Los rincones del API Win32
WinInet y HTTP (I)

En el artículo anterior dimos una panorámica general de Wininet, explicando su cometido, sus principales usos y dando una descripción detallada de las funciones de uso general: manipulación de URLs, estado de la conexión, marcado del módem y gestión de cookies. En esta ocasión vamos a lo realmente importante de Wininet: la descarga de recursos desde internet, concretamente utilizando el protocolo de aplicación HTTP.

Introducción al protocolo HTTP

Como ya dijimos, Wininet soporta tres protocolos de aplicación: HTTP, FTP y Gopher. Sin duda alguna, podemos decir que el más importante y utilizado de todos ellos es HTTP, ya que todos los sitios web visitados (y la mayoría de los archivos descargados desde el navegador) de internet son recuperados a través de este protocolo.

HTTP permite la transferencia de múltiples tipos de información de una forma eficiente, haciéndolo idóneo para una red tan hetereogénea como internet, donde los formatos en que se presenta la información son muy variados e impredecibles (páginas HTML, programas, imágenes, sonidos, videos, películas Flash, futuros formatos, etc.).

La estructura del protocolo HTTP es sencilla, más o menos como la de cualquier arquitectura cliente/servidor:
La máquina cliente establece una conexión (normalmente a través del protocolo de transporte TCP/IP, al puerto 80) con la máquina servidora, donde está ejecutándose un software llamado servidor web o servidor HTTP (Apache o Internet Information Server son algunos de ellos).
Una vez establecida la conexión, el cliente (por ejemplo el navegador web) envía tramas de datos que consisten en unas cabeceras especiales y una petición, que es recibida al otro lado de la conexión por el servidor HTTP.
Este servidor interpretará la petición del cliente, devolviendo un resultado, que dependerá del tipo de petición recibida.
Una vez que la respuesta ha sido enviada, la conexión se pierde. Es por esto que el protocolo HTTP se denomina "sin conexión", al contrario que otros como FTP que mantienen una conexión abierta continuamente. Además de ser "sin conexión", también se suele decir que es un protocolo "sin estado", ya que no tiene en cuenta peticiones anteriores de un mismo cliente, y considera que cada petición es única e independiente del resto.

Todo este trasiego de información se hace a través de un formato de texto de 8 bits, que está definido según el protocolo HTTP, en el RFC 1945 (http://www.ietf.org/rfc/rfc1945.txt) .

Aunque el API Wininet nos oculta los aspectos internos del protocolo, y no es necesario conocerlos para trabajar con HTTP, vamos a explicar superficialmente en qué consiste cada petición-respuesta en este protocolo.

Una petición típica suele ser algo así como esto (al principio de la línea de indica un número entre paréntesis para la explicación posterior):

```
(1) GET /index.htm HTTP/1.0
(2) Accept: text/html
    Accept: video/mpg
    Accept: image/*
    Accept: */*
(3) User-Agent: Mozilla/3.0
(4)
```

1. Operación, archivo solicitado y versión del protocolo que se está utilizando. En la versión 1.0 del protocolo, sólo se admitían 3 tipos de operaciones:

   - GET: se utiliza para recuperar el contenido de un recursos estático (página HTML, imágen, vídeo, etc), o bien ejecutar un programa o script en el servidor (como programas CGI, páginas ASP, JSP, PHP, etc.). Es la operación más común, que se lanza cada vez que pulsamos en un enlace y navegamos a una página web.

   - HEAD: es una operación especial que tan sólo nos recupera información del recurso, como el tamaño, la fecha de modificación, tipo, etc. Lo suelen utilizar los navegadores o servidores proxy para comprobar el estado de su caché u otras operaciones.

   - POST: envía información desde el cliente al servidor web, como pueden ser los datos de un formulario.

   En la versión 1.1 del protocolo se introdujeron nuevas operaciones, como PUT (almacenar recursos en el servidor), DELETE (borrar recursos del servidor), LINK (establecer enlaces entre documentos), OPTIONS (determinar las capacidades del servidor), etc.

2. Tipos MIME que acepta el equipo cliente. Los tipos MIME (Multipurpose Internet Mail Extensions) son un estándar para el envío de información binaria a través de caracteres alfanuméricos. Este estándar permite que, a través del protocolo HTTP (que maneja información en modo texto), podamos transferir archivos no-textuales, como pueden ser imágenes, audio, vídeo, programas ejecutables etc. Los tipos MIME definen grupos (antes del carácter "/") y tipos (después del carácter "/"). Así el tipo MIME "text/html" define a todos los archivos de texto que contienen código HTML, el tipo "video/mpeg" define a todos los archivos de vídeo almacenados en formato mpeg, etc. Para indicar cualquier tipo se puede utilizar el carácter "*", tanto en el tipo como en el grupo. De este modo, el tipo MIME "image/*" representa a todos los archivos de imagen, ya estén almacenadas en formato gif, jpeg, bmp, etc.

