Los rincones del API Win32
WinInet y FTP

Ahora nos toca cambiar de tema para centrarnos en el uso del protocolo FTP desde el API WinInet.

Introducción

En los artículos anteriores, nos hemos preocupado de explicar qué es el protocolo HTTP y cómo funciona internamente. Además, hemos profundizado en cómo utilizar el API WinInet para manejar más fácilmente las peticiones y respuestas de HTTP y algunos trucos prácticos que nos pueden ayudar a la hora de enfrentarnos a una situación real.

En este artículo vamos a hacer lo mismo, pero centrándonos en un protocolo muy distinto: el FTP (*File Transfer Protocol*, Protocolo de Transferencia de Archivos).

Durante las próximas páginas vamos a ver las diferencias más importantes entre HTTP y FTP, el modo en que funciona este protocolo, y cómo utilizarlo a través del API WinInet.

Diferencias entre HTTP y FTP

Para comenzar de un modo más sencillo, y como ya conocemos el protocolo HTTP, vamos a ver las principales diferencias entre estos dos protocolos:

- HTTP está orientado a la transmisión de archivos en formato HTML, aunque puede utilizarse (y se utiliza) para cualquier otro formato, siempre y cuando se pueda representar en caracteres ASCII. FTP se orienta a archivos de información binaria, desde ejecutables, hasta ficheros de texto plano, videos, imágenes, etc., y no sólo a su transmisión, sino también a la manipulación del sistema de archivos en el servidor (modificar la estructura de directorios, crear archivos, eliminarlos, etc.). Desde este punto de vista, FTP es mucho más versátil que HTTP, ya que nos permite trabajar con cualquier tipo de archivo.

- HTTP es un protocolo sin estado y sin conexión lógica, es decir: no es necesario realizar una operación de "entrada en el sistema" antes de empezar a trabajar, y no recuerda operaciones anteriores que hayamos hecho. Sin embargo, FTP es un protocolo con conexión y con estado: es imprescindible realizar una operación de "entrada" (*login*) antes de cualquier otra, y se conserva nuestro estado en el servidor. Esto permite aplicar ciertas restricciones, como permisos especiales a ciertos usuarios, cuota de utilización en disco, etc.

- HTTP utiliza un canal físico (el *socket*) para la petición-respuesta. Sin embargo, FTP utiliza dos canales físicos, uno de control y otro para datos. El canal de control se utiliza del mismo modo que en HTTP: para enviar y recibir las peticiones y respuestas del protocolo. El canal de datos es distinto, ya que por él viaja la información requerida, es decir: el archivo que hemos solicitado. Además, el canal de HTTP se cierra una vez terminada la petición, excepto cuando se usa la cabecera "Keep-Connection". En una conexión FTP, el canal de control debe permanecer activo continuamente, mientras que el de datos se creará para cada uno de los envíos que tenga que realizar. Debido a que el

canal de datos debe estar siempre abierto, los servidores FTP cuentan con un límite máximo de usuarios activos, ya que podrían desbordar la pila TCP (el número de conexiones o *sockets* disponibles para enviar y recibir información). Este esquema de dos canales permite que la conexión FTP sea *full-duplex*, es decir: que se permite el envío y recepción de información simultáneamente (mientras se utiliza el canal de datos para recibir el archivo, se pueden seguir enviando peticiones por el canal de control).

- HTTP cuenta con una extensión para canales seguros: HTTPS, al igual que FTP que cuenta con SFTP (Secure FTP) para cifrar la información que se envía y recibe del servidor. Sin embargo, el API WinInet sí que soporta la extensión HTTPS pero no SFTP. Si quisiéramos utilizar el protocolo SFTP, tendríamos que ayudarnos del API WinSock para la conexión al servidor y envío de información cifrada.

- HTTP es un protocolo anónimo, es decir: el usuario que hace la petición permanece en el anonimato, a excepción de la dirección IP que realiza la petición y el programa que se ha utilizado. Sin embargo, el protocolo FTP requiere de un usuario y contraseña para acceder a los recursos. Normalmente, este usuario y contraseña no son más que la cuenta del sistema donde residen los archivos a acceder. Opcionalmente, se dispone del usuario *anonymous* que funciona como usuario general, utilizando como contraseña opcional el domino desde el que realizamos la conexión, o la dirección de correo electrónico. Lógicamente, el acceso a través del login *anonymous* está mucho más restringido que el de otros usuarios.

- El servidor HTTP envía la información requerida dentro de la misma respuesta. En la respuesta de FTP, sólo se incluyen datos informativos, y no los propios datos que hemos pedido. Esto es debido a las dos conexiones que ya hemos comentado: una de control (para las peticiones y respuestas) y otra de datos (para la transferencia de los datos o el archivo que hemos pedido al servidor).

- HTTP se basa en la recomendación MIME para manejar los tipos de archivos, por lo que se abre un amplio abanico de posibilidades. Sin embargo, FTP sólo puede transferir cuatro tipos de archivo: ASCII, binario, local, EBCDIC. De todas formas, con los tipos ASCII y binario se cubren todos los tipos de archivos posibles.

- Para simplificar su uso, el protocolo HTTP oculta todas las características de la transmisión física. En el protocolo FTP se permite configurar este aspecto, pudiendo utilizar uno de los siguientes modos de transmisión: "de bloque", "comprimido" y "de flujo".

- Por debajo del protocolo HTTP sólo contamos con el protocolo propio de transmisión, es decir: TCP, sin embargo, en FTP se utiliza el protocolo TELNET para el envío de datos por la conexión de control y estos, a su vez, utilizan TCP para el envío físico.

Peticiones de FTP

Al igual que el protocolo HTTP, el FTP se basa en el típico esquema cliente/servidor. El cliente se conecta a una máquina remota (servidor) donde está ejecutándose un software especial (el servidor FTP). Una vez establecida la conexión, se inicia una *conversación* entre las dos máquinas, basada en el esquema petición - respuesta: el cliente hace una petición y el servidor contesta con una respuesta. Además, ciertas respuestas pueden llevar asociadas una transmisión de información, que se transmitirá por un canal de datos creado para tal efecto.

Las peticiones hechas por un cliente FTP no son más que comandos, al más puro estilo MS-DOS o Unix. Una palabra clave define el comando y los parámetros definen sobre qué objetos actúa o cómo debe comportarse.

