Los rincones del API Win32
Introducción al API WinInet

Comenzamos una nueva serie sobre Win32 y en esta ocasión trataremos, durante los próximos artículos, el acceso a "la red de redes" a través de WinInet.

De qué vamos a hablar

Durante los cuatro últimos artículos hemos ido tratando un tema muy básico dentro de cualquier arquitectura: la memoria. Hemos profundizado en aspectos internos, sobre cómo gestionarla en esta plataforma, así como las funciones más útiles en cada una de las tareas.

En esta ocasión vamos a "poner los pies en la tierra", y durante los próximos artículos vamos a tratar un aspecto con aplicaciones mucho más inmediatas, y no tan teórico como han podido ser los anteriores artículos: el acceso a recursos en internet.

No vamos a tratar los conceptos más internos de comunicaciones en internet, ni de los protocolos de transporte TCP/IP, ni siquiera vamos a tratar la programación de sockets, sino que nuestro objetivo se centra en un API muy concreto, desarrollado por Microsoft allá cuando salió Internet Explorer 3.0: el WinInet.

Todos estamos viendo como se está desarrollando la industria del software, y podemos percibir como cada vez toma más importancia en una aplicación el acceso a internet, la recuperación de datos y contenidos de la red y la información on-line. Ya pasaron los tiempos de aplicaciones cliente-servidor normales, de bases de datos en redes locales, de aplicaciones de escritorio simples, etc. Ahora lo más *"in"* son las aplicaciones que están constantemente conectadas a la red para recuperar información. Para cumplir este objetivo tenemos muchas alternativas: desde los nuevos protocolos de intercambio de ficheros *peer-to-peer* (como Napster, eDonkey, eMule, etc.), hasta ajustarse a protocolos más estándar como pueden ser los archiconocidos HTTP y FTP.

Esta serie de artículos que comenzamos se limita a utilizar protocolos y modos de acceso estándar, a través de un API desarrollado por Microsoft para estas tareas. Posiblemente, este modo de acceso sea el más adecuado para la mayoría de nuestras necesidades, en que sólo necesitamos acceder a contenidos en modo texto o binario. Para que os hagáis una idea de la importancia de esta tecnología, Internet Explorer, el componente TWebBrowser e incluso el propio Explorador de Windows, utilizan Wininet para acceder a los contenidos en red. De hecho, los planes de Microsoft en esta línea son muy claros: integrar los accesos a internet dentro del núcleo de Win32, de modo que el acceso a un recurso local se diferencie lo más mínimo del acceso a recursos en red (todo ello, en la plataforma *.NET*, por supuesto).

Introducción a WinInet

Básicamente, WinInet es un *conjunto de funciones de alto nivel, para al acceso a contenidos en internet*.

Estas funciones, están disponibles dentro de la librería "wininet.dll", que podréis encontrar en la carpeta de sistema. Esta librería se distribuye desde Windows 95, o con cualquier instalación del Internet Explorer a partir de la versión 3.0, así que no tendremos problemas en este aspecto. De todas formas, ciertas funciones sólo están disponibles a partir de la versión 4 del explorador de Microsoft, y otras más evolucionadas a partir de la versión 5.

La principal ventaja de su uso es que se nos ocultan los detalles de implementación de los distintos protocolos que podemos utilizar. De este modo, no necesitamos conocer la estructura de peticiones y respuestas de cada protocolo, sino que será la propia librería la encargada de convertir nuestras llamadas en peticiones en el lenguaje del protocolo adecuado.

Además de esto, cuando cambien los detalles internos del protocolo (por ejemplo si se establece una nueva versión), nuestras aplicaciones seguirán funcionando, ya que será la librería la encargada de modificar sus llamadas.

Otra de las ventajas que ofrece Wininet es que nos abstrae sobre el tipo de conexión que estamos realizando. Es decir, para acceder a los contenidos actuaremos de una forma similar estemos en una LAN o en internet, ya sea a través de proxy, conexión por modem o cualquier otro tipo. Además, podemos aprovecharnos de la configuración que haya establecido el usuario en las opciones de conexión a internet (en el Panel de control - Opciones de internet - Conexiones).

Por si fuera poco, con el API Wininet se nos permite acceder a programas y scripts desarrollados en el lado del servidor, ya sean CGI, ISAPI, ASP, PHP, J2EE, etc. Por ejemplo, supongamos que ya tenemos desarrolladas una serie de funciones, en PHP, para la validación de usuarios, y estas funciones están siendo usadas desde el sistema de *login* de nuestra página web. Desde nuestras aplicaciones de escritorio, podremos beneficiarnos de estos desarrollos, accediendo a través de Wininet a los *scripts* que ya están funcionando en el servidor. En una posible modificación de estos *scripts* del servidor, será beneficiado tanto el acceso desde la página web como desde la aplicación de escritorio.

Además, también nos permite gestionar las *cookies* del sistema, añadiendo, eliminando o manipulando nuestras propias *galletas*.

Y por último, una característica muy interesante: Wininet nos permite acceder de forma transparente al caché de datos que gestiona Internet Explorer. De este modo, si cierta página o archivo ha sido descargada previamente, desde nuestras aplicaciones podemos beneficiarnos de este caché, y recuperar los datos del disco duro en vez de acceder a internet.

Protocolos

Wininet, en su versión actual, soporta el acceso a contenidos en internet a través de tres protocolos: HTTP, FTP y Gopher.

