indicates that an HTTP request is about to automatically redirect the request. The *lpvStatusInformation* parameter points to the new URL. At this point, the application may read any data returned by the server with the redirect response, and may query the response headers. It may also cancel the operation by closing the handle. This callback is not made if the original request specified INTERNET_FLAG_NO_AUTO_REDIRECT.

INTERNET_STATUS_CLOSING_CONNECTION

Closing the connection to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_CONNECTION_CLOSED

Successfully closed the connection to the server. The *lpvStatusInformation* parameter is NULL.

INTERNET_STATUS_HANDLE_CREATED

Used by **InternetConnect** to indicate it has created the new handle. This lets the application call **InternetCloseHandle** from another thread, if the connect is taking too long.

INTERNET_STATUS_HANDLE_CLOSING

This handle value is now terminated

INTERNET_STATUS_REQUEST_COMPLETE

An asynchronous operation has been completed. See **InternetOpen** for details on INTERNET_FLAG_ASYNC.

The *lpvStatusInformation* parameter points to an INTERNET_ASYNC_RESULT structure. The *dwStatusInformationLength* parameter contains the final completion status of the asynchronous function. If this is ERROR_INTERNET_EXTENDED_ERROR, the application can retrieve the server error information by using **InternetGetLastResponseInfo**.

*lpvStatusInformation*

Address of a buffer that contains information pertinent to this call to the callback function.

*dwStatusInformationLength*

Size of the *lpvStatusInformation* buffer.

In the case of INTERNET_STATUS_REQUEST_COMPLETE, *lpvStatusInformation* is the address of an INTERNET_ASYNC_RESULT structure.

An application uses the callback function to indicate the progress of synchronous and asynchronous functions, and to indicate the completion of an asynchronous request.

Because callbacks are made during processing of the request, the application should spend as little time as possible in the callback function to avoid degrading data throughput on the network. For example,

displaying a dialog box in a callback function may be such a lengthy operation that the server terminates the request.

The callback function may be called in a thread context different from the thread that initiated the request.

See also **InternetCloseHandle**, **InternetConnect**, **InternetGetLastResponseInfo**, **InternetOpen**

## InternetTimeFromSystemTime

```
BOOL InternetTimeFromSystemTime(
    IN  CONST SYSTEMTIME *pst,
    IN  DWORD dwRFC,
    OUT LPSTR lpszTime,
    IN  DWORD cbTime
);
```

Formats a date and time according to the specified RFC format (as specified in the HTTP version 1.0 specification).

- Returns TRUE if the function succeeds, or FALSE otherwise. To get extended error information, call **GetLastError**.

***pst***
Address of a **SYSTEMTIME** structure that contains the date and time to format.

***dwRFC***
RFC format.

***lpszTime***
Address of a buffer that receives the formatted data and time.

***cbTime***
Size, in bytes, of the ***lpszTime*** buffer.

## InternetTimeToSystemTime

```
BOOL InternetTimeToSystemTime(
    IN LPCSTR lpszTime,
    OUT SYSTEMTIME *pst,
    IN DWORD dwReserved
);
```

Takes an HTTP time/date string and converts it to a **SYSTEMTIME** structure.

- Returns TRUE if the string was converted, or FALSE otherwise. To get extended error information, call **GetLastError**.

***lpszTime***
Pointer to a null-terminated date/time string to convert

***pst***
Address of the pointer to the converted time.

*dwReserved*
Reserved; must be zero.

## InternetWriteFile

```
BOOL InternetWriteFile(
    IN HINTERNET hFile,
    IN LPCVOID lpBuffer,
    IN DWORD dwNumberOfBytesToWrite,
    OUT LPDWORD lpdwNumberOfBytesWritten
);
```

Writes data to an open Internet file.

- Returns TRUE if the function succeeds, or FALSE otherwise. To get extended error information, call **GetLastError**. An application can also use **InternetGetLastResponseInfo**, when necessary.

*hFile*
Valid handle returned from a previous call to **FtpOpenFile**.

*lpBuffer*
Address of a buffer that contains the data to be written to the file.

*dwNumberOfBytesToWrite*
Number of bytes to write to the file.

*lpdwNumberOfBytesWritten*
Address of a variable that receives the number of bytes written to the buffer. The **InternetWriteFile** function sets this value to zero before doing any work or error checking.

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

See also **FtpOpenFile**, **InternetCloseHandle**

# FTP Functions

The FTP functions deal with FTP file and directory manipulation and navigation.

## FtpCreateDirectory

```
BOOL FtpCreateDirectory(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszDirectory
);
```

Creates a new directory on the FTP server.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call **GetLastError**. If the error code indicates that the

FTP server denied the request to create a directory, use **InternetGetLastResponseInfo** to determine why.

*hFtpSession*
Valid handle to an FTP session.

*lpszDirectory*
Address of a null-terminated string that contains the name of the directory to create on the remote system. This can be either a fully qualified path name or a name relative to the current directory.

An application should use **FtpGetCurrentDirectory** to determine the remote site's current working directory, instead of assuming that the remote system uses a hierarchical naming scheme for directories.

The *lpszDirectory* parameter can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The **FtpCreateDirectory** function translates the directory name separators to the appropriate character before they are used.

## FtpDeleteFile

```
BOOL FtpDeleteFile(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszFileName
);
```

Deletes a file stored on the FTP server.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call **GetLastError**.

*hFtpSession*
Valid handle to an FTP session.

*lpszFileName*
Address of a null-terminated string that contains the name of the file to delete on the remote system.

The *lpszFile* parameter can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The **FtpDeleteFile** function translates the directory name separators to the appropriate character before they are used.

## FtpFindFirstFile

```
HINTERNET FtpFindFirstFile(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszSearchFile OPTIONAL,
    OUT LPWIN32_FIND_DATA lpFindFileData,
    IN DWORD dwFlags
    IN DWORD dwContext
```

```
);
```

Searches the specified directory of the given FTP session. File and directory entries are returned to the application in the WIN32_FIND_DATA structure.

- Returns a valid handle for the request if the directory enumeration was started successfully; otherwise, returns NULL. To get a specific error code, call GetLastError. If the function finds no matching files, GetLastError returns ERROR_NO_MORE_FILES.

*hFtpSession*
Valid handle to an FTP session returned from InternetConnect.

*lpszSearchFile*
Address of a null-terminated string that specifies a valid directory path name or file name for the FTP server's file system. If the value of *lpszSearchFile* is NULL or if it is an empty string, it will find the first file in the current directory on the server.

*lpFindFileData*
Address of a WIN32_FIND_DATA structure that receives information about the found file or directory.

*dwFlags*
Application-defined value that associates this search with any application. For a description of the values, see InternetOpenUrl.

*dwContext*
Application-defined value that associates this search with any application data. This parameter is used only if the application has already called InternetSetStatusCallback to set up a status callback function.

This function enumerates both files and directories.

The FtpFindFirstFile function is similar to the Win32 FindFirstFile function. Note, however, that only one FtpFindFirstFile can occur at a time within a given FTP session. The enumerations, therefore, are correlated with the FTP session handle. This is because the FTP protocol allows only a single directory enumeration per session.

After calling FtpFindFirstFile and until calling InternetCloseHandle, the application cannot call FtpFindFirstFile again on a given FTP session handle. If this happens, calls to the FtpFindFirstFile function will fail with error code ERROR_FTP_TRANSFER_IN_PROGRESS.

After beginning a directory enumeration with FtpFindFirstFile, the InternetFindNextFile function can be used to continue the enumeration.

The InternetCloseHandle function is used to close the handle returned from FtpFindFirstFile. If the InternetCloseHandle function closes the handle before InternetFindNextFile fails with ERROR_NO_MORE_FILES, the directory enumeration will be terminated.

Because the FTP protocol provides no standard means of enumerating, some of the common information about files, such as file creation date and time, is not always available or correct. When this happens,

**FtpFindFirstFile** and **InternetFindNextFile** fill in unavailable information with a "best guess" based on available information. For example, creation and last access dates will often be the same as the file's modification date.

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

See also **FtpOpenFile**, **InternetCloseHandle**, **InternetFindNextFile**, **InternetSetStatusCallback**

## FtpGetCurrentDirectory

```
BOOL FtpGetCurrentDirectory(
    IN HINTERNET hFtpSession,
    OUT LPCTSTR lpszCurrentDirectory,
    IN OUT LPDWORD lpdwCurrentDirectory
);
```

Retrieves the current directory for the specified FTP session.

- Returns TRUE if successful, or FALSE otherwise. To get the specific error code, call **GetLastError**. If the error code indicates that the FTP server denied the request to change to a directory, use **InternetGetLastResponseInfo** to determine why.

*hFtpSession*
Valid handle to an FTP session.

*lpszCurrentDirectory*
Address of a buffer that receives the current directory string, which specifies the absolute path to the current directory. The string is null terminated.

*lpdwCurrentDirectory*
Address of a variable that 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 is sufficient for all path names. When the function returns, this parameter receives the number of characters copied into the buffer.

If the *lpszCurrentDirectory* buffer is not large enough, *lpdwCurrentDirectory* receives the number of bytes required to retrieve the full, current directory name.

## FtpGetFile

```
BOOL FtpGetFile(
    IN HINTERNET hFtpSession,
    IN LPCSTR lpszRemoteFile,
    IN LPCSTR lpszNewFile,
    IN BOOL fFailIfExists,
    IN DWORD dwFlagsAndAttributes,
```

```
    IN DWORD dwFlags,
    IN DWORD dwContext
);
```

Retrieves a file from the FTP server and stores it under the specified file name, creating a new local file in the process.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call GetLastError.

*hFtpSession*
Valid handle to an FTP session.

*lpszRemoteFile*
Address of a null-terminated string that contains the name of the file to retrieve from the remote system.