Por ejemplo, sabemos que para cambiar de directorio actual en MS-DOS o Unix hay que utilizar el comando:

```
cd [nuevo directorio]
```

Así que en FTP se hará de un modo parecido, simplemente cambiando el nombre del comando:

```
CWD [nuevo directorio]
```

Los comandos de FTP son de 3 ó 4 letras en mayúsculas, seguidos de un espacio y los parámetros que corresponda.

En la siguiente tabla tenéis una lista de algunos de los comandos definidos:

Comando	**Descripción**
`USER [nombre de usuario]`	Identifica a un usuario en el servidor FTP
`PASS [contraseña]`	Contraseña del usuario pasado con USER. Debe ir precedido de un comando USER.
`ABOR`	Cancela la operación que esté procesándose, y la transmisión que pueda estar ejecutándose en el canal de datos.
`QUIT`	Cierra la conexión con el servidor.
`PORT`	Especifica el puerto del cliente donde se realizará la conexión de datos.
`CWD [nuevo directorio]`	Cambia el directorio activo.
`CDUP`	Cambia al directorio padre (igual que "cd .." en MS-DOS o Unix).
`MKD [nombre directorio]`	Crea un nuevo directorio.
`RMD [nombre directorio]`	Elimina un directorio.
`DELE [nombre de archivo]`	Borra un archivo en la estructura de directorios del servidor.
`RNFR [nombre de archivo]`	Renombrar un archivo. Este comando indica el nombre de archivo a renombrar y debe estar seguido por el comando RNTO.
`RNTO [nuevo nombre archivo]`	Cambia el nombre del archivo indicado con el comando RNFR y lo establece al nombre pasado en este parámetro.
`SIZE [archivo]`	Retorna el tamaño (en bytes) del archivo remoto.
`LIST [archivo o directorio]`	Muestra una lista de los archivos en el directorio actual (si no se ha pasado ningun parámetro), en el directorio pasado por parámetro (si es un directorio), o bien muestra información del archivo pasado por parámetro (si es un archivo).
`NLST [directorio]`	Muestra una lista de archivos en el directorio pasado, o en el directorio activo si no se pasa ninguno.
`NOOP`	Este es un comando *dummy* que para lo único para lo que sirve es para que el servidor responda con un mensaje de OK. Se utiliza para asegurarnos que la conexión está activa y funcionando.
`APPE [nombre de`	Añade el contenido de un archivo local al final de un fichero remoto.

archivo]	
RETR [nombre de archivo]	Inicia la descarga de un fichero remoto.
STOR [nombre de archivo]	Inicia el envío al servidor de un fichero local.

Una vez que el comando ha sido ejecutado, se retorna un código de error, seguido de un mensaje descriptivo. Además, si la operación requiere de una transferencia de información, se crea el canal de datos para ello.

Códigos de retorno

Los códigos retornados por los comandos tienen tres dígitos, cada uno de los cuales tiene un significado. El significado de estos tres dígitos va de más genérico a más concreto:

1. Primer dígito: hace referencia al éxito o fracaso de la operación. Además indica quién es el causante del posible error, con los siguientes códigos:
    - 1xx: El servidor ha iniciado correctamente la ejecución del comando.
    - 2xx: El servidor ha terminado correctamente la ejecución del comando.
    - 3xx: El servidor ha recibido correctamente el comando, pero se necesitan más datos para comenzar la ejecución.
    - 4xx. El servidor ha recibido el comando pero no es correcto. Se continúa con la ejecución.
    - 5xx: El servidor ha recibido el comando pero es erróneo. No se puede continuar con la ejecución.

2. Segundo dígito: hace referencia al aspecto causante del éxito o fracaso. Se utilizan los siguientes códigos:
    - x0x: Sintaxis.
    - x1x: Solicitud de información, estado o ayuda.
    - x2x: Conexión de control o de datos.
    - x3x: Proceso de inicio de sesión.
    - x4x: Reservado.
    - x5x: Sistema de archivos.

3. Identifica distintos errores, agrupados según los significados del dígito uno y dos.

Combinando estos dígitos se consigue una serie de errores, algunos de los cuales podéis ver en la siguiente tabla:

Código	Descripción
101	Se reinicia la respuesta del comando.
120	El servicio estará listo en X minutos.
125	Conexión de datos abierta. Se inicia la transferencia.
200	Comando correcto.
202	Comando no soportado.
211	Información sobre el estado del sistema.
212	Información sobre el sistema de archivos.
213	Información sobre el archivo.
214	Mensaje de ayuda o información.
221	Se va a cerrar la conexión de control.
225	La conexión de datos está abierta, pero no hay ninguna transferencia en curso.
226	Se va a cerrar la conexión de datos. Transferencia terminada correctamente.
227	Entra en modo pasivo.
230	El usuario ha iniciado la sesión.
250	Petición de archivo correcta.
257	Archivo o directorio creado correctamente.
331	Nombre de archivo correcto. Todavía falta la contraseña.
425	No se puede abrir la conexión de datos.
426	Transferencia abortado y conexión cerrada
450	El archivo solicitado no existe.
452	Espacio libre en disco insuficiente.
500	Error de sintaxis, comando no reconocido.
501	Error de sintaxis en los parámetros.
502	Comando no implementado.
503	Secuencia de comandos incorrecta.
504	Comando no implementado con el parámetro pasado.
530	Sesión no iniciada.
532	Se necesita una cuenta para almacenar archivos.
552	Cuota de espacio en disco agotada.
553	Nombre de archivo incorrecto.

Modos del FTP: activo vs. pasivo

Una conexión FTP puede funcionar de dos modos: activo (el modo por defecto) y pasivo (conocido como *passive mode*).

Para entender cómo funcionan estos modos, debemos profundizar un poco más en el esquema cliente/servidor y explicar cómo se realiza paso a paso una conexión entre la máquina cliente y servidora.