3. Aplicación cliente que está lanzando la petición HTTP.

4. La última línea debe estar en blanco, para indicar el final de la petición.

La respuesta a esta petición puede ser algo parecido a la siguiente:

```
(1) HTTP/1.0 200 OK
(2) Date: Saturday, 19-May-01 22:30:00 GMT+01
(3) Server: Apache
(4) MIME-version: 1.0
(5) Content-type: text/html
(6) Content-length: 254
(7) Last-modified: 29-May-00 10:30:00
(8)
(9) <html>
      <head>
        <title>La web de JM</title>
      </head>
      <body>
      ...
    </html>
```

1. Versión del protocolo que utiliza el servidor HTTP, junto con el código de estado de la petición, y una descripción de este estado. En nuestro caso, todo ha ido bien (200 OK). Los códigos de retorno están compuestos por tres dígitos se dividen en 5 categorías, dependiendo de la naturaleza del estado. La forma de diferencias cada grupo es por el primer dígito del código:
   - 1xx: mensajes informativos. En la versión 1.0 del protocolo no se utilizan.
   - 2xx: mensajes de operaciones correctas.
   - 3xx: mensajes de redirección.
   - 4xx: mensajes de error en el cliente.
   - 5xx: mensajes de error en el servidor.

   Para una descripción detallada de todos los códigos de retorno, debe consultarse el documento RFT número 1945 (http://www.ietf.org/rfc/rfc1945.txt)
2. Fecha en que se ha recibido la petición.
3. Tipo y versión del servidor HTTP que ha generado la respuesta.
4. Versión de MIME que maneja el servidor HTTP.
5. Tipo MIME de los datos retornados. En este caso se va a retornar una página HTML (text/html).
6. Tamaño (en bytes) de los datos retornados.
7. Fecha de la última modificación de los datos.
8. Una línea en blanco que separa la cabecera de los datos retornados.
9. Aquí comienzan los datos según el formato que se indicó en "Content-type". En nuestro caso se trata de una página HTML pero podría tratarse de cualquier otro tipo, como una imagen, un programa ejecutable, un video, siempre codificado como caracteres alfanuméricos.

Al bloque de información hasta la línea en blanco (los puntos del 1 al 7) se denomina "cabecera". A partir de la línea en blanco, aparecerá la información que hemos solicitado, en el formato que indique el valor "Content-type".

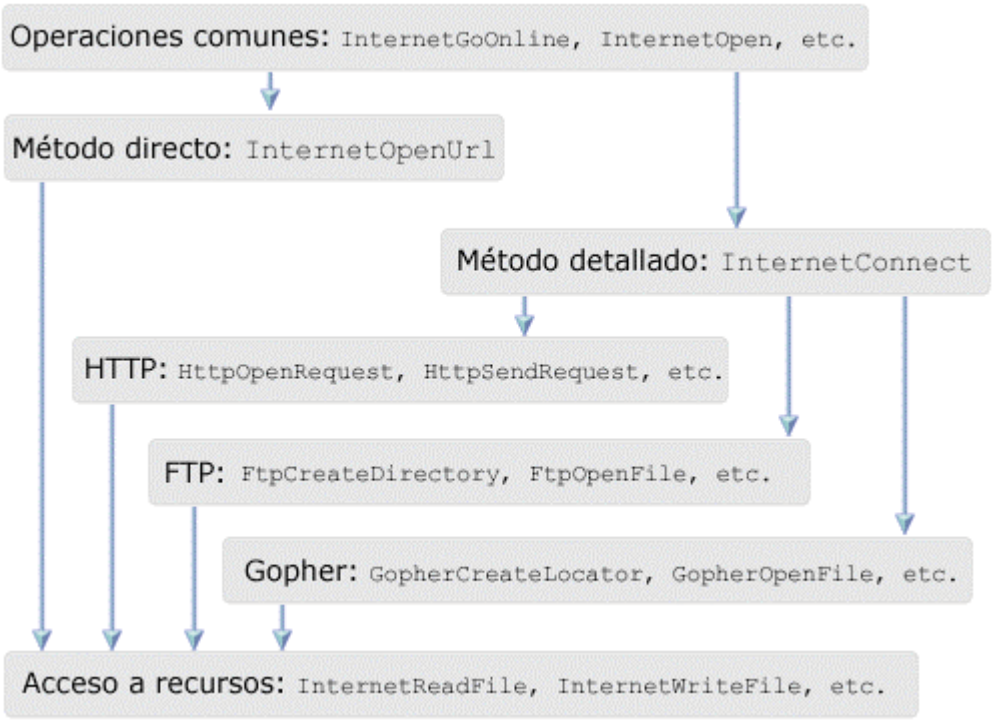
Bueno, creo que como introducción del protocolo ya es suficiente. Al menos nos sirve para darnos cuenta que para manejar un protocolo como HTTP es necesario un conocimiento bastante profundo de su funcionamiento interno. Para nuestra alegría, el API WinInet gestionará todos estos detalles internamente, haciéndonos la vida más fácil.

Primeros pasos dentro del API

El API Wininet nos permite enviar y recibir peticiones, a través de distintos protocolos, simplemente llamando a ciertas funciones, sin tener que preocuparnos de la estructura petición/respuesta que acabamos de ver.

Para ello, se cuenta con una serie de funciones para operaciones comunes, que se utilizarán siempre que vayamos a acceder al API, independientemente del protocolo a utilizar. Estas funciones comienzan por la palabra "Internet". Una vez realizadas estas operaciones, el programador tiene dos opciones: un método directo, que nos permite conectar con un recurso remoto, llamando simplemente a una función, o un método detallado, a través del cual tendremos que recorrer varios pasos hasta poder acceder al recurso. En este último caso, debemos decantarnos por uno de los protocolos disponibles, llamando unas funciones u otras dependiendo del protocolo elegido. Una vez hecho esto, lo último que queda por hacer es leer datos del recurso remoto, ya sean datos informativos (de cabecera) o los datos del recurso en sí (una página web, una imagen, etc).