*lpszNewFile*
Address of a null-terminated string that contains the name of the file to create on the local system.

*fFailIfExists*
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*
File attributes for the new file. Can be any combination of the FILE_ATTRIBUTE_* flags used by CreateFile. See CreateFile in the Win32 SDK for more information on FILE_ATTRIBUTE_* attributes.

*dwFlags*
Flags that control how the function will handle the file download. The first set of flag values indicates the conditions under which the transfer occurs. These transfer type flags can be used in combination with the second set of flags that control caching. The application can select one of these transfer type values:

FTP_TRANSFER_TYPE_ASCII
Transfer the file using FTP's ASCII (Type A) transfer method. Control and formatting information is converted to local equivalents.

FTP_TRANSFER_TYPE_BINARY
Transfer the file 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.

INTERNET_FLAG_TRANSFER_ASCII
Transfer the file as ASCII.

INTERNET_FLAG_TRANSFER_BINARY
Transfer the file as binary.

The following flags handle how the caching of this file will be handled. Any combination of the following flags can be used with the transfer type flag. The possible values are:

INTERNET_FLAG_DONT_CACHE

Do not add the returned entity to the cache.

INTERNET_FLAG_HYPERLINK

Force a reload if there was no Expires time and no Last-Modified time returned from the server when determining whether to reload the item from the network.

INTERNET_FLAG_MAKE_PERSISTENT

Add the returned entity to the cache as a persistent entity. This means that standard cache cleanup, consistency checking, or garbage collection cannot remove this item from the cache.

INTERNET_FLAG_MUST_CACHE_REQUEST

Cause the operation to fail if the downloaded file cannot be cached.

INTERNET_FLAG_NO_CACHE_WRITE

Do not add the returned entity to the cache.

INTERNET_FLAG_RELOAD

Force a download of the requested file, object, or directory listing from the origin server, not from the cache.

INTERNET_FLAG_RESYNCHRONIZE

Cause the FTP resource to be reloaded from the server.

*dwContext*

Application-defined value that associates this search with any application data. This is used only if the application has already called **InternetSetStatusCallback** to set up a status callback function.

The **FtpGetFile** function 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. An application that needs to retrieve file data only or that requires close 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 converts control and formatting characters to local equivalents. The default transfer is binary mode, where the file is downloaded in the same format as it is stored on the server.

Both *lpszRemoteFile* and *lpszNewFile* can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The **FtpGetFile** function translates the directory name separators to the appropriate character before they are used.

## FtpOpenFile

```
HINTERNET FtpOpenFile(
    IN HINTERNET hFtpSession,
    IN LPCSTR lpszFileName,
    IN DWORD fdwAccess,
    IN DWORD dwFlags,
    IN DWORD dwContext
```

```
);
```

Initiates access to a remote file for writing or reading.

- Returns a handle if successful. Otherwise, the function returns NULL. To get a specific error code, call GetLastError.

*hFtpSession*
Valid handle to an FTP session.

*lpszFileName*
Address of a null-terminated string that contains the name of the file to access on the remote system.

*fdwAccess*
Value that determines how the file will be accessed. This can be GENERIC_READ or GENERIC_WRITE, but not both.

*dwFlags*
Conditions under which the transfers occur. The application should select one transfer type and any of the flags that control how the caching of the file will be controlled. The transfer type can be any one of the following values:

FTP_TRANSFER_TYPE_ASCII
Transfer the file using FTP's ASCII (Type A) transfer method. Control and formatting information is converted to local equivalents.

FTP_TRANSFER_TYPE_BINARY
Transfer the file 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.

INTERNET_FLAG_TRANSFER_ASCII
Transfer the file as ASCII.

INTERNET_FLAG_TRANSFER_BINARY
Transfer the file as binary.

The application can use one or more of the following values to control the caching of the file:

INTERNET_FLAG_HYPERLINK
Force a reload if there was no Expires time and no Last-Modified time returned from the server when determining whether to reload the item from the network.

INTERNET_FLAG_MAKE_PERSISTENT
Add the returned entity to the cache as a persistent entity. This means that standard cache cleanup, consistency checking, or garbage collection cannot remove this item from the cache.

INTERNET_FLAG_MUST_CACHE_REQUEST
Cause the operation to fail if the downloaded file cannot be cached.

INTERNET_FLAG_RELOAD
Force a download of the requested file, object, or directory listing from the origin server, not from the cache.

INTERNET_FLAG_RESYNCHRONIZE

Cause the FTP resource to be reloaded from the server.

*dwContext*

Application-defined value that associates this search with any application data. This is only used if the application has already called **InternetSetStatusCallback** to set up a status callback function.

The **FtpOpenFile** function should be used in the following situations:

- An application has data it needs to send to an FTP server to be created as a file on the FTP server, but the application does not have a local file containing the data. After the file is opened with **FtpOpenFile**, the application uses **InternetWriteFile** to send the FTP file data to the server.
- An application needs to retrieve a file from the server into application-controlled memory, instead of writing the file to disk. The application uses **InternetReadFile** after using **FtpOpenFile** to open the file.
- An application needs a fine level of control over a file transfer. For example, the application may need to display a "thermometer" when downloading a file to indicate to the user that the file transfer is or is not proceeding correctly.

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

Only one file can be open in a single FTP session. Therefore, no file handle is returned and the application simply uses the FTP session handle when necessary.

The *lpszFile* parameter can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The **FtpOpenFile** function translates the directory name separators to the appropriate character before they are used.

The **InternetCloseHandle** function is used to close the handle returned from **FtpOpenFile**. If the **InternetCloseHandle** function closes the handle before all the data has been transferred, the transfer is terminated.

## FtpPutFile

```
BOOL FtpPutFile(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszLocalFile,
    IN LPCTSTR lpszNewRemoteFile,
    IN DWORD dwFlags,
```

```
    IN DWORD dwContext
);
```

Stores a file on the FTP server.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call GetLastError.

*hFtpSession*
Valid handle to an FTP session.

*lpszLocalFile*
Address of a null-terminated string that contains the name of the file to send from the local system.

*lpszNewRemoteFile*
Address of a null-terminated string that contains the name of the file to create on the remote system.

*dwFlags*
Conditions under which the transfer occurs. Can be any combination of FTP_TRANSFER_* defined constants. For further information on the FTP_TRANSFER_* constants, see FtpOpenFile.

*dwContext*
Application-defined value that associates this search with any application data. This parameter is used only if the application has already called InternetSetStatusCallback to set up a status callback.

The FtpPutFile function is a high-level routine that handles all the bookkeeping and overhead associated with reading a file locally and storing it on an FTP server. An application that needs to send file data only, or that requires close 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 converts control and formatting characters to local equivalents.

Both *lpszNewRemoteFile* and *lpszLocalFile* can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The FtpPutFile function translates the directory name separators to the appropriate character before they are used.

## FtpRemoveDirectory

```
BOOL FtpRemoveDirectory(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszDirectory
);
```

Removes the specified directory on the FTP server.

- Returns TRUE if successful, or FALSE otherwise. To get the specific error code, call GetLastError. If the error code indicates that the FTP server denied the request to remove a directory, use InternetGetLastResponseInfo to determine why.

*hFtpSession*
Valid handle to an FTP session.

*lpszDirectory*
Address of a null-terminated string that contains the name of the directory to remove on the remote system. This can be either a fully qualified path name or a name relative to the current directory.

An application should use FtpGetCurrentDirectory to determine the remote site's current working directory, instead of assuming that the remote system uses a hierarchical naming scheme for directories.

The *lpszDirectory* parameter can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The FtpRemoveDirectory function translates the directory name separators to the appropriate character before they are used.

## FtpRenameFile

```
BOOL FtpRenameFile(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszExisting,
    IN LPCTSTR lpszNew
);
```

Renames a file stored on the FTP server.

- Returns TRUE if successful, or FALSE otherwise. To get a specific error code, call GetLastError.

*hFtpSession*
Valid handle to an FTP session.

*lpszExisting*
Address of a null-terminated string that contains the name of the file that will have its name changed on the remote FTP server.

*lpszNew*
Address of a null-terminated string that contains the new name for the remote file.

The *lpszExisting* and *lpszNew* parameters can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The FtpRenameFile function translates the directory name separators to the appropriate character before they are used.

## FtpSetCurrentDirectory

```
BOOL FtpSetCurrentDirectory(
    IN HINTERNET hFtpSession,
    IN LPCTSTR lpszDirectory
);
```

Changes to a different working directory on the FTP server.

- Returns TRUE if successful, or FALSE otherwise. To get the specific error code, call **GetLastError**. If the error code indicates that the FTP server denied the request to change a directory, use **InternetGetLastResponseInfo** to determine why.

*hFtpSession*
Valid handle to an FTP session.

*lpszDirectory*
Address of a null-terminated string that contains the name of the directory to change to on the remote system. This can be either a fully qualified path name or a name relative to the current directory.

An application should use **FtpGetCurrentDirectory** to determine the remote site's current working directory, instead of assuming that the remote system uses a hierarchical naming scheme for directories.

The *lpszDirectory* parameter can be either partially or fully qualified file names relative to the current directory. A backslash (\) or forward slash (/) can be used as the directory separator for either name. The **FtpSetCurrentDirectory** function translates the directory name separators to the appropriate character before they are used.

# ›pher Functions

The Gopher functions control the creation and use of the Internet Gopher utilities.

## GopherCreateLocator

```
BOOL GopherCreateLocator(
    IN LPCTSTR lpszHost,
    IN INTERNET_PORT nServerPort,
    IN LPCTSTR lpszDisplayString OPTIONAL,
    IN LPCTSTR lpszSelectorString OPTIONAL,
    IN DWORD dwGopherType,
    OUT LPCTSTR lpszLocator OPTIONAL,
    IN OUT LPDWORD lpdwBufferLength
);
```

Creates a Gopher or Gopher+ locator string from its component parts.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call **GetLastError** or **InternetGetLastResponseInfo**.