Para entender los siguientes pasos, debemos comprender un concepto muy importante del protocolo TCP/IP: el concepto de **puerto.**

Un puerto es un identificador de conexión remota. No es más que un número, e identifica la conexión de tal modo que para conectarnos a una máquina remota, debemos utilizar un puerto que esté libre. Normalmente, se admiten números de puerto desde 1 hasta 65535, y los 1024 primeros están reservados para el sistema (esto depende el sistema operativo). Cuando nos conectamos a una máquina remota, debemos indicar la dirección o nombre de la máquina (dirección IP y nombre de dominio) y el número de puerto del servidor (el identificador de nuestra conexión). De este modo, cuando llega un dato a la máquina remota, sabe perfectamente a qué conexión pertenece dicho dato. Pensad que si no existiera el puerto, una máquina sólo podría atender una conexión remota simultáneamente, porque no sería capaz de diferenciar los datos de las distintas conexiones. Tanto el cliente como el servidor deben tener un puerto activo durante la conexión. Normalmente, el que inicia la conexión (cliente) decide el puerto del servidor al que quiere conectarse y es el sistema el que asigna un número de puerto libre para el cliente.
Espero que con esta pequeña introducción quede claro el concepto.

Ahora vamos allá con los pasos de una conexión típica en FTP.

1. El cliente arranca e intenta conectarse contra el servidor. Para ello crea en canal de control estableciendo una conexión física a la dirección IP (o nombre de dominio) del servidor, y desde cualquier puerto del cliente (puerto X) hacia el puerto 21 del servidor.

2. Una vez que está establecida la conexión, se envía el comando PORT al servidor, para especificar el número de puerto que debe utilizar el servidor para crear la conexión de datos (puerto Y).

3. Para ciertas operaciones, es necesario realizar el envío de un fichero, por lo que se crea el canal de datos. Para ello **el servidor** crea una conexión física desde su puerto 20 hasta un número de puerto del cliente. Este número de puerto fue el que indicó el cliente a través del comando PORT (puerto Y). De este modo, el servidor establece el canal de datos por el que se transmite la información.

Este esquema que hemos visto es el denominado **modo activo**, porque es el servidor el que activamente crea el canal de datos. En la **Figura 1** podéis ver un esquema de cómo y quién realizan las conexiones.

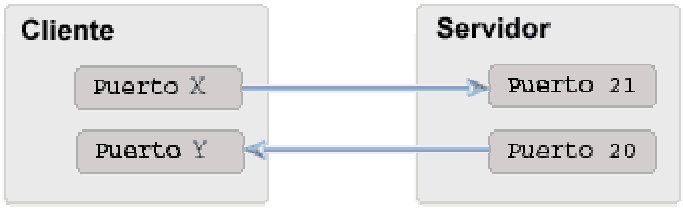

Figura 1

A continuación vamos a ver el modo pasivo y os daréis cuenta de la principal diferencia:

1. El cliente arranca e intenta conectarse contra el servidor. Para ello crea en canal de control estableciendo una conexión física a la dirección IP (o nombre de dominio) del servidor, y desde cualquier puerto del cliente (puerto X) hacia el puerto 21 del servidor. (este punto es igual que en el modo activo).

2. El cliente envía el comando PASV para activar el modo pasivo. Como respuesta a este comando, el servidor retorna un número de puerto que tenga disponible (puerto Z).

3. Para ciertas operaciones, es necesario realizar el envío de un fichero, por lo que se crea el canal de datos. Para ello **el cliente** crea una conexión física desde uno de sus puerto (puerto Y) hasta un número de puerto del cliente. Este número de puerto fue el que indicó el servidor como respuesta del comando PASV (puerto Z). De este modo, el cliente establece el canal de datos por el que se transmite la información.

Como habréis visto, la principal diferencia es que en el modo pasivo *es el cliente el que inicia las conexiones y nunca el servidor*. En la **Figura 2** podéis ver un esquema de este modo de conexión:

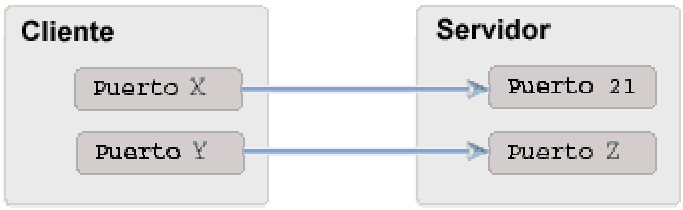

Figura 2

Esto, aunque parezca un detalle sin importancia, puede ser vital en algunas situaciones. Supongamos que nuestra conexión a internet utiliza un cortafuegos (hoy en día algo muy recomendable). El principal cometido del cortafuegos es cerrar los puertos para conexiones entrantes, es decir: que sobre ciertos puertos sea imposible realizar una conexión remota. Si el cliente FTP utiliza un cortafuegos, y se utiliza el modo activo de conexión, tendremos muchas posibilidades de que no funcione correctamente, ya que cuando el servidor necesite abrir el canal de datos, se podría utilizar uno de los puertos que el cortafuegos mantiene cerrado. Sin embargo, en el modo pasivo, un cortafuegos no supone ningún problema, ya que las conexiones siempre las inicia el cliente, y el servidor nunca intentará abrir una conexión con el cliente.

Así que, sabiendo esto, es muy recomendable (por no decir imprescindible) utilizar el modo pasivo si tenemos alguna sospecha de que puede haber sus cortafuegos en nuestra conexión, o si vemos que hay algún tipo de error en la conexión FTP.

FTP desde WinInet

Hasta ahora sólo hemos visto aspectos teóricos del protocolo FTP, que nos van a servir para tener una idea más exacta de lo que ocurre cuando utilizamos en API WinInet.

A partir de ahora vamos a centrarnos en las funciones de WinInet que debemos utilizar para implementar un cliente FTP. Estas funciones, a su vez, manejan todos los aspecto del protocolo tal y como hemos explicado, facilitándonos el trabajo a los programadores.

Algunos de los pasos a dar ya los hemos explicado, ya que son comunes con los pasos del protocolo HTTP.

Apertura de la instancia

Como ya sabemos, el primer paso que debemos hacer a la hora de trabajar con WinInet es llamar a la función InternetOpen. No voy a explicar esta función porque ya hemos profundizado en ella en el artículo sobre WinInet y HTTP (www.lawebdejm.com/?id=22220). Todo lo dicho allí nos vale para ahora, así que os animo a que deis un repaso a ese artículo.

Conexión con el servidor

Este paso también lo conocemos, ya que lo explicamos en profundidad en el artículo Más sobre WinInet y http (www.lawebdejm.com/?id=22230), cuando hablamos del método detallado de conexión para HTTP y la función InternetConnect.