En la siguiente figura podemos ver esta secuencia que acabamos de explicar:



En este artículo vamos a entrar en las partes comunes (operaciones comunes y lectura de datos) y en el método directo. Dejaremos para siguientes artículos el método detallado para HTTP y FTP.

Operaciones comunes

Antes de comenzar a enviar peticiones a través del API Wininet, es necesario realizar una serie de operaciones para preparar las estructuras internas. Estas operaciones son generales en el uso de WinInet, es decir: se tendrán que realizan tanto si queremos establecer una conexión HTTP, FTP o Gopher.

Comprobación de la conexión y/o marcado del módem

Antes de cualquier intento de conexión, es muy recomendable asegurarnos de que el equipo en el que estamos trabajando es capaz de conectarse a internet, aunque no es estrictamente necesario realizarlo en este momento. Para ello podemos hacer uso de las funciones de comprobación de la conexión.

Otra de las operaciones previas puede ser el marcado del módem, (a través de InternetAttemptConnect, InternetAutodial, etc.), siempre y cuando vayamos a utilizar este tipo de conexión.

Y por último, podemos hacer uso de las funciones InternetAttemptConnect e InternetGoOnline, que se encargará de hacer una comprobación de la bandera "Trabajar sin conexión", para informar al usuario y desactivarla cuando lo crea conveniente.

Todas estas funciones ya las explicamos en profundidad durante nuestro artículo anterior.

Apertura del API

El primer paso obligado dentro de Wininet es abrir una instancia del API. Esta apertura de instancia, lo que hace es crearnos un objeto que representará a la aplicación que hace uso de Wininet, identificada por un descriptor. Este objeto funcionará como descriptor raíz, a partir del cual iremos creando otros descriptores (por ejemplo: de conexión, de petición, etc.).

Dentro del API Wininet, cualquier descriptor que creemos será de tipo HINTERNET. Este tipo de dato nos obliga a que estos descriptores sean utilizados solamente con las funciones propias del API Wininet, por lo que no podremos utilizarlos como parámetros de las funciones básicas del API Win32, como CloseHandle, DuplicateHandle, etc. Ni que decir tiene que al contrario tampoco será posible: no podemos crear un descriptor con CreateFile y utilizarlo en una función de Wininet como InternetReadFile.

La función para la apertura del API es InternetOpen, y la situación más normal es que sea llamada una única vez por una aplicación, aunque si queremos definir distintos modos de comportamiento en la conexión (por ejemplo, acceso a través de distintos servidores proxy), debemos llamar a esta función una vez por cada tipo de comportamiento que queramos utilizar.

Vamos a ver cómo se llama a la función InternetOpen:

```
function InternetOpen(
            lpszAplicación:  PChar;
            dwTipoAcceso:    LongWord;
            lpszProxy:       PChar;
            lpszProxyBypass: PChar;
            dwFlags:         LongWord
        ): HINTERNET;
```

Los parámetros son los siguientes:

- **lpszAplicación**: un puntero a cadena que contiene el nombre de la aplicación (en inglés: user-agent) que va a enviar las peticiones. Este nombre será el utilizado en la llamada HTTP que haga el cliente, concretamente en la clave "User-Agent" de la cabecera, y sirve para identificar a un cliente frente al servidor que reciba las peticiones HTTP. Estableciendo un "agente" especial, podemos hacer que nuestro servidor HTTP responda de un modo diferente a ciertas llamadas, por ejemplo, las que llegan desde una aplicación concreta. Un ejemplo práctico de esto son las estadísticas sobre el tipo de navegador o sistema operativo que utilizamos los usuarios a la hora de navegar por internet. Es sencillo imaginarse que cada navegador (que no es más que un cliente HTTP) utilizará un "user-agent" distinto, por lo que un servidor web sólo tendrá que registrar el número de accesos de cada tipo, y calcular estadísticas a partir de este dato. A modo de curiosidad: podemos ver fácilmente el "user-agent" que envía nuestro navegador, utilizando JavaScript, en la siguiente url:
  http://developer.apple.com/internet/javascript/internetdev-sniffer.html

- **dwTipoAcceso**: indica el tipo de acceso que se va utilizar en la conexión:

  - INTERNET_OPEN_TYPE_DIRECT: hace una conexión directa a internet, sin utilizar servidores proxy ni corta-fuegos entre nuestro equipo y los recursos remotos. Esta configuración es válida para conexiones permanentes, como ADSL, cable, etc.

  - INTERNET_OPEN_TYPE_PRECONFIG: lee del registro el tipo de conexión que hayamos configurado en "Panel de Control - Opciones de Internet - Conexiones - Configuración LAN". Este modo es el más recomendable para asegurarnos de que vamos a utilizar el tipo de conexión que esté utilizando el usuario desde Internet Explorer.

  - INTERNET_OPEN_TYPE_PROXY: se utiliza un proxy para realizar la conexión, a no ser que la conexión se haga a un host que se encuentre en la lista de "excepciones" (en el parámetro lpszExcepciónProxy).