*lpszHost*
Address of a string that contains the name of the host, or a dotted-decimal IP address (such as 198.105.232.1).

*nServerPort*
Number of the port on which the Gopher server at *lpszHost* lives, in host byte order. If *nServerPort* is INTERNET_INVALID_PORT_NUMBER, the default Gopher port is read from the \etc\services file.

*lpszDisplayString*
Gopher document or directory to be displayed. If this parameter is NULL, the function returns the default directory for the Gopher server.

*lpszSelectorString*
Address of the selector string to send to the Gopher server in order to retrieve information. This parameter can be NULL.

*dwGopherType*
Value that specifies whether *lpszSelectorString* refers to a directory or document, and whether the request is Gopher+ or Gopher. For more information, see the GOPHER_FIND_DATA structure.

*lpszLocator*
Address of a buffer that receives the locator string. If *lpszLocator* is NULL, *lpdwBufferLength* receives the needed buffer length, but the function performs no other processing.

*lpdwBufferLength*
Length of the *lpszLocator* buffer. When the function returns, this parameter receives the number of bytes written to the *lpszLocator* buffer. If GetLastError returns ERROR_INSUFFICIENT_BUFFER, this parameter receives the number of bytes required to form the locator successfully.

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 normally used for calls to the GopherFindFirstFile function to retrieve a specific piece of information.

## GopherGetLocatorType

```
BOOL GopherGetLocatorType(
    IN LPCTSTR lpszLocator,
    OUT LPDWORD lpdwGopherType
);
```

Parses a Gopher locator and determines its attributes.

*lpszLocator*
Address of the Gopher locator string to parse.

*lpdwGopherType*
Address of a variable that receives the type of the locator. The type is a bitmask that consists of a combination of the following values:

| Value | Meaning |
|---|---|
| GOPHER_TYPE_TEXT_FILE | An ASCII text file. |
| GOPHER_TYPE_DIRECTORY | A directory of additional Gopher items. |
| GOPHER_TYPE_CSO | A CSO telephone book server. |
| GOPHER_TYPE_ERROR | Indicator of an error condition. |
| GOPHER_TYPE_MAC_BINHEX | A Macintosh file in BINHEX format. |
| GOPHER_TYPE_DOS_ARCHIVE | An MS-DOS® archive file. |
| GOPHER_TYPE_UNIX_UUENCODED | A UUENCODED file. |
| GOPHER_TYPE_INDEX_SERVER | An index server. |
| GOPHER_TYPE_TELNET | A Telnet Server. |
| GOPHER_TYPE_BINARY | A binary file. |
| GOPHER_TYPE_REDUNDANT | Indicator of a duplicated server. The information contained within is a duplicate of the primary server. The primary server is defined as the last directory entry that did not have a GOPHER_TYPE_REDUNDANT type. |
| GOPHER_TYPE_TN3270 | A TN3270 server. |
| GOPHER_TYPE_GIF | A GIF graphics file. |
| GOPHER_TYPE_IMAGE | An image file. |
| GOPHER_TYPE_BITMAP | A bitmap file. |
| GOPHER_TYPE_MOVIE | A movie file. |
| GOPHER_TYPE_SOUND | A sound file. |
| GOPHER_TYPE_HTML | An HTML document. |
| GOPHER_TYPE_PDF | A PDF file. |
| GOPHER_TYPE_CALENDAR | A calendar file. |
| GOPHER_TYPE_INLINE | An inline file. |
| GOPHER_TYPE_UNKNOWN | The item type is unknown. |
| GOPHER_TYPE_ASK | An Ask+ item. |
| GOPHER_TYPE_GOPHER_PLUS | A Gopher+ item. |

The GopherGetLocatorType function 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.

## GopherFindFirstFile

```
HINTERNET GopherFindFirstFile(
    IN HINTERNET hGopherSession,
IN LPCTSTR lpszLocator OPTIONAL,
```

```
    IN LPCTSTR lpszSearchString OPTIONAL,
    OUT LPGOPHER_FIND_DATA lpFindData OPTIONAL,
    IN DWORD dwFlags,
    IN DWORD dwContext
);
```

Uses a Gopher locator and some search criteria to create a session with the server and locate the requested documents, binary files, index servers, or directory trees.

- Returns a valid search handle if successful, or NULL otherwise. To get extended error information, call **GetLastError** or **InternetGetLastResponseInfo**.

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

*lpszLocator*
Name of the item to locate. Can be one of the following items:

- A Gopher locator returned by *lpGopherFindData*, or a locator obtained by a previous call to this function or the **InternetFindNextFile** function.
- A NULL pointer or zero-length string indicating that the topmost information from a Gopher server is being returned.
- A locator created by the **GopherCreateLocator** function.

*lpszSearchString*
Address of a buffer that contains the strings to search, if this request is to an index server. Otherwise, this parameter should be NULL.

*lpFindData*
Address of a GOPHER_FIND_DATA structure that receives the information retrieved by this function.

*dwFlags*
Action flag. For a list of the valid flag values, see **InternetOpenUrl**.

*dwContext*
Application-defined value that associates this search with any application data.

The **GopherFindFirstFile** function closely resembles the Win32 **FindFirstFile** function. 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 retrieve the first Gopher object in an enumeration, an application can use the **InternetFindNextFile** function 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 as closed pending. Any open sessions are terminated, and any data waiting for the caller is discarded. In addition, any allocated buffers are freed.

See also InternetCloseHandle, InternetConnect, InternetFindNextFile

## GopherGetAttribute

```
BOOL GopherGetAttribute(
    IN HINTERNET hGopherSession,
    IN LPCTSTR lpszLocator,
    IN LPCTSTR lpszAttributeName OPTIONAL,
    OUT LPBYTE lpBuffer,
    IN DWORD dwBufferLength,
    OUT LPDWORD lpdwCharactersReturned,
    IN GOPHER_ATTRIBUTE_ENUMERATOR lpfnEnumerator OPTIONAL,
    IN DWORD dwContext
);
```

Allows an application to retrieve specific attribute information from the server.

- Returns TRUE if the request is satisfied, or FALSE otherwise. To get extended error information, call GetLastError or InternetGetLastResponseInfo.

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

*lpszLocator*
Address of a string that identifies the item at the Gopher server on which to return attribute information.

*lpszAttributeName*
Address of a space-delimited string specifying the names of attributes to return. If *lpszAttributeName* is NULL, GopherGetAttribute will return information about all attributes.

*lpBuffer*
Address of an application-defined buffer from which attribute information is retrieved.

*dwBufferLength*
Size, in bytes, of the *lpBuffer* buffer.

*lpdwCharactersReturned*
Number of characters read into the *lpBuffer* buffer.

*lpfnEnumerator*
Address of a callback function that enumerates each attribute of the locator. This parameter is optional. If it is NULL, all the Gopher attribute information is placed into *lpBuffer*. If *lpfnEnumerator* is specified, the callback function is called once for each attribute of the object.

The callback function receives the address of a single GOPHER_ATTRIBUTE_TYPE structure with each call. The enumeration callback function allows the application to avoid having to parse the Gopher attribute information.

*dwContext*
Application-defined value that associates this operation with any application data.

Generally, applications call this function after calling **GopherFindFirstFile** or **InternetFindNextFile**.

The size of the *lpBuffer* parameter must be equal to or greater than the value of MIN_GOPHER_ATTRIBUTE_LENGTH (currently defined in Wininet.h as 256 bytes).

See also **InternetConnect**

## GopherAttributeEnumerator

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

Defines a callback function that processes attribute information from a Gopher server. This callback function is installed by a call to the **GopherGetAttribute** function.

- Returns TRUE to continue the enumeration, or FALSE to immediately stop it. This function is primarily used for returning the results of a Gopher+ ASK item.

*lpAttributeInformation*
Address of a buffer that contains a single GOPHER_ATTRIBUTE_TYPE structure. The *lpBuffer* parameter to **GopherGetAttribute** is used for storing this structure. The *lpBuffer* size must be equal to or greater than the value of MIN_GOPHER_ATTRIBUTE_LENGTH.

*dwError*
Error value. This parameter is NO_ERROR if the attribute was parsed and written to the buffer successfully. If a problem was encountered, an error value is returned.

## GopherOpenFile

```
HINTERNET GopherOpenFile(
    IN HINTERNET hGopherSession,
    IN LPCTSTR lpszLocator,
    IN LPCTSTR lpszView OPTIONAL,
    IN DWORD dwFlags,
    IN DWORD dwContext
);
```

Begins reading a Gopher data file from a Gopher server.

- Returns a handle if successful, or NULL if the file cannot be opened. To get extended error information, call GetLastError or InternetGetLastResponseInfo.

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

*lpszLocator*
Address of a string that identifies the file to open. Generally, this locator is returned from a call to GopherFindFirstFile or InternetFindNextFile. Because the Gopher protocol has no concept of a current directory, the locator is always fully qualified.

*lpszView*
Address of a string that describes the view to open if several views of the file exist at the server. If *lpszView* is NULL, the function uses the default file view.

*dwFlags*
Any combination of the INTERNET_FLAG_* flag values. For a list of valid flag values, see InternetOpenUrl.

*dwContext*
Application-defined value that associates this operation with any application data.

This function opens a file at a Gopher server. Because a file cannot be actually opened or locked at a server, this function simply associates location information with a handle that an application can use 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 as closed pending. Any open sessions are terminated, and any data waiting for the caller is discarded. In addition, any allocated buffers are freed.

See also GopherFindFirstFile, GopherGetAttribute, InternetCloseHandle, InternetConnect, InternetFindNextFile, InternetOpenUrl, InternetReadFile

# TTP Functions

The HTTP functions control the transmission and content of HTTP requests.