En nuestro caso, el único cuidado que debemos tener es pasar los valores correctos a los siguientes parámetros:

- **nPuerto**: se debe pasar el valor numérico del puerto en el que está escuchando el servidor FTP. Normalmente se utiliza el puerto 21 (en la constante INTERNET_DEFAULT_FTP_PORT), aunque el administrador del servidor podría haber configurado cualquier otro puerto.
- **dwProtocolo**: indica el protocolo a utiliza en la conexión, en nuestro caso debe ser el valor INTERNET_SERVICE_FTP.
- **dwOpciones**: para conexiones FTP contamos con la opción INTERNET_FLAG_PASSIVE, que nos permiten realizar una conexión pasiva al servidor (tal y como ya hemos explicado). Este valor es muy recomendable si la conexión se realiza a través de un *firewall* o servidor *proxy*.

Como vemos, no son muchos los aspectos a tener en cuenta a la hora de llamar a InternetConnect para la conexión FTP.

Una vez que hemos establecido la conexión, ya podemos empezar a trabajar con el sistema de archivos del servidor, lo que básicamente se traduce en:

- Manipular la estructura de directorios
- Obtener información de archivos y carpetas
- Manipulación de archivos
- Descarga y envío de archivos
- Ejecutar comandos genéricos

Manipular la estructura de directorios

El equipo remoto tendrá una estructura de directorios como cualquier sistema operativo, en la cual nos podremos mover y manipular. Como en UNIX y MS-DOS, siempre estaremos situados en un directorio activo, en el cual se ejecutarán todas las operaciones que realicemos. Podemos averiguar cual es nuestro directorio activo a través de la función FtpGetCurrentDirectory:

```
function FtpGetCurrentDirectory(
             hConexión: HINTERNET;
             lpszDirectorioActivo: PChar;
             var lpdwLongitudDirectorio: DWORD
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszDirectorioActivo**: un puntero a una cadena en la que se almacenará la ruta absoluta del directorio activo. Hay que tener en cuenta que los servidores FTP utilizan el separador de UNIX (“/”) y no la barra invertida de Windows (“\”).
- **lpdwLongitudDirectorio**: un puntero a un valor de 32 bits en el que se pasa la longuitud máxima de la cadena lpszDirectorioActivo. Cuando la función retorna, en este valor se almacena el número de caracteres copiados.

La función retornará TRUE o FALSE dependiendo del éxito o fracaso.

Como es lógico, además de averiguar el directorio activo, también podremos cambiar este directorio activo, a través de la función SetCurrentDirectory:

```
function FtpGetCurrentDirectory(
             hConexión: HINTERNET;
             lpszNuevoDirectorio: PChar
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszNuevoDirectorio**: un puntero a una cadena en la que se almacenará la ruta (absoluta o relativa) del nuevo directorio activo.

La función retorna TRUE si ha tenido éxito o FALSE en caso de error. En caso de retornar error, una de las posibles causas es que no tenemos permisos para acceder a dicho directorio. En tal caso, podemos hacer uso de la función InternetGetLastResponseInfo (de la que ya hablamos en el artículo WinInet y http - www.lawebdejm.com/?id=22220) para obtener el código de error retornado por el servidor.

Si en un momento dado queremos crear un nuevo directorio, debemos hacerlo a través de la función FtpCreateDirectory, con la siguiente sintaxis:

```
function FtpCreateDirectory(
             hConexión: HINTERNET;
             lpszNombreDirectorio: PChar
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszNuevoDirectorio**: un puntero a una cadena en la que se indica la ruta (absoluta o relativa) del directorio a crear. Para dar el nombre a la carpeta hay que tener en cuenta las normas del sistema de archivos del servidor, normalmente UNIX o Windows.

La función retorna TRUE o FALSE. Podremos llamar, como ya hemos dicho, a la función InternetGetLastResposeInfo para averiguar el código y texto del mensaje de error retornado por el servidor.

Y lógicamente, también se podrá borrar un directorio (y todo su contenido) a través de la siguiente función:

```
function FtpRemoveDirectory(
             hConexión: HINTERNET;
             lpszDirectorio: PChar
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszDirectorio**: un puntero a una cadena en la que se indica la ruta (absoluta o relativa) del directorio a borrar. Es importante saber que no podemos borrar el directorio si es nuestro directorio activo, por lo que es recomendable hacer una llamada a FtpSetCurrentDirectory antes de realizar el borrado.

Esta función, como el resto, retornará TRUE o FALSE dependiendo del éxito o fracaso.

Obtener información de archivos y carpetas

Otro tipo de operaciones que podemos realizar con el protocolo FTP es la obtención de información de archivos y carpetas remotos.

La única información que podemos obtener de un archivo es su tamaño, a través de la función FtpGetFileSize.

Pero antes de llamar a esta función, es necesario obtener un descriptor de un archivo remoto, a través de la función FtpOpenFile:

```
function FtpOpenFile(
             hConexión:    HINTERNET;
             lpszArchivo:  PChar;
             dwTipoAcceso: LongWord;
             dwOpciones:   LongWord;
             dwContexto:   LongWord
         ): HINTERNET;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszArchivo**: un puntero a una cadena en la que se pasa la ruta (absoluta o relativa) del archivo que queremos abrir.
- **dwTipoAcceso**: el tipo de acceso que se va utilizar con fichero. Se puede utilizar el valor GENERIC_READ o GENERIC_WRITE.
- **dwOpciones**: las opciones de acceso al fichero, desde el punto de vista de el tipo de transmisión:
    - FTP_TRANSFER_TYPE_ASCII: utiliza el tipo de transferencia ASCII propia de los servidores FTP.
    - FTP_TRANSFER_TYPE_BINARY: utiliza el tipo de transferencia binaria propia de los servidores FTP.
    - FTP_TRANSFER_TYPE_UNKNOWN: igual que FTP_TRANSFER_TYPE_ASCII.
    - INTERNET_FLAG_TRANSFER_ASCII: utiliza codificación ASCII estándar.
    - INTERNET_FLAG_TRANSFER_BINARY: utiliza codificación binaria estándar.

Y desde el punto de vista del caché interno se puede configurar los siguientes aspectos:

- INTERNET_FLAG_HYPERLINK: fuerza a cargar si no hay datos sobre la caducidad o la última modificación.
- INTERNET_FLAG_NEED_FILE: guarda el contenido en un archivo temporal si no ha podido guardarse en el caché interno.
- INTERNET_FLAG_RELOAD: vuelve a descargar el archivo, aunque esté en el caché.

- **dwContexto**: el valor de contexto que queramos pasar al llamar a la función de *callback*.

La función retorna el descriptor del archivo si todo ha ido correctamente, o NULL si ha ocurrido algún tipo de error. Se puede llamar a GetLastError para averiguar las causas del error.

Cuando ya no necesitemos más el descriptor de archivo, debemos llamar a InternetCloseHandle para cerrarlo.

Entre la llamada a FtpOpenFile e InternetCloseHandle, no podemos llamar a ninguna otra función de WinInet, utilizando el mismo descriptor de conexión, ya que el canal de control permanece ocupado. Si hacemos una llamada a otra función (por ejemplo FtpCreateDirectory o cualquier otra) con la misma conexión, obtendremos el error ERROR_FTP_TRANSFER_IN_PROGRESS.
Una consecuencia de esto es que sólo podemos mantener abierto un archivo por cada conexión FTP.

Ahora que ya sabemos cómo obtener un descriptor de archivo con FtpOpenFile, podemos explicar el modo de utilizar la función FtpGetFileSize:

```
function FtpGetFileSize(
             hArchivo:       HINTERNET;
             lpdwTamañoHigh: ^LongWord
         ): LongWord;
```

- **hArchivo**: un descriptor de archivo obtenido a través de FtpOpenFile.
- **lpdwTamañoHigh**: un puntero a un valor de 32 bits en el que se almacenarán los 32 bits más altos del tamaño total del archivo. En caso de que el tamaño del archivo pueda almacenarse en un espacio de 32 bits (menos de 2 GB.), en este valor no se copiará nada.

La función retornará los 32 bits más bajos del tamaño de archivo que pueden combinarse con el parámetro lpdwTamañoHigh para obtener un valor de 64 bits con el tamaño total del archivo.

Para aquellos que no hayáis trabajado nunca a nivel de bits, vamos a hacer una pequeña pausa para explicar el modo de combinar ambos valores, a través de los operadores de desplazamiento de bits. En nuestro caso, desplazaremos 32 bits a la izquierda el valor *high* y haciendo un *or* lógico con el valor *low*, más o menos del siguiente modo:

1. Paso 0: valores con los que vamos a trabajar:

```
High:
    Decimal:  3991676535
    Binario:  11101101111011000010011001110111
Low:
    Decimal:  2863659011
    Binario:  10101010101011111111100000000011
Total:
    Decimal:  17144120176899258371
    Binario:  1110110111101100001001100111011110101010101011111111100000000011
```

2. Paso 1: desplazar el valor *high* 32 bits a la izquierda (se crea un valor de 64 bits). Al desplazar un número de bits a la izquierda, se rellena los espacio sobrantes con ceros.

```
0000000000000000000000000000000011101101111011000010011001110111
desplazado 32 bits da:
1110110111101100001001100111011100000000000000000000000000000000
```

3. Paso 2: combinar con el valor *high* con el valor *low*, a través de un operación *or* lógica. Como sabéis, la operación *or* lógica nos retornará 1 cuando alguno de los operandos sea 1.

```
High:     11101101111011000010011001110111000000000000000000000000000000
       or
Low:      00000000000000000000000000000001010101010101111111110000000011
          ----------------------------------------------------------------
Total:    11101101111011000010011001110111010101010101111111110000000011
Decimal:  17144120176899258371
```

Como veis, a partir de dos valores de 32 bits, se ha creado otro de 64 bits que contiene ambos, en la parte alta y baja.

Fácil ¿verdad? De todas formas, esta misma operación que hemos realizado a nivel de bits, se puede realizar aritméticamente más fácilmente, aplicando la siguiente ecuación:

```
total = (high * 4294967296) + low;
```

El valor 4294967296 se obtiene de sumar 1 al valor máximo para un rango de 32 bits.

Estas dos opciones, en nuestros lenguajes de programación se puede implementar muy fácilmente:

```
var
   SizeHigh: integer;
   SizeLow:  integer;
   total: Int64;      { ¡¡un numérico de 64 bits!! }
begin
   total := (SizeHigh shl 32) or SizeLow; { a nivel de bits }
   total := (SizeHigh * ($FFFFFFFF + 1)) + SizeLow; { aritmeticamente }
end;
```

Bueno, después de esta pequeña interrupción (que espero que os haya servido para entender algo más las operaciones a nivel de bits) vamos a retomar el tema que nos ocupa.

Nos quedamos en la obtención de información de los archivos y carpetas del servidor. Ya sabemos que la única información que se puede obtener de un archivo es su tamaño, con la función FtpGetFileSize. Para utilizar esta función debemos obtener un descriptor de archivo a través de FtpOpenFile, y a su vez, para llamar a esta debemos conocer el nombre del archivo al que queremos acceder. ¿Y cómo sabemos el nombre de los archivos que tiene el servidor? Pues aquí es donde enlazamos con lo siguiente: obtener información de las carpetas.

Básicamente, la información que podemos obtener de una carpeta es un listado de los archivos y otras carpetas que contiene. Para ello debemos hacer uso de dos funciones: FtpFindFirstFile para iniciar la búsqueda e InternetFindNextFile para recorrer la lista de resultados.
Vamos allá:

```
function FtpFindFirstFile(
               hConexión:    HINTERNET;
               lpszArchivo:  PChar;
               var lpDatos:  WIN32_FIND_DATA;
               dwOpciones:   LongWord;
               dwContexto:   LongWord
            ): HINTERNET;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszArchivo**: un puntero a una cadena en la que se pasa el nombre de archivo a encontrar. En este parámetro se puede pasar el valor NULL, para obtener un listado de todos el contenido del directorio activo, o bien utilizar los comodines que todos conocemos: “*” y “?”. Si se pasa un nombre de archivo, o una máscara con comodines, se puede pasar tanto la ruta absoluta como relativa.
- **lpDatos**: un puntero a una estructura de tipo WIN32_FIND_DATA en la que se almacena los datos del archivo encontrado. La estructura tiene los siguientes campos.

```
WIN32_FIND_DATA = record
    dwFileAttributes:  LongWord;
    ftCreationTime:    FILETIME;
    ftLastAccessTime:  FILETIME;
    ftLastWriteTime:   FILETIME;
    nFileSizeHigh:     LongWord;
    nFileSizeLow:      LongWord;
    dwReserved0:       LongWord;
    dwReserved1:       LongWord;
    cFileName:         array[0..259] of char;
    cAlternateFileName: array[0..13] of char;
end;
```