- **lpszProxy**: una cadena que contiene el nombre de el(los) servidor(es) proxy a utilizar en la conexión. Si en el parámetro dwTipoAcceso se ha indicado un valor distinto de INTERNET_OPEN_TYPE_PROXY, este parámetro debe ser nulo (nunca una cadena vacía). En caso contrario, debe pasarse una cadena que puede indicar símplemente un nombre de host (o dirección IP) que se utilizará como proxy, de este modo:
  "servidor_proxy" ó "127.100.100.1"
  Aunque con el nombre de host es suficiente, esta función permite una sintaxis más compleja, definiendo un proxy para cada uno de los protocolos soportados por Wininet. La sintaxis es la siguiente:
  "[prot=][prot://]host[:puerto]"
  Las opciones encerradas entre corchetes indican que son opcionales. "prot" indica el tipo de protocolo para el que estamos definiendo un proxy, es decir: si queremos especificar un protocolo especial para una conexión por FTP, debemos pasar la siguiente cadena:
  "ftp=ftp://servidor_proxy"
  "host" indica el nombre o dirección IP del servidor que se utilizará como proxy. "puerto" indica el número de puerto al que se accederá. Si se omite, se utilizará el puerto por defecto para cada uno de los protocolos (21 para FTP, 70 para Gopher, 80 para HTTP y 443 para HTTPS).
  Para definir otros servidores proxy, basta con separar las distintas cadenas por un espacio, por ejemplo:
  "ftp=ftp://servidor_proxy_1:21 http=http://10.10.10.1:80"
  De este modo se indica que se utilice el "servidor_proxy_1" para conexiones FTP y el

"10.10.10.1" para conexiones HTTP.
También es posible indicar un servidor proxy para un protocolo en particular, y otro servidor para el resto de protocolos, del siguiente modo:
"ftp=ftp://servidor_proxy_1:21 servidor_el_resto"

- **lpszExcepciónProxy**: este parámetro sirve para indicar una serie de direcciones que serán accedidas directamente, sin utilizar el proxy. Normalmente, esta lista es configurable por el usuario, a través de la opción "Panel de control - Opciones de Internet - Conexiones - Configuración LAN - Avanzadas - Excepciones", aunque utilizando el API Wininet podemos definir nuestra propia lista de excepciones. La cadena de excepciones también sigue una sintaxis especial, pudiendo utilizar el carácter comodín "*", o incluír varios servidores separándolos por un espacio. Existe una cadena especial, , que indica que todas las direcciones que no contengan un punto (locales), se accederán sin utilizar un proxy.
Por ejemplo, podemos indicar una cadena como la siguiente, separando las distintas excepciones por espacio:
" 127.*.100.* 128.*.*.* 192.1.45.12"

- **dwOpciones**: un valor que combina las opciones de comportamiento. Se puede incluír cualquiera de los siguientes valores:

    - INTERNET_FLAG_ASYNC: indica que las llamadas se harán de forma asíncrona, esto es, cada función que llamemos retornará antes de que termine su ejecución. Posteriormente, cuando la ejecución haya terminado, se notificará al programador a través de una función de retrollamada (callback). Esta bandera afecta a las llamadas hechas a través del descriptor retornado por InternetOpen, como las hechas con cualquier descriptor descendiente de este. Trataremos el tema de las funciones asíncronas en el próximo artículo.

    - INTERNET_FLAG_FROM_CACHE: cuando se tenga que recuperar algún dato de la red, se intentará buscar esta información en el caché que mantiene Internet Explorer, y si existe y es correcto, lo retornará. En caso de que el caché no contenga esta información, la función correspondiente retornará un error, normalmente el valor de la constante ERROR_FILE_NOT_FOUND. Esta bandera debe establecerse cuando una aplicación funcione en modo "Trabajar sin conexión".

    - INTERNET_FLAG_OFFLINE: funciona del mismo modo que el modo anterior, INTERNET_FLAG_FROM_CACHE, intentarndo resolver todas las peticiones contra el caché, sin hacer ningún acceso remoto.

Esta función retornará **nil** si ha ocurrido algún error o un descriptor válido si todo ha ido bien. Debemos almacenar en una variable el descriptor retornado ya que lo necesitaremos para crear el resto de descriptores. En caso de error, se puede llamar a la función GetLastError para averiguar el código de error.

Método directo

Una vez abierta una instancia en el API, el modo más sencillo de acceder a los datos remotos es a través de la llamada a una sola función: InternetOpenUrl.

Esta función, se encargará se decidir el tipo de protocolo a utilizar (dependiendo de la URL que le pasemos) y darnos acceso a un recurso remoto. Por esta razón, podemos utilizar esta función con cualquier protocolo de los soportados por Wininet, y no solamente HTTP.

InternetOpenUrl está pensada para aquellos casos en que no es necesario utilizar alguna característica especial del protocolo (como crear carpetas con FTP o enviar un formulario con HTTP), sino que símplemente nos basta con acceder a los datos del recurso.

Internamente, esta función hace distintas operaciones, que veremos desglosadas cuando hablemos sobre el método detallado.
La sintaxis es la siguiente:

```
function InternetOpenUrl(
            hInternet:     HINTERNET;
            lpszUrl:       PChar;
            lpszCabeceras: PChar;
            dwLongitudCabeceras LongWord;
            dwOpciones:    LongWord;
            dwContexto:    LongWord
        ): HINTERNET;
```