## HttpAddRequestHeaders

```
BOOL HttpAddRequestHeaders(
    IN HINTERNET hHttpRequest,
    IN LPCTSTR lpszHeaders,
    IN DWORD dwHeadersLength,
        IN DWORD dwModifiers
    );
```

Adds one or more HTTP request headers to the HTTP request handle.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError.

*hHttpRequest*
Open HTTP request handle returned by HttpOpenRequest.

*lpszHeaders*
Headers to append to the request. Each header must be terminated by a CR/LF (carriage return/line feed) pair.

*dwHeadersLength*
Length, in characters, of *lpszHeaders*. If this parameter is -1L, the function assumes that *lpszHeaders* is zero-terminated (ASCIIZ), and the length is computed.

*dwModifiers*
Values used to modify the semantics of this function. Can be a combination of these values:

HTTP_ADDREQ_FLAG_COALESCE_WITH_COMMA
Coalesces headers of the same name. For example, adding "Accept: text/*" followed by "Accept: audio/*" with this flag results in the formation of the single header "Accept: text/*, audio/*". This causes the first header found to be coalesced. It is up to the calling application to ensure a cohesive scheme with respect to coalesced/separate headers.

HTTP_ADDREQ_FLAG_COALESCE_WITH_SEMICOLON
Coaleses headers of the same name using a semicolon.

HTTP_ADDREQ_FLAG_COALESCE
Coaleses headers of the same name.

HTTP_ADDREQ_FLAG_REPLACE
Replaces or removes a header. If the header value is empty and the header is found, it is removed. If not empty, the header value is replaced.

HTTP_ADDREQ_FLAG_ADD
Adds the header if it does not exist. Used with REPLACE.

HTTP_ADDREQ_FLAG_ADD_IF_NEW
Adds the header only if it does not already exist; otherwise, an error is returned.

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

Note that for basic HttpAddRequestHeaders, the application can pass in multiple headers in a single buffer. If the application is trying to remove or replace a header, only one header can be supplied in *lpszHeaders.*

See also HttpOpenRequest, HttpSendRequest

# HttpOpenRequest

```
HINTERNET HttpOpenRequest(
    IN HINTERNET hHttpSession,
    IN LPCTSTR lpszVerb,
    IN LPCTSTR lpszObjectName,
    IN LPCTSTR lpszVersion,
    IN LPCTSTR lpszReferer OPTIONAL,
    IN LPCTSTR FAR * lpszAcceptTypes OPTIONAL,
    IN DWORD dwFlags,
    IN DWORD dwContext
);
```

Opens an HTTP request handle.

- Returns a valid (non-NULL) HTTP request handle if successful, or NULL otherwise. To get extended error information, call **GetLastError**.

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

***lpszVerb***
Address of a string that contains the verb to use in the request. If this parameter is NULL, the function uses "GET" as the verb.

***lpszObjectName***
Address of a string that contains the name of the target object of the specified verb. This is generally a file name, an executable module, or a search specifier.

***lpszVersion***
Address of a string that contains the HTTP version. If this parameter is NULL, the function uses "HTTP/1.0" as the version.

***lpszReferer***
Address of a string that specifies the address (URL) of the document from which the URL in the request (*lpszObjectName*) was obtained. If this parameter is NULL, no "referrer" is specified.

***lpszAcceptTypes***
Address of a null-terminated array of LPCTSTR pointers indicating content types accepted by the client. If this parameter is NULL, no types are accepted by the client. Servers interpret a lack of accept types to indicate that the client accepts only documents of type "text/*" (that is, only text documents and not pictures or other binary files).

***dwFlags***
Internet flag values. For a list of valid flag values, see **InternetOpenUrl**.

***dwContext***
An application-defined value that associates this operation with any application data.

This function creates a new HTTP request handle and stores the

specified parameters in that handle. An HTTP request handle holds a request to be sent to an HTTP server and 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** cancels all outstanding I/O on the handle.

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

See also **HttpAddRequestHeaders**, **HttpQueryInfo**, **HttpSendRequest**, **InternetCloseHandle**, **InternetConnect**, **InternetOpen**, **InternetReadFile**

## HttpQueryInfo

```
BOOL HttpQueryInfo(
    IN HINTERNET hHttpRequest,
    IN DWORD dwInfoLevel,
    IN LPVOID lpvBuffer OPTIONAL,
    IN LPDWORD lpdwBufferLength,
    IN OUT LPDWORD lpdwIndex OPTIONAL,
);
```

Queries for information about an HTTP request.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call **GetLastError**.

*hHttpRequest*
Open HTTP request handle returned by **HttpOpenRequest**.

*dwInfoLevel*
Combination of the attribute to query and the flags that modify the request. The following flags can be used to modify a request:

HTTP_QUERY_INFO_NUMBER
This flag is required to set the type of the data returned by **HttpQueryInfo** to a DWORD.

HTTP_QUERY_CUSTOM
If this query level is specified, *lpvBuffer* contains an ASCIIZ header name. This header name is searched for and its value returned in *lpvBuffer* on output.

HTTP_QUERY_FLAG_COALESCE
Combine the values from several headers of the same name into the output buffer.

HTTP_QUERY_FLAG_REQUEST_HEADERS
Typically, response headers are queried, but an application can

| | |
|---|---|
| | also query request headers by using this flag. |
| HTTP_QUERY_FLAG_SYSTEMTIME | For those headers whose value is a date/time string, such as "Last-Modified-Time", specifying this flag returns the header value as a standard Win32 SYSTEMTIME structure, which does not require the application to parse the data. |
| HTTP_QUERY_FLAG_NUMBER | For those headers whose value is a number, such as the status code, specifying this flag returns the data as a 32-bit number. |

*lpvBuffer*
Address of the buffer that receives the information.

*lpdwBufferLength*
Address of a value that contains the length of the data buffer. When the function returns, this parameter contains the address of a value specifying the length of the information written to the buffer. When the function returns strings, the following rules apply:

- If the function succeeds, *lpdwBufferLength* specifies the length of the string, in characters, minus 1 for the terminating null.
- If the function fails and ERROR_INSUFFICIENT_BUFFER is returned, *lpdwBufferLength* specifies the number of bytes that the application must allocate in order to receive the string.

*lpdwIndex*
Address of a zero-based header index used to enumerate multiple headers with the same name. When calling the function, this parameter is the index of the specified header to return. When the function returns, this parameter is the index of the next header. If the next index cannot be found, ERROR_HTTP_HEADER_NOT_FOUND is returned.

The possible values for the *dwInfoLevel* parameter include:

HTTP_QUERY_MIME_VERSION

HTTP_QUERY_CONTENT_TYPE

HTTP_QUERY_CONTENT_TRANSFER_ENCODI
NG

HTTP_QUERY_CONTENT_ID

HTTP_QUERY_CONTENT_DESCRIPTION

HTTP_QUERY_CONTENT_LENGTH

HTTP_QUERY_ALLOW

HTTP_QUERY_PUBLIC

HTTP_QUERY_DATE

HTTP_QUERY_EXPIRES

HTTP_QUERY_LAST_MODIFIED
HTTP_QUERY_MESSAGE_ID
HTTP_QUERY_URI
HTTP_QUERY_DERIVED_FROM
HTTP_QUERY_LANGUAGE
HTTP_QUERY_COST
HTTP_QUERY_WWW_LINK
HTTP_QUERY_PRAGMA
HTTP_QUERY_VERSION
HTTP_QUERY_STATUS_CODE
HTTP_QUERY_STATUS_TEXT
HTTP_QUERY_RAW_HEADERS
HTTP_QUERY_RAW_HEADERS_CRLF
HTTP_QUERY_REQUEST_METHOD

In HTTP_QUERY_REQUEST_METHOD, the *lpvBuffer* parameter receives the verb that is being used in the request, typically "GET" or "POST".

This function is used to return response or request headers from an HTTP request. You can retrieve different types of data from **HttpQueryInfo**:

- strings (default)
- SYSTEMTIME (for Data: Expires:, headers)
- DWORD (for STATUS_CODE, CONTENT_LENGTH, and so on if HTTP_QUERY_INFO_NUMBER has been used)

See also **HttpOpenRequest**

## HttpSendRequest

```
BOOL HttpSendRequest(
    IN HINTERNET hHttpRequest,
    IN LPCTSTR lpszHeaders OPTIONAL,
    IN DWORD dwHeadersLength,
    IN LPVOID lpOptional OPTIONAL,
    DWORD dwOptionalLength
);
```

Sends the specified request to the HTTP server.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call **GetLastError**.

*hHttpRequest*
Open HTTP request handle returned by **HttpOpenRequest**.

*lpszHeaders*
Additional headers to be appended to the request. This parameter can be NULL if there are no additional headers to append.

*dwHeadersLength*
Length, in characters, of the additional headers. If this parameter is -1L and *lpszHeaders* is not NULL, the function assumes that *lpszHeaders* is zero-terminated (ASCIIZ), and the length is calculated.

*lpOptional*
Address of any optional data to send immediately after the request headers. This is generally used for POST and PUT operations. This parameter can be NULL if there is no optional data to send.

*dwOptionalLength*
Length, in bytes, of the optional data. This parameter can be zero if there is no optional data to send.

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

The function also lets the client specify optional data to send to the HTTP server immediately following the request headers. This feature is generally used for "write" operations such as PUT and POST.

After the request is sent, the status code and response headers from the HTTP server are read. These headers are maintained internally and are available to client applications through the **HttpQueryInfo** function.

An application can use the same HTTP request handle in multiple calls to **HttpSendRequest**, but the application must read all data returned from the previous call before calling the function again.