Estos campos tienen los siguientes significados:

- **dwFileAttributes**: indica los atributos que tiene asignados el archivo encontrado. Este valor es una máscara de bits en la que pueden aparecen los siguientes valores:
  - FILE_ATTRIBUTE_ARCHIVE: indica que se debe marcar como "archivado".
  - FILE_ATTRIBUTE_COMPRESSED: indica que los datos están comprimidos (sólo validos para sistemas de archivos NTFS).
  - FILE_ATTRIBUTE_HIDDEN: el archivo está oculto.
  - FILE_ATTRIBUTE_NORMAL: no tiene otros atributos asignados.
  - FILE_ATTRIBUTE_READONLY: el archivo está marcado como "sólo lectura".
  - FILE_ATTRIBUTE_SYSTEM: el archivo es de sistema.
  - FILE_ATTRIBUTE_TEMPORARY: el archivo se ha marcado como temporal.
- **ftCreationTime**: fecha en que se creó el archivo.
- **ftLastAccessTime**: fecha de último acceso al archivo.
- **ftLastWriteTime**: fecha de la última modificación de archivo.
- **nFileSizeHigh**: 32 bits más altos del tamaño del archivo.
- **nFileSizeLow**: 32 bits más bajos del tamaño del archivo.
- **dwReserved0** y **dwReserved1**: no se usan.
- **cFileName**: el nombre completo del archivo.
- **cAlternameFileName**: el nombre corto del archivo, con un máximo de 8 caracteres para el nombre y 3 para la extensión.

- **dwOpciones**: configura el comportamiento de la función. Pueden utilizarse cualquier combinación de los siguientes valores.
  - INTERNET_FLAG_HYPERLINK: fuerza a cargar si no hay datos sobre la caducidad o la última modificación.
  - INTERNET_FLAG_NEED_FILE: guarda el resultado de la búsqueda en un archivo temporal si no ha podido guardarse en el caché interno.

  - INTERNET_FLAG_RELOAD: vuelve a descargar el resultado de la búsqueda, aunque esté en el caché.
  - INTERNET_FLAG_NO_CACHE_WRITE: no almacena el resultado de la búsqueda en el caché local.

- **dwContexto**: el valor de contexto que queramos pasar al llamar a la función de *callback*.

La función retorna un descriptor de búsqueda, que nos servirá para las siguientes llamadas a InternetFindNextFile. En caso de error, la función retorna NULL y en este caso, podemos llamar a GetLastError para averiguar la causa del error. Os recuerdo, como ya sabéis todos, que si GetLastError retorna ERROR_INTERNET_EXTENDED_ERROR, debemos llamar a la función InternetGetLastResponseInfo para conseguir más datos sobre el error.

Una vez conseguido el descriptor de la búsqueda (el valor retornado por FtpFindFirstFile), la estructura WIN32_FIN_DATA tendrá los valores del primer archivo que cumpliese el criterio de búsqueda. Si queremos buscar el siguiente archivo, debemos hacer uso de la función InternetFindNextFile.

Os habréis fijado que esta función no empieza por “Ftp”sino por “Internet”. Hace ya unos cuantos meses, en el artículo “Introducción al API WinInet”, dijimos que las funciones comenzaban por el nombre del protocolo con el que trabajaban (Http, Ftp o Gopher) excepto las que comenzaban por “Internet”, que eran para uso general o servían para varios protocolos. Este es el caso de InternetFindNextFile, que sirve tanto para una conexión FTP como Gopher. Como no explicaremos el procolo Gopher, sólo explicaré el uso de esta función aplicada al protocolo FTP. La sintaxis es la siguiente:

```
function InternetFindNextFile(
            hBúsqueda: HINTERNET;
            lpvDatos:  Pointer
        ): LongBool;
```

- **hBúsqueda**: un descriptor de búsqueda conexión obtenido a través de FtpFindFirstFile.
- **lpDatos**: un puntero a una estructura de tipo WIN32_FIND_DATA en la que se almacena los datos del archivo encontrado. En realidad este valor es un puntero genérico, pero para conexiones FTP debe pasarse un puntero a esta estructura.

La función retorna TRUE o FALSE dependiendo del éxito o error. Cuando retorne FALSE, debe comprobarse el valor de GetLastError, y si retorna el valor ERROR_NO_MORE_FILES significará que no se han encontrado más archivos que cumpliesen la condición de búsqueda.

Sabiendo esto, el uso más típico de estas funciones es el que aparece a continuación:

```
var
   datos:    WIN32_FIND_DATA;
   busqueda: HINTERNET;
   ok:       boolean;
begin
   busqueda := FtpFindFirstFile(hConexion, nil, @datos, 0, 0);
   try
      ok := (busqueda <> nil);
      while ok do
      begin
         { aquí se procesa el archivo encontrado }
         ok := InternetFindNextFile(busqueda, @datos);
      end;

      if GetLastError() <> ERROR_NO_MORE_FILES then
         MessageBox(GetActiveWindow(), 'Error buscando archivos en FTP.',
                    'Error', MB_ICONERROR);
   finally
      if busqueda <> nil then
         InternetCloseHandle(busqueda);
   end;
end;
```

Manipulación de archivos

Antes de llegar a lo realmente importante (descarga y envío de archivos), lo último que nos queda por ver antes es la manipulación de archivos: renombrado y borrado de archivos.

El renombrado de un archivo remoto se hace a través de la función FtpRenameFile:

```
function FtpRenameFile(
            hConexión:        HINTERNET;
            lpszNombreActual: PChar;
            lpszNombreNuevo:  PChar
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszNombreActual**: un puntero a una cadena en la que se pasa el nombre del archivo que queremos renombrar. El nombre de archivo puede incluir la ruta, absoluta o relativa. Si no se indica alguna ruta, se buscará el archivo en el directorio activo. Además de archivos, esta función admite el renombrado de directorios, por lo que en este parámetro se puede pasar tanto un nombre de archivo como de directorio.
- **lpszNombreNuevo**: un puntero a una cadena en la que se pasa el nuevo nombre que queremos darle al archivo o directorio. Este nombre, al igual que con el anterior parámetro, puede incluír la ruta absoluta o relativa del nuevo nombre. Si esta ruta es distinta a la ruta especificada en el nombre original (o el directorio activo si no se especificó ninguna), el archivo, además de ser renombrado, se moverá al directorio que indiquemos. Es por esto que esta función, además de renombrar archivos, sirve para moverlos de directorio.

Esta función retorna TRUE si todo ha ido bien, o FALSE en caso de error.

Además de renombrar o mover un archivo, podemos eliminarlo a través de la función FtpDeleteFile:

```
function FtpDeleteFile(
            hConexión:         HINTERNET;
            lpszNombreArchivo: PChar
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszNombreArchivo**: un puntero a una cadena en la que se pasa el nombre del archivo que queremos eliminar. Opcionalmente, se puede incluir la ruta absoluta o relativa del archivo.