- **hInternet**: un descriptor que hemos obtenido a través de la llamada a InternetOpen.
- **lpszUrl**: un puntero a cadena que contiene la URL a la que queremos acceder. Esta URL debe contener obligatoriamente los componentes: protocolo, servidor y recurso, aunque opcionalmente podemos incluír también el puerto, usuario y contraseña, y parámetros adicionales. Un buen método de crear esta URL es a través de la función InternetCreateUrl, explicada en nuestro anterior artículo. Dado que Wininet sólo soporta 3 protocolos, esta URL sólo puede comenzar por gopher://, ftp://, http:// y https://
- **lpszCabeceras**: es un puntero a una cadena con la que podemos enviar cabeceras adicionales en la petición HTTP. Por ejemplo, podríamos crear nuestra propia cabecera y enviarla desde un cliente que utilice WinInet, para que, al otro lado de la línea, el servidor HTTP interprete esta cabecera de un modo especial. En este caso, nos estaríamos saliendo del estándar definido por HTTP. En caso de no necesitar cabeceras adicionales (lo más habitual para un uso normal del protocolo), debemos pasar el valor **nil**.
- **dwLongitudCabeceras**: el número de caracteres de las cabeceras adicionales pasadas en lpszCabeceras. Si pasamos el valor -1, y lpszCabeceras es distinto de **nil**, la función calculará la longitud correcta (buscando el carácter nulo final). Este parámetro se ignorará si hemos pasado el valor **nil** en lpszCabeceras.
- **dwOpciones**: en este parámetro podemos configurar el comportamiento. A continuación explico algunos de los valores posibles, aunque sólo incluyo aquellos que tienen alguna utilidad en el protocolo HTTP. Cuando hablemos del protocolo FTP daremos un repaso a esta función, explicando el resto de los valores posibles:
    - INTERNET_FLAG_HYPERLINK: obliga a descargar el recurso cuando el servidor no haya retornado valores de caducidad o última modificación (claves "Expires" y "Last-modified" de las cabeceras del protocolo).

  - INTERNET_FLAG_NO_AUTO_REDIRECT: Wininet, por defecto, gestiona las redirecciones de forma transparente. De este modo, si el recurso al que queremos acceder nos redirecciona a otro, en realidad accederemos al segundo. Incluyendo este valor, obligamos a que Wininet no gestione estas redirecciones, dejando la situación en manos del programador.

  - INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTP: Cancela las redirecciones desde una conexión segura (HTTPS) a una conexión normal (HTTP).

  - INTERNET_FLAG_IGNORE_REDIRECT_TO_HTTPS: este parámetro es igual que el anterior, pero en sentido contrario, es decir: una redirección de una conexión normal a una conexión segura.

  - INTERNET_FLAG_NEED_FILE: si el recurso no puede ser almacenado en el caché, se guardará en un archivo temporal.

  - INTERNET_FLAG_NO_CACHE_WRITE: no almacena el recurso en el caché de datos.

  - INTERNET_FLAG_NO_COOKIES: ignora todas las peticiones de cookies que haga el recurso al que estamos accediendo, ya sean de lectura o grabación.

  - INTERNET_FLAG_NO_UI: Cuando se intenta crear una cookie, y hemos configurado Internet Explorer para pedir confirmación (según el nivel de seguridad establecido en Panel de control - Opciones de Internet - Seguridad), se muestra al usuario una ventana de diálogo pidiendo confirmación. Si incluímos este valor, se ignorará esta ventana.

  - INTERNET_FLAG_PRAGMA_NOCACHE: obliga a descargar el recurso del servidor original, incluso si existe una copia en el caché de algún servidor proxy intermedio. A propósito de esto: podríamos utilizar esta bandera para evitar uso del proxy-caché que ha instalado Telefónica a todos sus clientes de ADSL, aunque mucho me temo que Internet Explorer no hace.

  - INTERNET_FLAG_RELOAD: fuerza a descargar el recurso del servidor original, aunque exista una copia en nuestro caché local.

  - INTERNET_FLAG_RESYNCHRONIZE: obliga a volver a sincronizar el contenido del caché local con los datos del servidor original.

  - INTERNET_FLAG_SECURE: establece una conexión segura, utilizando la extensión SSL (Secure Socket Layer). Para utilizar una conexión segura, debemos tener correctamente instalada la extensión para canales seguros, incluída en la librería "schannel.dll". La versión más sencilla de esta librería se incluye a partir de Windows 95 ORS2.

- **dwContexto**: este valor se utilizará para pasarlo a la función de "callback" en las conexiones asíncronas (pasando INTERNET_FLAG_ASYNC en el parámetro dwOpciones de la función InternetOpen). Normalmente podremos pasar un puntero a una estructura donde almacenamos datos sobre la conexión, el estado de la descarga, etc. Para conexiones síncronas (por defecto), este parámetro se ignorará, por lo que debemos pasar el valor 0.

Esta función retornará un descriptor válido si la conexión ha sido establecida o **nil** si ha ocurrido un error (en este caso podemos utilizar GetLastError para obtener el código de error).

Si hemos conseguido acceder correctamente a la Url, el siguiente paso es leer directamente de los recursos remotos, utilizando las funciones que explicamos a continuación.

Lectura de datos

Una vez que hemos enviado una petición a través de la red, ya sea con el método directo o detallado, será procesada por el servidor HTTP que devolverá una respuesta al cliente. Esta respuesta, como ya hemos visto, se compone de una cabecera y los datos que hemos pedido en la petición. A través de Wininet, podemos acceder tanto a los datos de cabecera (que nos proporcionan información sobre el recurso), como al recurso en sí.

Lectura de datos de cabecera

Wininet mantiene internamente la estructura de la cabecera retornada, y nos permite acceder a ella a través de una función: HttpQueryInfo.

Esta función nos facilita la lectura de datos de la cabecera, indicando el atributo del cual queremos obtener el dato.