See also **HttpOpenRequest**, **HttpQueryInfo**, **InternetReadFile**

# ookie Functions

Cookies are a means by which, under HTTP protocol, a server or a script can maintain state information on the client workstation. The Win32 Internet functions have implemented a persistent cookie database for this purpose. Cookie functions are provided for users of Win32 Internet functions in order to set cookies into, and access them from, the cookie database. The caller of these functions should be familiar with cookies as outlined in ftp://ds.internic.net/internet-drafts/draft-ietf-http-state-mgmt-*.txt. Please note that the implementation of these functions is evolving; be cautious when using them.

## InternetGetCookie

```
BOOL InternetGetCookie(
    IN  LPCSTR lpszUrlName,
    IN  LPCSTR lpszCookieName,
    OUT LPSTR lpszCookieData,
    IN OUT LPDWORD  lpdwSize
);
```

Returns cookies for the specified URL and all its parent URLs.

- Returns TRUE if successful, or FALSE otherwise. To get the specific error value, call GetLastError. The following error values apply to InternetGetCookie:

| Value | Description |
|---|---|
| ERROR_NO_MORE_ITEMS | There is no cookie for the specified URL and all its parents. |
| ERROR_INSUFFICIENT_BUFFER | The value passed in *lpdwSize* is insufficient to copy all the cookie data. The value returned in *lpdwSize* is the size of the buffer necessary to get all the data. |

*lpszUrlName*
Address of a string that contains the URL to get cookies for.

*lpszCookieName*
Address of the name of the cookie to get for the specified URL. This has not been implemented in this release.

*lpszCookieData*
Address of the buffer that receives the cookie data. This value can be NULL.

*lpdwSize*
Address of a variable that specifies the size of the *lpszCookieData* buffer. If the function succeeds, the buffer receives the amount of data copied to the *lpszCookieData* buffer. If *lpCookieData* is NULL, this parameter receives a value that specifies the size of the buffer necessary to copy all the cookie data.

No call to InternetOpen is required to use this function. InternetGetCookie checks in the windows\cookies directory for cookies and searches memory for any cookies that do not have an expiration date, since these cookies are not written to any files. Rules for creating cookie files are internal to Win32 Internet functions and may change in the future.

## InternetSetCookie

```
BOOL InternetSetCookie(
    IN LPCSTR lpszUrlName,
    IN LPCSTR lpszCookieName,
    IN LPCSTR lpszCookieData
);
```

Sets a cookie on the specified URL.

- Returns TRUE if successful, or FALSE otherwise. To get the specific error code, call GetLastError.

*lpszUrlname*
Address of a null-terminated string that specifies the URL for which the cookie should be set.

*lpszCookieName*
Address of a string that contains the name to associate with the cookie. If this parameter is NULL, no name is associated with the cookie. This parameter is not implemented in this release and should be set to NULL.

*lpszCookieData*
Address of the actual data to associate with the URL.

Creating a new cookie may cause a dialog box to appear on the screen if the appropriate registry value is set. There is no way to change the registry value from a Win32 Internet function.

# ersistent URL Cache Functions

This section describes the functions used by clients that need persistent caching services. The functions allow an application to save data in the local file system for subsequent use, such as in situations where access to the data is over a low bandwidth link, or the access is not available at all. The calling program that inserts data into the persistent cache assigns a source name that is used to do operations such as retrieve the data, set and get some properties on it, and delete it.

The protocols implemented in Win32 Internet functions use the cache functions to provide persistent caching and off-line browsing. Unless explicitly specified not to cache through the INTERNET_FLAG_NO_CACHE_WRITE flag, Win32 Internet functions cache all data downloaded from the network. The responses to POST data are not cached.

## ndling Structures with Variable Size formation

The cache may contain variable size information for each URL stored. This is reflected in the INTERNET_CACHE_ENTRY_INFO structure. When the cache functions return this structure, they create a buffer that is always the size of INTERNET_CACHE_ENTRY_INFO plus any variable-size information. If a pointer member is not NULL, it points to the memory area immediately after the structure. While copying the returned buffer from a function into another buffer, the pointer members should be fixed to point to the appropriate place in the new buffer, as the following example shows:

```
lpDstCEInfo->lpszSourceUrlName =
    (LPINTERNET_CACHE_ENTRY_INFO) ((LPBYTE) lpSrcCEInfo +
((DWORD) (lpOldCEInfo->lpszSourceUrlName) - (DWORD) lpOldCEInfo))
```

Some cache functions fail with the ERROR_INSUFFICIENT_BUFFER error value if you specify a buffer that is too small to contain the cache-entry information retrieved by the function. In this case, the function also returns the required size of the buffer. You can then allocate a buffer of the appropriate size and call the function again. If you want the function to succeed on the first call, allocate a buffer of the size specified by the MAX_CACHE_ENTRY_INFO_SIZE value, and then set the

**dwStructSize** member of the **INTERNET_CACHE_ENTRY_INFO** structure to MAX_CACHE_ENTRY_INFO_SIZE when calling the function.

# Cache APIs

## CommitUrlCacheEntry

```
BOOL CommitUrlCacheEntry(
    IN LPCSTR lpszUrlName,
    IN LPCTSTR lpszLocalFileName,
    IN FILETIME ExpireTime,
    IN FILETIME LastModifiedTime,
    IN DWORD CacheEntryType,
    IN LPCBYTE lpHeaderInfo,
    IN DWORD dwHeaderSize,
    IN LPCTSTR lpszFileExtension,
    IN DWORD dwReserved
    );
```

Caches data in the specified file in the cache storage and associates it with the given URL.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call **GetLastError**. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_FILE_NOT_FOUND | The specified local file is not found. |
| ERROR_DISK_FULL | The cache storage is full. |

*lpszUrlName*
Address of a string that contains the source name of the cache entry. The name string must be unique, and should not contain any escape characters.

*lpszLocalFileName*
Address of a string that contains the name of the local file that is being cached. This should be the same name as that returned by **CreateUrlCacheEntry**.

*ExpireTime*
A **FILETIME** structure that contains the expire date and time (GMT) of the file that is being cached. If the expire date and time is unknown, set this parameter to zero.

*LastModifiedTime*
A **FILETIME** structure that contains the last modified date and time (GMT) of the URL that is being cached. If the last modified date and time is unknown, set this parameter to zero.

*CacheEntryType*
Type of the cache entry.

*lpHeaderInfo*
Address of the header information. If this parameter is not NULL, the header information is treated as extended attributes of the URL and is returned back in the INTERNET_CACHE_ENTRY_INFO structure.

*dwHeaderSize*
Size of the header information. If *lpHeaderInfo* is not NULL, this value is assumed to indicate the size of the header information. An application can maintain headers as part of the data and provide *dwHeaderSize*, together with a NULL value for *lpHeaderInfo*.

*lpszFileExtension*
Address of a buffer that contains information maintained in the cache database for future use. In this version of Win32 Internet functions, this information is not used.

*dwReserved*
Reserved; must be zero.

If the cache storage is full, the function invokes cache cleanup to make space for this new file. If the file size is bigger than the cache size, the function returns ERROR_DISK_FULL. If the cache entry already exists, the function overwrites the entry. The user could specify SPARSE_CACHE_ENTRY in the Commit to indicate that the size of the data is incomplete.

## CreateUrlCacheEntry

```
BOOL CreateUrlCacheEntry(
    IN LPCSTR lpszUrlName,
    IN DWORD dwExpectedFileSize,
    IN DWORD lpszFileExtension,
    OUT LPTSTR lpszFileName,
    IN DWORD dwReserved
);
```

Allocates requested cache storage, and creates a local file name for saving the cache entry corresponding to the source name.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError.

*lpszUrlName*
Address of a string that contains the name of the URL. The string should not contain any escape characters.

*dwExpectedFileSize*
Expected size of the file needed to store the data corresponding to the source entity. If the expected size is unknown, set this value to zero.

*lpszFileExtension*
Address of a string that contains an extension name of the file in the local storage.

*lpszFileName*
Address of a buffer that receives the file name. The buffer should be large enough (MAX_PATH) to store the file path name of the created file.

*dwReserved*
Reserved; must be zero.

Internet services that use the cache should call this function to write directly into the cache storage. The caller should indicate the expected size of the file, but it is not guaranteed. Once the file is completely received, the caller should call CommitUrlCacheEntry to commit the entry in the cache.

## GetUrlCacheEntryInfo

```
BOOL GetUrlCacheEntryInfo(
    IN LPCSTR lpszUrlName,
    IN LPINTERNET_CACHE_ENTRY_INFO lpCacheEntryInfo,
    IN OUT LPDWORD lpdwCacheEntryInfoBufferSize
);
```

Retrieves information about a cache entry.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_INSUFFICIENT_BUFFER | The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information. |
| ERROR_FILE_NOT_FOUND | The specified cache entry is not found in the cache. |

*lpszUrlName*
Address of a string that contains the name of the cache entry. The name string should not contain any escape characters.

*lpCacheEntryInfo*
Address of an INTERNET_CACHE_ENTRY_INFO structure that receives information about the cache entry.

*lpdwCacheEntryInfoBufferSize*
Address of a variable that specifies the size of the *lpszCacheEntryInfo* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

## ReadUrlCacheEntryStream

```
BOOL ReadUrlCacheEntryStream(
    IN HANDLE hUrlCacheStream,
    IN  DWORD dwLocation,
```

```
    IN OUT LPVOID lpBuffer,
    IN OUT LPDWORD lpdwLen,
    IN DWORD dwReserved
    );
```

Reads the cached data from a stream that has been opened using the RetrieveUrlCacheEntryStream function.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError.

*hUrlCacheStream*
Handle that was returned by the RetrieveUrlCacheEntryStream function.

*dwLocation*
Offset to read from.

*lpBuffer*
Address of a buffer that receives the data.

*lpdwLen*
Address of a variable that specifies the length of the *lpBuffer* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

*dwReserved*
Reserved; must be zero.