Esta función, como otras muchas, retorna TRUE o FALSE.

Descarga y envío de archivos

Como es lógico, la tarea más importante a la hora de conectarse a un servidor FTP es la descarga y envío de archivos, y ahora que ya sabemos renombrar, mover y eliminar archivos, podemos centrarnos en esta parte del protocolo FTP.

Tanto la descarga como en envío podemos realizarlos de dos formas, una directa (con una única llamada a una función) y otra al estilo de la lectura de archivos locales: con apertura, lectura y cierre del fichero.

La descarga con el método directo puede hacerse a través de la función FtpGetFile:

```
function FtpGetFile(
            hConexión:         HINTERNET;
            lpszArchivoRemoto: PChar;
            lpszArchivoLocal:  PChar;
            fErrorSiExiste:    LongBool;
            dwAtributos:       LongWord;
            dwOpciones:        LongWord;
            dwContexto:        LongWord
         ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszArchivoRemoto**: un puntero a una cadena en la que se pasa el nombre del archivo remoto que queremos descargar. Se puede incluir la ruta absoluta o relativa.
- **lpszArchivoLocal**: un puntero a una cadena en la que se pasa el nombre del archivo local donde se almacenará el contenido del archivo remoto. Se puede incluir la ruta absoluta o relativa.
- **fErrorSiExiste**: indica si la función fallará si el archivo local ya existe, o si el archivo local debe ser sobrescrito.
- **dwAtributos**: indica el conjunto de atributos que se establecerá al archivo local. Se puede incluír cualquier valor de los admitidos por el atributos dwFileAttributes, dentro de la estructura WIN32_FIND_DATA (que ya hemos explicado anteriormente).
- **dwOpciones**: indica el conjunto de opciones que controlan la descarga del archivo. Se puede utilizar cualquier combinación de los valores que explicamos para el parámetro dwOpciones de la función FtpOpenFile.
- **dwContexto**: el valor de contexto que queramos pasar al llamar a la función de callback.

La función retorna TRUE o FALSE, dependiendo de su éxito o fracaso.

Por el otro lado, podemos realizar en envío de un fichero a un servidor FTP a través de la función FtpPutFile:

```
function FtpPutFile(
            hConexión:         HINTERNET;
            lpszArchivoLocal:  PChar;
            lpszArchivoRemoto: PChar;
            dwOpciones:        DWORD;
            dwContexto:        DWORD
        ): LongBool;
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **lpszArchivoLocal**: un puntero a una cadena en la que se pasa el nombre del archivo local que se quiere enviar al servidor. Se puede incluir la ruta absoluta o relativa.
- **lpszArchivoRemoto**: un puntero a una cadena en la que se pasa el nombre del archivo remoto donde queremos almacenar la información a enviar. Se puede incluir la ruta absoluta o relativa.
- **dwContexto**: el valor de contexto que queramos pasar al llamar a la función de *callback*.

La función retorna TRUE o FALSE, dependiendo de su éxito o fracaso.

Como dijimos antes, podemos descargar o enviar un archivo a través del método directo (con las funciones FtpGetFile y FtpPutFile) o bien hacerlo como si de un archivo local se tratase, siguiendo los siguientes pasos:

1. Apertura del archivo origen y remoto. En caso de una descarga, el archivo origen será un archivo remoto, y el archivo destino, será uno local, creado vacío.
2. Lectura del archivo origen mientras queden datos y grabación de los datos leídos en el archivo destino.
3. Cierre del fichero origen y destino.

Para la apertura de archivos locales, podemos hacer uso de la función CreateFile, tal y como explicamos durante nuestro artículo sobre "Archivos proyectados en memoria" (www.lawebdejm.com/?id=22140). Si se trata de abrir un archivo un remoto en un servidor FTP, tenemos que hacer uso de la función FtpOpenFile, que hemos explicado en este mismo artículo.

Para la lectura y grabación en archivos locales, tendremos que recurrir a las funciones ReadFile y WriteFile respectivamente, mientras que si se trata de leer o escribir en un archivo remoto, tenemos las funciones InternetReadFile e InternetWriteFile, que ya explicamos durante el artículo "WinInet y HTTP" (www.lawebdejm.com/?id=22220). Si estamos leyendo de un archivo remoto, es recomendable también utilizar la función InternetQueryDataAvailable, tal y como explicamos en el artículo "Más sobre WinInet y HTTP" (www.lawebdejm.com/?id=22230).

Este otro método, aunque más complejo y laborioso, es el recomendable a la hora de mostrar un indicador de progreso en la descarga de un archivo ya que podemos averiguar el tamaño total del archivo (con FtpGetFileSize) y controlamos completamente el proceso de lectura *byte a byte* del archivo remoto.

Hay que tener cuidado con este punto, ya que la apertura de archivos remoto requiere de conexiones a través puertos especiales. Si nuestra conexión a internet se hace a través de un cortafuegos o un servidor proxy, es muy posible que estas conexiones se dos denieguen. Para solucionar este problema debemos realizar la conexión siguiendo el modo pasivo, tal y como explicamos anteriormente.

No voy a explicar las funciones, ya que profundizamos en ellas cuando hablamos del protocolo HTTP (en los artículos que he indicado antes).