La sintaxis es la siguiente:

```
function HttpQueryInfo(
            hPetición: HINTERNET;
            dwInformación: LongWord;
            lpBuffer: Pointer;
            var lpdwLongitudBuffer: LongWord;
            var lpdwIndice: LongWord
         ): LongBool;
```

- **hPetición**: un descriptor de petición, obtenido a través de la función InternetOpenUrl (o HttpOpenRequest, utilizando el método detallado, aunque no lo hayamos explicado).
- **dwInformación**: es un valor combinado entre el atributo a leer y los modificadores que podemos aplicar. A continuación doy una lista de los atributos más importantes que podemos leer, aunque se puede consultar una lista más completa en web del MSDN de Microsoft, en la siguiente url:
  http://msdn.microsoft.com/library/en-us/wininet/wininet/query_info_flags.asp.
  La mayoría de estas contantes corresponden con un atributo de la respuesta del protocolo HTTP, como Last-Modified, Content-Type, etc.
  - HTTP_QUERY_STATUS_CODE: el código de retorno devuelto por el servidor.
  - HTTP_QUERY_STATUS_TEXT: descripción del estado devuelto por el servidor.
  - HTTP_QUERY_DATE: fecha y hora en que se originó la petición.
  - HTTP_QUERY_SERVER: información acerca del servidor HTTP que ha generado esta respuesta.
  - HTTP_QUERY_MIME_VERSION: la versión de los tipos MIME utilizados en la respuesta.
  - HTTP_QUERY_CONTENT_TYPE: el tipo MIME del recurso leído.
  - HTTP_QUERY_CONTENT_LENGTH: el tamaño, en bytes, del recurso leído.
  - HTTP_QUERY_LAST_MODIFIED: fecha y hora de la última modificación el recurso.

- HTTP_QUERY_CUSTOM: nos permite buscar un atributo dentro de las cabeceras adicionales pasadas durante la llamada a InternetOpenUrl o HttpSendRequest. El nombre de la cabecera que queremos buscar debemos pasarlo a través del parámetro lpBuffer.
- HTTP_QUERY_EXPIRES: la fecha y hora a partir de la cual se considera el recurso como caducado.
- HTTP_QUERY_ALLOW: consulta las operaciones soportadas por el servidor.
- HTTP_QUERY_REQUEST_METHOD: el tipo de operación que se ha utilizado en la petición.
- HTTP_QUERY_ACCEPT: los tipos de datos MIME que acepta el servidor.
- HTTP_QUERY_RAW_HEADERS: retorna toda la cabecera recibida. Cada atributo está terminado por el carácter 0 y la secuencia final está terminada por un doble carácter 0, del siguiente modo: "attr: valor\0attr: valor\0attr: valor\0\0"
- HTTP_QUERY_RAW_HEADERS_CRLF: igual que el anterior, pero se separa cada cabecera por un retorno de carro, del siguiente modo: "attr: valor\nattr: valor\nattr: valor\0"

Además de los atributos posibles, se puede combinar con alguno de los siguientes modificadores:

- HTTP_QUERY_FLAG_NUMBER: retorna el atributo pedido como si fuera un número de 32 bits (por ejemplo en el atributo Content-Length).
- HTTP_QUERY_FLAG_SYSTEMTIME: retorna el atributo pedido como una estructura de tipo SYSTEMTIME. Se puede ver una descripción de la estructura SYSTEMTIME en la siguiente url:
  http://msdn.microsoft.com/library/en-us/sysinfo/base/systemtime_str.asp
  Esta bandera puede utilizarse para retornar cualquier atributo de tipo fecha-hora como Last-Modified, Expires, etc.

- **lpBuffer**: un puntero a un espacio de memoria donde se almacenará el atributo que hemos pedido a través del parámetro dwInformación. El atributo se retornará en formato cadena, a no ser que se haya utilizado alguna bandera de modificación, en cuyo caso se retornará en el formato indicado (número o estructura SYSTEMTIME).
- **lpdwLongitudBuffer**: un puntero a un número de 32 bits que almacena el tamaño del buffer pasado en lpBuffer. Cuando la función retorna, almacena en este parámetro el tamaño en bytes del valor retornado (incluido el carácter nulo final para cadenas) o bien el tamaño en bytes requerido si la función ha retornado FALSE y GetLastError nos ha devuelto el código de error ERROR_INSUFFICIENT_BUFFER.
- **lpdwIndice**: cuando se retorna un atributo con varios valores (como HTTP_QUERY_ACCEPT), este valor indica el índice (comenzando en 0) del valor a obtener. Al retornar, la función almacena en este parámetro el índice del siguiente valor, o el valor de la contstante ERROR_HTTP_HEADER_NOT_FOUND si ya no hay más valores. Si se va a consultar un atributo sencillo, se debe pasar el valor 0.

La función retornará TRUE o FALSE dependiendo del éxito o fracaso. En caso de error, debemos llamar a GetLastError para averiguar las razones del fracaso (retornará, por ejemplo el valor ERROR_INSUFFICIENT_BUFFER si el tamaño de la memoria en lpBuffer es demasiado pequeño).

Disponibilidad de datos

Los servidores web comerciales (como Apache, Internet Information Server, etc.) son programas muy completos que permiten gestionar todos los aspectos involucrados en las peticiones HTTP. Uno de los puntos más críticos es la disponibilidad de datos, ya que en un servidor web, lo más normal es que multitud de usuarios estén accediendo a los mismos recursos, y muy posiblemente en el mismo momento.

Para optimizar al máximo estas operaciones, los servidores web cuentan con un caché en el que van situando los datos que pueden ser descargados posteriormente. Por ejemplo, si el servidor recibe una petición HEAD (para recuperar los datos de cabecera de un archivo), lo más probable es que posteriormente se acceda al recurso, así que lo prepara para comenzar a leerse. Del mismo modo, si comenzamos a leer datos de un recurso, el servidor web intentará mantener en el caché aquellos datos que todavía no hemos leído, para que estén disponibles en el momento en que los necesitemos.