## RetrieveUrlCacheEntryFile

```
BOOL RetrieveUrlCacheEntryFile(
    IN LPCSTR lpszUrlName,
    OUT LPINTERNET_CACHE_ENTRY_INFO lpCacheEntryInfo,
    IN OUT LPDWORD lpdwCacheEntryInfoBufferSize
    IN DWORD dwReserved
    );
```

Retrieves a cache entry from the cache in the form of a file.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_FILE_NOT_FOUND | The cache entry specified by the source name is not found in the cache storage. |
| ERROR_INSUFFICIENT_BUFFER | The size of the *lpCacheEntryInfo* buffer as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to get all the information. |

*lpszUrlName*
Source name of the cache entry. This must be a unique name. The name string should not contain any escape characters.

lpCacheEntryInfo
Address of a cache entry information buffer. If the buffer is not sufficient to accommodate all the information associated with the URL, one or more of the embedded pointers will be NULL.

lpdwCacheEntryInfoBufferSize
Address of a variable that specifies the size of the *lpCacheEntryInfo* buffer. When the function returns, this variable contains the size of the actual buffer used or required. The caller should check the return value in this variable. If the return size is less than or equal to the size passed in, all the relevant data has been returned.

*dwReserved*
Reserved; must be zero.

If an extension was provided while calling CommitUrlCacheEntry, the file will have the specified extension. If the entry is not available in the cache, this function returns ERROR_FILE_NOT_FOUND; otherwise, it returns the name of the local file. The caller is given only read permission to the local file, so the caller should not attempt to write or delete the file.

The file is locked for the caller when it is retrieved; the caller should unlock the file after it has been used up. The cache manager will automatically unlock the locked files after a certain interval. While the file is locked, the cache manager will not remove the file from the cache. It is important to note that this function may be efficient or expensive, depending on the internal implementation of the cache. For instance, if the URL data is stored in a packed file that contains data for other URLs, the cache will make a copy of the data to a file in a temporary directory maintained by the cache. The cache will eventually delete the copy. It is recommended that this function be used only in situations where a file name is needed to launch an application. RetrieveUrlCacheEntryStream and associated stream functions should be used in most cases.

## RetrieveUrlCacheEntryStream

```
HANDLE RetrieveUrlCacheEntryStream(
    IN LPCSTR lpszUrlName,
    OUT LPINTERNET_CACHE_ENTRY_INFO lpCacheEntryInfo,
    IN OUT LPDWORD lpdwCacheEntryInfoBufferSize,
    IN BOOL fRandomRead,
    IN DWORD dwReserved
    );
```

Provides the most efficient and implementation-independent way of accessing the cache data.

- Returns a valid handle for use in the ReadUrlCacheEntryStream and UnlockUrlCacheEntryStream functions if successful, or

INVALID_HANDLE_VALUE otherwise. To get extended error information, call GetLastError. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_FILE_NOT_FOUND | The cache entry specified by the source name is not found in the cache storage. |
| ERROR_INSUFFICIENT_BUFFER | The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information. |

*lpszUrlName*
Address of a string that contains the source name of the cache entry. This must be a unique name. The name string should not contain any escape characters.

*lpCacheEntryInfo*
Address of an INTERNET_CACHE_ENTRY_INFO structure that receives information about the cache entry.

*lpdwCacheEntryInfoBufferSize*
Address of a variable that specifies the size of the *lpCacheEntryInfo* buffer. When the function returns, the variable receives the number of bytes copied to the buffer, or the required size of the buffer.

*fRandomRead*
Flag to indicate whether the stream is opened for random access or not. Set the flag to TRUE to open the stream for random access.

*dwReserved*
Reserved; must be zero.

Cache clients that do not need URL data in the form of a file should use this function to access the data for a particular URL.

## SetUrlCacheEntryInfo

```
BOOL SetUrlCacheEntryInfo(
    IN LPCSTR lpszUrlName,
    IN LPINTERNET_CACHE_ENTRY_INFO lpCacheEntryInfo,
    IN DWORD dwFieldControl
);
```

Sets the specified members of the INTERNET_CACHE_ENTRY_INFO structure.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_FILE_NOT_FOUND | The specified cache entry is not |

| | |
|---|---|
| | found in the cache. |
| ERROR_INVALID_PARAMETER | The value(s) to be set is invalid. |

*lpszUrlName*
Address of a string that contains the name of the cache entry. The name string should not contain any escape characters.

*lpCacheEntryInfo*
Address of an INTERNET_CACHE_ENTRY_INFO structure.

*dwFieldControl*
Bitmask that indicates the members that are to be set. Can be a combination of the following values:

CACHE_ENTRY_ATTRIBUTE_FC
CACHE_ENTRY_HITRATE_FC
CACHE_ENTRY_MODTIME_FC
CACHE_ENTRY_EXPTIME_FC
CACHE_ENTRY_ACCTIME_FC
CACHE_ENTRY_SYNCTIME_FC
CACHE_ENTRY_HEADERINFO_FC

The last two flags have not been implemented in this release.

## UnlockUrlCacheEntryFile

```
BOOL UnlockUrlCacheEntryFile(
    IN LPCSTR lpszUrlName,
    IN DWORD dwReserved
    );
```

Unlocks the cache entry that was locked while the file was retrieved for use from the cache.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. ERROR_FILE_NOT_FOUND indicates that the cache entry specified by the source name is not found in the cache storage.

*lpszUrlName*
Address of a string that contains the source name of the cache entry that is being unlocked. The name string should not contain any escape characters.

*dwReserved*
Reserved; must be zero.

The application should not access the file after calling this function.

When this function returns, the cache manager is free to delete the cache entry.

## UnlockUrlCacheEntryStream

```
BOOL UnlockUrlCacheEntryStream(
    IN HANDLE hUrlCacheStream,
    IN DWORD dwReserved
);
```

Closes the stream that has been retrieved using the RetrieveUrlCacheEntryStream function.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError.

*hUrlCacheStream*
Handle that was returned by the RetrieveUrlCacheEntryStream function.

*dwReserved*
Reserved; must be zero.

See also RetrieveUrlCacheEntryStream

# che Enumeration

## DeleteUrlCacheEntry

```
BOOL DeleteUrlCacheEntry(
    IN LPCSTR lpszUrlName
);
```

Removes the file associated with the source name from the cache, if the file exists.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_FILE_NOT_FOUND | The file is not in the cache. |
| ERROR_ACCESS_DENIED | The file is in use. |

*lpszUrlName*
Address of a string that contains the name of the source corresponding to the cache entry.

## FindCloseUrlCache

```
BOOL FindCloseUrlCache(
    IN HANDLE hEnumHandle
    );
```

Closes the specified enumeration handle.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError.

*hEnumHandle*
Handle returned by a previous call to the FindFirstUrlCacheEntry function.

See also FindFirstUrlCacheEntry

## FindFirstUrlCacheEntry

```
HANDLE FindFirstUrlCacheEntry (
    IN LPCSTR lpszUrlSearchPattern,
    OUT LPINTERNET_CACHE_ENTRY_INFO lpFirstCacheEntryInfo,
    IN OUT LPDWORD lpdwFirstCacheEntryInfoBufferSize
);
```

Begins the enumeration of the cache.

- Returns a handle that the application can use in the FindNextUrlCacheEntry function to retrieve subsequent entries in the cache. If the function fails, the return value is NULL. To get extended error information, call GetLastError. ERROR_INSUFFICIENT_BUFFER indicates that the size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information.

*lpszUrlSearchPattern*
Address of a string that contains the source name pattern to search for. If this parameter is NULL, the function uses *.*. (In this version, only *.* semantics are implemented.)

*lpFirstCacheEntryInfo*
Address of an INTERNET_CACHE_ENTRY_INFO structure.

*lpdwFirstCacheEntryInfoBufferSize*
Address of a variable that specifies the size of the *lpFirstCacheEntryInfo* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

This function and the FindNextUrlCacheEntry function return variable size information. In order to not have the enumeration terminate due to ERROR_INSUFFICIENT_BUFFER, an application should create one buffer of the size specified by the MAX_CACHE_ENTRY_INFO_SIZE value, and pass the address of the buffer repeatedly to all the enumeration functions. After the function succeeds, another buffer may be used of the size returned by *lpdwCacheEntryInfoBufferSize* to keep the returned information. Be careful to fix the pointer elements while copying the buffer.

See also FindNextUrlCacheEntry

## FindNextUrlCacheEntry

```
BOOL FindNextUrlCacheEntry(
    IN HANDLE hEnumHandle,
    OUT LPINTERNET_CACHE_ENTRY_INFO lpNextCacheEntryInfo,
    IN OUT LPWORD lpdwNextCacheEntryInfoBufferSize
);
```

Retrieves the next entry in the cache.

- Returns TRUE if successful, or FALSE otherwise. To get extended error information, call GetLastError. Possible error values include:

| Value | Meaning |
|---|---|
| ERROR_NO_MORE_FILES | The enumeration completed. |
| ERROR_INSUFFICIENT_BUFFER | The size of *lpCacheEntryInfo* as specified by *lpdwCacheEntryInfoBufferSize* is not sufficient to contain all the information. The value returned in *lpdwCacheEntryInfoBufferSize* indicates the buffer size necessary to contain all the information. |

*hEnumHandle*
Enumeration handle obtained from a previous call to FindFirstUrlCacheEntry.

*lpNextCacheEntryInfo*
Address of an INTERNET_CACHE_ENTRY_INFO structure that receives information about cache entry.

*lpdwNextCacheEntryInfoBufferSize*
Address of a variable that specifies the size of the *lpNextCacheEntryInfo* buffer. When the function returns, the variable contains the number of bytes copied to the buffer, or the required size of the buffer.

See also FindFirstUrlCacheEntry

# :ructures

This list identifies the Win32 Internet function data structures and their uses.