Ejecutar comandos genéricos

Si dentro de todas estas funciones no encuentras la que necesitas, *"no problemo"*, que todavía hay más. Puedes utilizar la función FtpCommand para ejecutar cualquier tipo de comando admitido por el servidor FTP. Eso sí, esta función sólo está disponible si tienes instalado Internet Explorer 5.0 o superior.

```
function FtpCommand(
                  hConexión:      HINTERNET;
                  fHayRespuesta:  LongBool;
                  dwOpciones:     LongWord;
                  lpszCommando:   PChar;
                  dwContexto:     LongWord;
                  var hRespuesta: HINTERNET
               ): LongBool;
```

```
Nota: el módulo WinInet.pas de Delphi contiene un error en la definición de esta función.
La cabecera correcta es la que aquí aparece, y si quieres utilizar FtpCommand desde tus programas,
debes importar  la función de forma dinámica, o bien del siguiente modo:
          function FtpCommand(
              hConexion:      HINTERNET;
              fHayRespuesta:  BOOL;
              dwOpciones:     DWORD;
              lpszCommando:   PChar;
              dwContexto:     DWORD;
              var hRespuesta: HINTERNET): BOOL; stdcall;
          external 'wininet.dll' name 'FtpCommandA';
```

- **hConexión**: un descriptor de conexión obtenido a través de InternetConnect.
- **fHayRespuesta**: un valor booleano que indica si el comando retorna algo. En caso de pasarse el valor TRUE, se copiará en phRespuesta el descriptor de la petición creada.
- **dwOpciones**:
  - FTP_TRANSFER_TYPE_ASCII: utiliza el tipo de transferencia ASCII propia de los servidores FTP. Realiza las conversiones de la tabla ASCII correspondiente.
  - FTP_TRANSFER_TYPE_BINARY: utiliza el tipo de transferencia binaria propia de los servidores FTP. El contenido del archivo no se cambia.
- **lpszComando**: una cadena con el comando a ejecutar. Puedes consultar una lista con algunos de los comandos en la tabla que puedes encontrar al principio del artículo.
- **dwContexto**: el valor de contexto que queramos pasar al llamar a la función de *callback*.
- **phRespuesta**: se trata de un puntero a un descriptor donde se copiará el descriptor obtenido de crear la petición. Este descriptor se utiliza para leer la respuesta con la función InternetReadFile. Hay que recordar que este descriptor, como cualquier otro, debe ser cerrado con InternetCloseHandle una vez que ya no lo necesitemos.

La función retorna TRUE o FALSE dependiendo de su éxito. Como sabemos, un comando FTP retorna un código de respuesta, que podemos obtener a través de InternetGetLastResponseInfo, y si además retorna un resultado (como un archivo, un texto, etc.) podemos leer del canal de datos a través de InternetReadFile, utilizando el descriptor obtenido en phRespuesta.

Para utilizar esta función es necesario conocer bien los comandos FTP, saber cómo se llaman y qué retornan, así que podemos decir que se trata de una función para avanzados.

De todas formas, como introducción y para que veáis cómo se utiliza, podéis echar un vistazo al siguiente código, donde se hace un uso muy sencillo de FtpCommand para obtener el listado de archivos en un directorio, a través del comando NLST:

```
var
   respuesta:  HINTERNET;
   leido:      DWORD;
   disponible: DWORD;
   buffer:     PChar;
begin
   if FtpCommand(<<descriptor de conexión>>, true, FTP_TRANSFER_TYPE_ASCII,
                 PChar('NLST'), 0, respuesta) then
   begin
      try
         InternetQueryDataAvailable(respuesta, disponible, 0, 0);

         buffer := AllocMem(disponible + 1);
         try
            InternetReadFile(respuesta, buffer, disponible, leido);
            MessageBox(GetActiveWindow, buffer, 'Comando NLST',
                       MB_ICONINFORMATION);
         finally
            FreeMem(buffer);
         end;
      finally
         InternetCloseHandle(respuesta);
      end;
   end;
end;
```

Una forma más fácil

Después de explicar en profundidad cómo utilizar el protocolo FTP, os voy a confesar un secreto: existe un método mucho más sencillo de acceder a un recurso y descargarlo.

¿Recordáis el método directo para el protocolo HTTP? ¿Y la función InternetOpenUrl? Pues en FTP también podemos hacer uso de esta función. Simplemente pasando una URL que comience por “ftp://”, la función realizará todo el trabajo sucio y nos retornará un descriptor al recurso que hemos pasado por parámetro. Una vez que tenemos ese descriptor, podemos hacer uso de él a través de la función InternetReadFile, y debemos cerrarlo con InternetCloseHandle una vez hayamos terminado.

No voy a entrar en más detalles porque ya explicamos el uso de esta función cuando hablamos del protocolo HTTP. De todas formas, en el siguiente listado tenéis un ejemplo muy sencillo de uso, para que veáis que en realidad no hay nada nuevo que explicar:

```
var
   hURL:       HINTERNET;
   disponible: DWORD;
   leido:      DWORD;
   buffer:     PChar;
begin
   hURL := InternetOpenUrl(<<descriptor obtenido con InternetOpen>>,
                  'ftp://ftp.borland.com/pub/delphi/devsupport/general/index.txt',
                  nil, 0, 0, 0);
   if hURL <> nil then
   begin
      InternetQueryDataAvailable(hURL, disponible, 0, 0);

      buffer := AllocMem(disponible + 1);
      try
         InternetReadFile(hURL, buffer, disponible, leido);
         MessageBox(GetActiveWindow, buffer, 'Contenido', MB_ICONINFORMATION);
      finally
         FreeMem(buffer);
      end;
   end;
end;
```

Conclusión

Y esto ya está terminando. Como veis, el uso del protocolo FTP y hemos sido capaces de resumirlo todo en un solo artículo. Espero que haya quedado todo lo suficientemente claro, y que estas pequeñas indicaciones os permitan empezar a desarrollar vuestros clientes de FTP.

En el siguiente artículo vamos a profundizar sobre el caché interno de WinInet (que es el mismo que el de Internet Explorer) y las funciones que podemos utilizar para consultarlo y modificarlo.

Los ejemplos

Delphi 5

Junto con este artículo, podéis estudiar el ejemplo escrito en Delphi. Se trata de un cliente FTP semi-completo, que realiza conexiones a un servidor FTP, descarga y envía archivos, consulta las propiedades de un archivo, etc. Podéis ir viendo los distintos métodos de la clase TFtpConnection, en los que se realizan las llamadas al API WinInet que hemos ido explicando durante el artículo. Además, y a modo de repaso, podéis ver cómo hacer uso de la función de *callback* utilizando el API WinInet.

Fuentes en ZIP - www.lawebdejm.com/?id=22240

Autor: JM - http://www.lawebdejm.com