Desde el API Wininet podemos consultar qué datos están disponibles para que los leamos desde nuestro cliente HTTP, o bien forzar al servidor a que disponga un bloque de datos para que esté listo para ser leído.

Todo ello lo podemos hacer a través de InternetQueryDataAvailable. Esta función nos retorna el número de bytes disponibles para que leamos en una operación posterior.

Si el servidor no tiene ningún byte disponible, la función fuerza a que se prepare un bloque de datos, y no retornará hasta que esto haya ocurrido.

Un comportamiento especial de esta función se da cuando el recurso se encuentra en el caché y hemos permitido que se lea de él. La función nos retornará siempre el tamaño completo del recurso, ya que al estar en un archivo local, tendremos disponibles todos los datos. Para este caso, podremos leer el recurso con una sola operación e lectura.

La sintaxis es la siguiente:

```
function InternetQueryDataAvailable(
             hPetición: HINTERNET;
             var lpdwBytesDisponibles: LongWord;
             dwOpciones: LongWord;
             dwContexto: LongWord
         ): LongBool;
```

- **hPetición**: un descriptor de petición, obtenido a través de la función InternetOpenUrl (o HttpOpenRequest, utilizando el método detallado, aunque no lo hayamos explicado).
- **lpdwBytesDisponibles**: un puntero a un valor de 32 bits en el que se almacenará el número de bytes que tiene el servidor disponibles para la lectura.
- **dwOpciones**: actualmente no hay ninguna opción, así que este valor debe ser 0.
- **dwContexto**: también debe ser 0.

La función retorna TRUE o FALSE dependiendo del éxito. En caso de error, se puede llamar a la función GetLastError para averiguar el código de error. Si este código es ERROR_NO_MORE_FILES, significará que no se han podido preparar los datos ya que el recurso pedido no existe.

Lectura de datos del recurso

Cuando accedemos a un recurso en internet, lo realmente importante para nosotros es la información que nos proporciona ese recurso, ya sea una página web, un archivo de texto, una imagen o un video, etc.

Para ello, contamos con algunas funciones, muy parecidas a las de lectura/escritura de ficheros, para el acceso a recursos a través de internet.

Utilizando el método directo (el único que conocemos por ahora), solo podemos realizar lecturas de archivos y nunca modificar un dato a través del protocolo HTTP. En el próximo artículo veremos cómo se utiliza el método detallado y cómo utilizar las funciones de escritura de ficheros.

Para leer un recurso remoto, debemos hacer uso de la función InternetReadFile. Esta función se comporta igual que cualquier función de lectura de ficheros, es decir: de un modo secuencial. Una vez leídos los primeros 10 bytes del fichero, la siguiente operación de lectura comenzará a partir del byte 11, así hasta llegar al final.

La sintaxis es la siguiente:

```
function InternetReadFile(
            hPetición: HINTERNET;
            lpBuffer: Pointer;
            dwTamañoLeer: LongWord;
            var lpdwTamañoLeído: LongWord
          ): LongBool;
```

- **hPetición**: un descriptor de petición, obtenido a través de la función InternetOpenUrl (o HttpOpenRequest, utilizando el método detallado, aunque no lo hayamos explicado). Esta petición debe ser obligatoriamente de tipo GET.
- **lpBuffer**: es un puntero a una zona de memoria donde se copiarán los bytes leídos. El espacio disponible de este buffer debe ser, al menos, de "dwTamañoLeer" bytes.
- **dwTamañoLeer**: número de bytes que debe leer la función, como máximo, el tamaño del bloque de memoria pasado en lpBuffer. Este valor se debe obtener a través de la función InternetQueryDataAvailable, como ya hemos explicado anteriormente, y el bloque de memoria "lpBuffer" debe reservarse también acorde a este valor.
- **dwTamañoLeído**: un puntero a una variable (que se pasará a 0) donde se copiará el número de bytes leídos. Normalmente, debemos leer dentro de un bucle hasta que la función copie el valor 0 en este parámetro, para indicar que ya no hay más que leer.

La función retorna TRUE o FALSE. En caso de error, se puede utilizar la función GetLastError para retornar el código de error. Si GetLastError retorna ERROR_INTERNET_EXTENDED_ERROR, debemos llamar a la función InternetGetLastResponseInfo, para que nos retorne el código y la descripción del último mensaje de error recibido desde el servidor. Esta función tiene la siguiente sintaxis:

```
function InternetGetLastResponseInfo(
            var lpdwError: LongWord;
            lpszDescripción: PChar;
            var lpdwLongitudDescripción: LongWord
          ): LongBool;
```

El uso es sencillo: basta con pasar un puntero a un valor de 32 bits, donde se copiará el código de error, y un puntero a una cadena, donde se copiará la descripción del error.

A partir de IE 4.0, Wininet proporciona una función extendida llamada InternetReadFileEx. El principal uso que se le da es leer el recurso completo, junto con las cabeceras, en una sola llamada a la función.

Como hemos dicho antes, la lectura de fichero de internet es muy parecida a los ficheros locales, ya que internamente se mantiene un puntero que nos indica donde se debe realizar la próxima lectura.

Este puntero, es desplazable por el programación a través de la función InternetSetFilePointer.