## GOPHER_ATTRIBUTE_TYPE

```
typedef struct {
    DWORD CategoryId
        DWORD AttributeId
    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_ASK_ATTRIBUTE_TYPE Ask;
        GOPHER_UNKNOWN_ATTRIBUTE Unknown;
    } AttributeType;
} GOPHER_ATTRIBUTE_TYPE, *LPGOPHER_ATTRIBUTE_TYPE;
```

Contains the relevant information of a single Gopher attribute for an object.

**CategoryId**
Gopher name for the attribute. The possible values include:

GOPHER_CATEGORY_ID_ALL
GOPHER_CATEGORY_ID_INFO
GOPHER_CATEGORY_ID_ADMIN
GOPHER_CATEGORY_ID_VIEWS
GOPHER_CATEGORY_ID_ABSTRACT
GOPHER_CATEGORY_ID_VERONICA
GOPHER_CATEGORY_ID_UNKNOWN

**AttributeId**
Identifier of the structure contained in the **AttributeType** member. The possible values include:

GOPHER_ATTRIBUTE_ID_ADMIN
GOPHER_ATTRIBUTE_ID_MOD_DATE
GOPHER_ATTRIBUTE_ID_TTL
GOPHER_ATTRIBUTE_ID_SCORE
GOPHER_ATTRIBUTE_ID_RANGE
GOPHER_ATTRIBUTE_ID_SITE
GOPHER_ATTRIBUTE_ID_ORG
GOPHER_ATTRIBUTE_ID_LOCATION
GOPHER_ATTRIBUTE_ID_GEOG
GOPHER_ATTRIBUTE_ID_TIMEZONE
GOPHER_ATTRIBUTE_ID_PROVIDER
GOPHER_ATTRIBUTE_ID_VERSION
GOPHER_ATTRIBUTE_ID_ABSTRACT
GOPHER_ATTRIBUTE_ID_VIEW
GOPHER_ATTRIBUTE_ID_TREEWALK

GOPHER_ATTRIBUTE_ID_UNKNOWN

**AttributeType**
Actual setting for the Gopher attribute. The specific value of **AttributeType** depends on the **AttributeId** member. The definitions of the various attribute structures is available in Wininet.h.

See also **GopherGetAttribute**

## GOPHER_FIND_DATA

```
typedef struct {
    TCHAR DisplayString[MAX_GOPHER_DISPLAY_TEXT + 1];
    DWORD GopherType;
    DWORD SizeLow;
    DWORD SizeHigh;
    FILETIME LastModificationTime;
    TCHAR Locator[MAX_GOPHER_LOCATOR_LENGTH + 1];
} GOPHER_FIND_DATA, FAR *LPGOPHER_FIND_DATA;
```

Contains information retrieved by the **GopherFindFirstFile** and **InternetFindNextFile** functions.

**DisplayString**
String that contains the friendly name of an object. An application can display this string to allow the user to select the object.

**GopherType**
Mask of flags that describe the item returned.

**FileSizeLow**
Low 32 bits of the file size.

**FileSizeHigh**
High 32 bits of the file size.

**LastModificationTime**
Time when the file was last modified.

**Locator**
String that identifies the file. An application can pass the locator string to **GopherOpenFile** or **GopherFindFirstFile**.

See also **GopherFindFirstFile**

## INTERNET_ASYNC_RESULT

```
typedef struct {
    DWORD dwResult;
    DWORD dwError;
} INTERNET_ASYNC_RESULT, * LPINTERNET_ASYNC_RESULT;
```

Contains the result of a call to an asynchronous function. This structure is used with the **InternetStatusCallback** function.

**dwResult**
HINTERNET, DWORD, or BOOL return code from an asynchronous function.

**dwError**
Error code if dwResult indicates that the function failed. If the operation succeeded, this member usually contains ERROR_SUCCESS.

See also **InternetStatusCallback**

## INTERNET_CACHE_ENTRY_INFO

```
typedef struct _CACHE_ENTRY_INFO {
    DWORD dwStructSize;
    LPSTR lpszSourceUrlName;
    LPTSTR lpszLocalFileName;
    DWORD CacheEntryType;
    DWORD dwUseCount;
    DWORD dwHitRate;
    DWORD dwSizeLow;
    DWORD dwSizeHigh;
    FILETIME LastModifiedTime;
    FILETIME ExpireTime;
    FILETIME LastAccessTime;
    FILETIME LastSyncTime;
    LPBYTE lpHeaderInfo;
    DWORD dwHeaderInfoSize;
    LPTSTR lpszFileExtension;
    DWORD dwReserved;
} INTERNET_CACHE_ENTRY_INFO, *LPINTERNET_CACHE_ENTRY_INFO;
```

Contains information about an entry in the cache.

**dwStructSize**
Size, in bytes, of this structure.

**lpszSourceUrlName**
Address of a string that contains the URL name. The string occupies memory area at the end of this structure.

**lpszLocalFileName**
Address of a string that contains the local file name. The string occupies memory area at the end of this structure.

**CacheEntryType**
Cache type bit mask. Can be one of these values:

| Value | Meaning |
|---|---|
| NORMAL_CACHE_ENTRY | Normal cache entry; may be deleted to recover space for new entries. |
| STABLE_CACHE_ENTRY | Stable cache entry such as graphic and audio/video files; may be deleted to recover space for the new entries only when there is no more NORMAL_CACHE_ENTRY. |

| | |
|---|---|
| STICKY_CACHE_ENTRY | Entries that will never be removed automatically by the cache management system. |
| SPARSE_CACHE_ENTRY | This cache entry is incomplete. |
| OCX_CACHE_ENTRY | Special OCX type cache entry. |

**dwUseCount**
Current user count of the cache entry.

**dwHitRate**
Number of times the cache entry was retrieved.

**dwSizeLow**
Low-order double word of the file size.

**dwSizeHigh**
High-order double word of the file size.

**LastModifiedTime**
Last modified time of this URL in GMT format.

**ExpireTime**
Expiration time of this file in GMT format.

**LastAccessTime**
Last accessed time in GMT format.

**LastSyncTime**
Last time the cache was synchronized.

**lpHeaderInfo**
Address of a buffer that contains the header information. The buffer occupies memory at the end of this structure.

**dwHeaderInfoSize**
Size of the lpHeaderInfo buffer.

**lpszFileExtension**
Address of a string that contains the file extension used to retrieve the data as a file. The string occupies memory area at the end of this structure.

**dwReserved**
Reserved; must be zero.

The MAX_CACHE_ENTRY_INFO_SIZE value defines the maximum size of the INTERNET_CACHE_ENTRY_INFO structure that could be returned by the implementation of the cache functions. Thus passing in a buffer of this size in functions returning INTERNET_CACHE_ENTRY_INFO guarantees that the function does not fail because of an insufficient buffer.

## INTERNET_CERTIFICATE_INFO

```
typedef struct {
    FILETIME ftExpiry;
    FILETIME ftStart;
    LPTSTR lpszSubjectInfo;
        LPTSTR lpszIssuerInfo;
    LPTSTR lpszProtocolName;
    LPTSTR lpszSignatureAlgName;
```

```
    LPTSTR lpszEncryptionAlgName;
    DWORD dwKeySize;
} INTERNET_CERTIFICATE_INFO, * LPINTERNET_CERTIFICATE_INFO;
```

Contains certificate information returned from the server. This structure is used by the **InternetQueryOption** function.

**ftExpiry**
FILETIME structure that contains the date the certificate expires.

**ftStart**
FILETIME structure that contains the date the certificate becomes valid.

**lpszSubjectInfo**
Address of a buffer that contains the name of the organization, site, and server for which the certificate was issued.

**lpszIssuerInfo**
Address of a buffer that contains the name of the organization, site, and server that issued the certificate.

**lpszProtocolName**
Address of a buffer that contains the name of the protocol used to provide the secure connection.

**lpszSignatureAlgName**
Address of a buffer that contains the name of the algorithm used for signing the certificate.

**lpszEncryptionAlgName**
Address of a buffer that contains the name of the algorithm used for doing encryption over the secure channel (SSL/PCT) connection.

**dwKeySize**
Size, in bytes, of the key.

Applications requesting this information must free pointers that are allocated and placed in the returned structure.

See also **InternetQueryOption**

## INTERNET_PREFETCH_STATUS

```
typedef struct {
    DWORD dwStatus;
    DWORD dwSize;
} INTERNET_PREFETCH_STATUS, * LPINTERNET_PREFETCH_STATUS;
```

Contains the status of a prefetch download operation.

**dwStatus**
Status of the download. Can be one of these values:

| | |
|---|---|
| INTERNET_PREFETCH_PROGRESS | The operation is in progress. |
| INTERNET_PREFETCH_COMPLETE | The operation has completed. |
| INTERNET_PREFETCH_ABORTED | The operation was aborted. |

**dwSize**
Size, in bytes, of data downloaded so far.

## INTERNET_PROXY_INFO

```
typedef struct {
    DWORD dwAccessType;
    LPCTSTR lpszProxy;
    LPCTSTR lpszProxyBypass;
} INTERNET_PROXY_INFO, * LPINTERNET_PROXY_INFO;
```

Contains information that is supplied with the INTERNET_OPTION_PROXY value to get or set proxy information on a handle obtained from a call to the InternetOpen function.

**dwAccessType**
Access type. Can be one of these values:

| Value | Meaning |
|---|---|
| INTERNET_OPEN_TYPE_DIRECT | Internet accessed through a direct connection. |
| INTERNET_OPEN_TYPE_PROXY | Internet accessed using a proxy. |
| INTERNET_OPEN_TYPE_PRECONFIG | Applies only when setting proxy information. |

**lpszProxy**
Proxy server list.

**lpszProxyBypass**
Proxy bypass list.

## INTERNET_SCHEME

```
typedef enum {
    INTERNET_SCHEME_PARTIAL = -2,
    INTERNET_SCHEME_UNKNOWN = -1,
    INTERNET_SCHEME_DEFAULT = 0,
    INTERNET_SCHEME_FTP,
    INTERNET_SCHEME_GOPHER,
    INTERNET_SCHEME_HTTP,
    INTERNET_SCHEME_HTTPS,
    INTERNET_SCHEME_FILE,
    INTERNET_SCHEME_NEWS,
    INTERNET_SCHEME_MAILTO,
    INTERNET_SCHEME_FIRST = INTERNET_SCHEME_FTP,
    INTERNET_SCHEME_LAST = INTERNET_SCHEME_MAILTO
} INTERNET_SCHEME, * LPINTERNET_SCHEME;
```

Defines the flags used with the nScheme member of the URL_COMPONENTS structure.

## INTERNET_VERSION_INFO

```
typedef struct {
    DWORD dwMajorVersion;
    DWORD dwMinorVersion;
} INTERNET_VERSION_INFO, * LPINTERNET_VERSION_INFO;
```

Contains the version number of the DLL that contains the Windows Internet functions (Wininet.dll). This structure is used when passing the INTERNET_OPTION_VERSION flag to the InternetQueryOption function.

**dwMajorVersion**
Major version number.

**dwMinorVersion**
Minor version number.

## URL_COMPONENTS

```
typedef struct {
    DWORD dwStructSize;
    LPSTR lpszScheme;
    DWORD dwSchemeLength;
    INTERNET_SCHEME nScheme;
    LPSTR lpszHostName;
    DWORD dwHostNameLength;
    INTERNET_PORT nPort;
    LPSTR lpszUserName;
    DWORD dwUserNameLength;
    LPSTR lpszPassword;
    DWORD dwPasswordLength'
    LPSTR lpszUrlPath;
    DWORD dwUrlPathLength;
    LPTSTR lpszExtraInfo;
    DWORD dwExtraInfoLength;
} URL_COMPONENTS;
```

Contains the constituent parts of a URL. This structure is used with the InternetCrackUrl and InternetCreateUrl functions.

**dwStructSize**
Size, in bytes, of this structure. Used for version checking.

**lpszScheme**
Address of a buffer that contains the scheme name.

**dwSchemeLength**
Length of the scheme name.

**nScheme**
Enumerated scheme type (if known). For a list of scheme types, see INTERNET_SCHEME.

**lpszHostName**
Address of a buffer that contains the host name.

**dwHostNameLength**
Length of the host name.

**nPort**
Converted port number.

**lpszUserName**
Address of a buffer that contains the user name.

**dwUserNameLength**
Length of the user name.

**lpszPassword**
Address of a buffer that contains the password.

**dwPasswordLength**
Length of the password.

**lpszUrlPath**
Address of a buffer that contains the URL path.

**dwUrlPathLength**
Length of the URL path.

**lpszExtraInfo**
Address of a buffer that contains the extra information (for example, ?foo or #foo).

**dwExtraInfoLength**
Length of the extra information.

For **InternetCrackUrl**, if a pointer member and its corresponding length member are both zero, that component is not returned. If the pointer member is NULL but the length member is not zero, both the pointer and length members are returned. If both pointer and corresponding length members are non-zero, the pointer member points to a buffer where the component is copied. The component may be un-escaped, depending on the *dwFlags* parameter of **InternetCrackUrl**.

For **InternetCreateUrl**, the pointer members should be NULL if the component is not required. If the corresponding length member is zero, the pointer member is the address of a zero-terminated string. If the length member is not zero, it is the string length of the corresponding pointer member

# ror Codes

The HTTP functions control the transmission and content of HTTP requests.

The Win32 Internet functions return Win32 error codes where appropriate. The following error codes are specific to the Win32 Internet functions:

**ERROR_INTERNET_CLIENT_AUTH_NOT_SETUP**
Client authorization is not set up on this computer.

**ERROR_INTERNET_OUT_OF_HANDLES**
No more handles could be generated at this time.

**ERROR_INTERNET_TIMEOUT**
The request has timed out.

**ERROR_INTERNET_EXTENDED_ERROR**

An extended error was returned from the server. This is typically a string or buffer containing a verbose error message. Call **InternetGetLastResponseInfo** to retrieve the error text.

**ERROR_INTERNET_INTERNAL_ERROR**

An internal error has occurred.

**ERROR_INTERNET_INVALID_URL**

The URL is invalid.

**ERROR_INTERNET_UNRECOGNIZED_SCHEME**

The URL scheme could not be recognized, or is not supported.

**ERROR_INTERNET_NAME_NOT_RESOLVED**

The server name could not be resolved.

**ERROR_INTERNET_PROTOCOL_NOT_FOUND**

The requested protocol could not be located.

**ERROR_INTERNET_INVALID_OPTION**

A request to **InternetQueryOption** or **InternetSetOption** specified an invalid option value.

**ERROR_INTERNET_BAD_OPTION_LENGTH**

The length of an option supplied to **InternetQueryOption** or **InternetSetOption** is incorrect for the type of option specified.

**ERROR_INTERNET_OPTION_NOT_SETTABLE**

The request option cannot be set, only queried.

**ERROR_INTERNET_SHUTDOWN**

The Win32 Internet function support is being shut down or unloaded.

**ERROR_INTERNET_INCORRECT_USER_NAME**

The request to connect and log on to an FTP server could not be completed because the supplied user name is incorrect.

**ERROR_INTERNET_INCORRECT_PASSWORD**

The request to connect and log on to an FTP server could not be completed because the supplied password is incorrect.

**ERROR_INTERNET_LOGIN_FAILURE**

The request to connect and log on to an FTP server failed.

**ERROR_INTERNET_INVALID_OPERATION**

The requested operation is invalid.

**ERROR_INTERNET_OPERATION_CANCELLED**

The operation was canceled, usually because the handle on which the request was operating was closed before the operation completed.

**ERROR_INTERNET_INCORRECT_HANDLE_TYPE**

The type of handle supplied is incorrect for this operation.

ERROR_INTERNET_INCORRECT_HANDLE_STATE

The requested operation cannot be carried out because the handle supplied is not in the correct state.

ERROR_INTERNET_NOT_PROXY_REQUEST

The request cannot be made via a proxy.

ERROR_INTERNET_REGISTRY_VALUE_NOT_FOUND

A required registry value could not be located.

ERROR_INTERNET_BAD_REGISTRY_PARAMETER

A required registry value was located but is an incorrect type or has an invalid value.

ERROR_INTERNET_NO_DIRECT_ACCESS

Direct network access cannot be made at this time.

ERROR_INTERNET_NO_CONTEXT

An asynchronous request could not be made because a zero context value was supplied.

ERROR_INTERNET_NO_CALLBACK

An asynchronous request could not be made because a callback function has not been set.

ERROR_INTERNET_REQUEST_PENDING

The required operation could not be completed because one or more requests are pending.

ERROR_INTERNET_INCORRECT_FORMAT

The format of the request is invalid.

ERROR_INTERNET_ITEM_NOT_FOUND

The requested item could not be located.

ERROR_INTERNET_CANNOT_CONNECT

The attempt to connect to the server failed.

ERROR_INTERNET_CONNECTION_ABORTED

The connection with the server has been terminated.

ERROR_INTERNET_CONNECTION_RESET

The connection with the server has been reset.

ERROR_INTERNET_FORCE_RETRY

The Win32 Internet function needs to redo the request.

ERROR_INTERNET_ZONE_CROSSING

Not used in this release.

ERROR_INTERNET_MIXED_SECURITY

The content is not entirely secure. Some of the content being viewed may have come from unsecured servers.

ERROR_INTERNET_SSL_CERT_CN_INVALID

The certificate returned by an SSL/PCT server is invalid because of a mismatched server name. The server name that was given by the caller does not match the common name inside the certificate.

ERROR_INTERNET_HANDLE_EXISTS

The request failed because the handle already exists.

**ERROR_FTP_TRANSFER_IN_PROGRESS**

The requested operation cannot be made on the FTP session handle because an operation is already in progress.

**ERROR_FTP_DROPPED**

The FTP operation was not completed because the session was aborted.

**ERROR_GOPHER_PROTOCOL_ERROR**

An error was detected while parsing data returned from the gopher server.

**ERROR_GOPHER_NOT_FILE**

The request must be made for a file locator.

**ERROR_GOPHER_DATA_ERROR**

An error was detected while receiving data from the gopher server.

**ERROR_GOPHER_END_OF_DATA**

The end of the data has been reached.

**ERROR_GOPHER_INVALID_LOCATOR**

The supplied locator is not valid.

**ERROR_GOPHER_INCORRECT_LOCATOR_TYPE**

The type of the locator is not correct for this operation.

**ERROR_GOPHER_NOT_GOPHER_PLUS**

The requested operation can only be made against a Gopher+ server, or with a locator that specifies a Gopher+ operation.

**ERROR_GOPHER_ATTRIBUTE_NOT_FOUND**

The requested attribute could not be located.

**ERROR_GOPHER_UNKNOWN_LOCATOR**

The locator type is unknown.

**ERROR_HTTP_HEADER_NOT_FOUND**

The requested header could not be located.

**ERROR_HTTP_DOWNLEVEL_SERVER**

The server did not return any headers.

**ERROR_HTTP_INVALID_SERVER_RESPONSE**

The server response could not be parsed.

**ERROR_HTTP_INVALID_HEADER**

The supplied header is invalid.

**ERROR_HTTP_INVALID_QUERY_REQUEST**

The request made to **HttpQueryInfo** is invalid.

**ERROR_HTTP_HEADER_ALREADY_EXISTS**

The header could not be added because it already exists.

**ERROR_INVALID_HANDLE**

The handle that was passed to the API has been either invalidated or closed.