La sintaxis es la siguiente:

```
function InternetSetFilePointer(
              hRecurso: HINTERNET;
              lDesplazamiento: LongInt;
              pReservado: Pointer;
              dwInicio: LongWord;
              dwContexto: LongWord
          ): LongWord;
```

- **hPetición**: un descriptor de petición, obtenido a través de la función InternetOpenUrl (o HttpOpenRequest utilizando el método detallado, aunque no lo hayamos explicado). Esta petición debe ser de tipo GET. Además, para utilizar esta función, el descriptor no ha podido crearse con INTERNET_FLAG_DONT_CACHE o INTERNET_FLAG_NO_CACHE_WRITE.
- **lpDesplazamiento**: número de bytes a desplazar el puntero. Se pueden utilizar tanto valores positivos como negativos.
- **lpReservado**: debe ser **nil**.
- **dwInicio**: define el origen del movimiento, a partir del cual se desplazará el puntero. Se puede indicar uno de los siguientes valores:
    - FILE_BEGIN: se desplaza "lDesplazamiento" bytes desde el inicio del archivo.
    - FILE_CURRENT: se desplaza "lDesplazamiento" bytes desde la posición actual.
    - FILE_END: se desplaza "lDesplazamiento" bytes desde el final del archivo. Este método fallará si el servidor no es capaz de averiguar la longitud del archivo.
- **dwContexto**: está reservado para el futuro. Debe pasarse siempre 0.

La función retorna la posición actual, si tiene éxito, o -1 si ha ocurrido un error. Hay que tener en cuenta que la función no puede usarse si hemos alcanzado el final del fichero con repetidas lecturas a través de InternetReadFile.

Terminando con todo...

Y como todos os podréis imaginar, falta la operación obligada: el cierre de descriptores y liberación de memoria.

Cuando hablamos de los objetos del núcleo, dentro del artículo sobre archivos proyectados en memoria (www.lawebdejm.com/?id=22141), dijimos que cualquier descriptor de objeto del núcleo se cerraba utilizando la misma función: CloseHandle. En nuestro caso, con Wininet, nos ocurre lo mismo: cualquier descriptor creado dentro del API Wininet, los de tipo HINTERNET, se cierra con la misma función: InternetCloseHandle.

La sintaxis es muy sencilla:

```
function InternetCloseHandle(
            hInet: HINTERNET
        ): LongBool;
```

Creo que en este caso no hay mucho que explicar. Simplemente debemos pasar el descriptor a cerrar y la función nos retornará un valor booleano indicando si la operación ha tenido éxito.

Es muy conveniente que, en caso de haber conseguido cerrar el descriptor correctamente, le asignemos el valor **nil**, para si posteriormente si hace alguna comprobación, seamos capaces de saber que este descriptor ya ha sido cerrado.

Con esto, ya tenemos todos los datos para completar nuestra función, y así, poder hacer las llamadas a las funciones básicas del protocolo HTTP, dentro de Wininet.

Conclusión

Y esto es todo, por ahora. Hemos podido ver cómo utilizar el protocolo HTTP desde el API Wininet, aunque sólo a través del método directo.

En el siguiente artículo trataremos el método detallado, con el que tendremos un mayor control sobre la conexión al servidor y las peticiones que le enviamos. Además, trataremos el uso de Wininet de forma asíncrona.

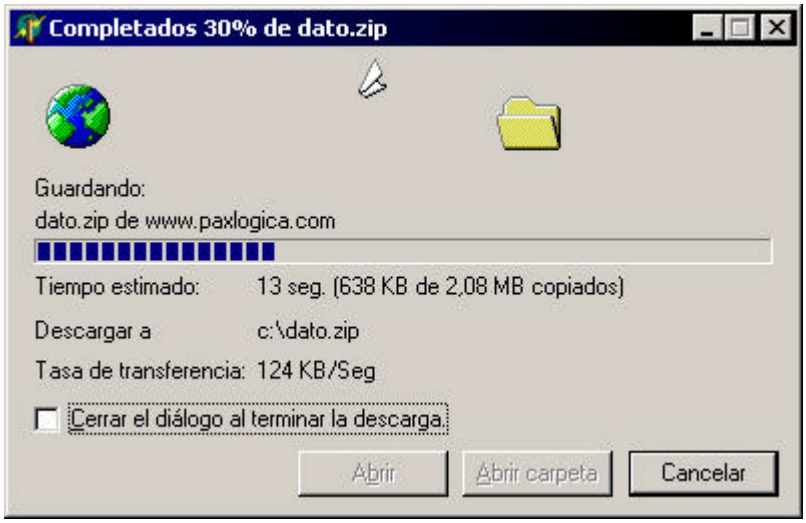
Espero que este artículo os haya resultado útil y ya sabéis que cualquier duda o sugerencia, desde el Grupo Albor estaremos encantados de recibirla.

Los ejemplos

Como el tema puede resultar algo laborioso (aunque no complicado), he incluído algún ejemplo para aquellos que queráis estudiar el código.

Delphi 5

Una serie de unidades que implementan un sistema de descarga "listo para usar". La clase THiloDescarga implementa un descendiente de TThread para realizar una descarga utilizando el método directo. Se puede utilizar por sí sola, aunque está pensada para ser utilizada desde la clase THttpFile. La clase TProgresoFrm es una ventana que imita la que utiliza Internet Explorer 5 para mostrar el progreso de una descarga.



Y por último, THttpFile nos permite definir una URL y descargarla directamente a una carpeta, ya sea con interfaz gráfica o sin ella. Esta clase hereda de TComponent, por lo que podréis registrarla en el sistema y utilizarla como otro componente más.

Fuentes en ZIP - www.lawebdejm.com/?id=22221

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