No vamos a profundizar en los detalles de cada uno de ellos, porque no es nuestro objetivo, aunque sí vamos a dar una pequeña descripción de su principal objetivo:

- **HTTP** (Hyper Text Transfer Protocol): a todo el mundo le sonará este protocolo, el más utilizado de internet. Su principal uso es el envío y recepción de información en formato texto, como pueden ser páginas HTML. Además, Wininet soporta la extensión HTTPS, para conexiones seguras.

- **FTP** (File Transfer Protocol): este protocolo también es bastante utilizado, y está orientado a la transmisión de archivos binarios de gran tamaño, como pueden ser ejecutables, archivos comprimidos, imágenes, etc.

- **Gopher** (Go-for...): Este protocolo actualmente está en desuso, aunque en los inicios de internet tuvo un papel muy importante. Su principal función fue la de catalogar y buscar contenidos a través de la red, almacenando índices con todos los recursos disponibles. Hoy en día, gracias a motores de búsqueda basados en HTTP, como Google o Yahoo, este servicio ya no es necesario.

En esta nueva serie de artículos, vamos a profundizar en los dos primeros protocolos, HTTP y FTP, ya que no creo que el protocolo Gopher vaya a ser de mucha utilidad para alguno de vosotros (para mi no lo es).

Esquema general

En este artículo y los sucesivos, vamos a intentar bucear dentro del API Wininet, dejando a un lado las funciones propias del protocolo Gopher.

Básicamente, lo que vamos a explicar es lo siguiente:

1. Introducción a la tecnología y funciones generales:
   Daremos detalles sobre la implementación de Wininet, así como de las funciones generales del API, para conexión, cierre de descriptores, manipulación de *cookies*, etc.

2. Funciones para el protocolo HTTP:
   Entraremos en detalles dentro el protocolo HTTP, explicando sus funciones y usos más típicos. Además, desarrollaremos un pequeño cliente de internet a través de HTTP.

3. Funciones para el protocolo FTP:
   Entraremos en detalles dentro de este protocolo, y desarrollaremos un pequeño cliente FTP, para manipulación de archivos en remoto.

4. Otros aspectos:
   Cubriremos el resto de aspectos, como gestión de caché, llamadas asíncronas, etc.

Esto es todo lo que vamos a tratar durante esta parte de *"Los rincones del API Win32"*.

La tecnología Wininet

Wininet nació con la idea de abstraer al programador de la implementación de los protocolos de aplicación más utilizados en internet. Gracias a esta idea, podemos acceder a internet sin saber, ni siquiera, qué es un protocolo o cómo implementarlo. Para ello, se ha creado como una capa de abstracción sobre el API Winsock.

Winsock es la tecnología utilizada por Win32 para comunicar máquinas a través de los protocolos de transporte TCP/IP y UPD. Estos protocolos sólo se encargan del nivel de transporte en el modelo OSI. Por encima de esto, se encuentran los protocolos de aplicación, que son los que nos ocupan en nuestro caso: HTTP, FTP, SMTP, NNTP, etc.

Antes de la aparición del API Wininet, era obligatorio utilizar Winsock para acceder a algún recurso en internet. Eso requería un entendimiento bastante profundo del protocolo a utilizar, además de conceptos como *socket*, *puerto*, etc. Actualmente, sólo será necesario utilizar Winsock para hacer uso de protocolos no soportados por Wininet, como puede ser envío y recepción de correos electrónicos (protocolos SMTP y POP3), acceso a servidores de noticias (protocolo NNTP), etc. Tampoco podremos utilizar Wininet cuando desarrollemos una aplicación que utilice un protocolo no estándar, como pueden ser las aplicaciones de intercambio de ficheros, mensajería instantánea, etc.

Parece que la cosa se está liando, así que vamos a zanjar el tema aquí. Dejaremos los detalles sobre el protocolo TCP/IP y el API WinSock para otra ocasión. Símplemente nos basta con tener claro que nos estamos moviendo en niveles muy altos de abstracción y que los detalles internos quedan ocultos para nosotros, siendo manejados por el propio API.

En la siguiente figura podemos ver cómo viaja la información desde nuestro API Wininet hasta el servidor destino.

La secuencia lógica es la siguiente:

1. Nuestra aplicación cliente llama a las funciones del API Wininet.
2. El API Wininet traduce nuestras llamadas en accesos al API Winsock.
3. El API Winsock establece la conexión TCP/IP y envía las tramas de datos.
4. Los datos son transmitidos por internet, a través de un número indeterminado de nodos.
5. Los datos llegan al servidor en cuestión, interpretándolos según el protocolo utilizado.
6. El servidor retorna la respuesta, utilizando el mismo protocolo.

Nosotros tan sólo vamos a ocuparnos del primer punto, es decir: qué llamadas debemos hacer desde nuestras aplicaciones al API Wininet.

Funciones generales

Como ya hemos dicho, Microsoft ha definido un grupo de funciones de uso general, para tratar situaciones comunes a cualquier tipo de conexión, independientemente del protocolo que utilicemos.

Estas funciones se encargan, entre otras cosas, de establecer la conexión, manipular el marcado del modem, tratar cadenas de URL, mostrar diálogos de error, manipular *cookies*, etc.

Por cierto, se me olvidaba comentar a los programadores en Delphi, que el API Wininet lo podemos encontrar implementado en la unidad "Wininet.pas", así que será necesario añadir esta unidad en el *uses*.

Bueno, como todo esto puede ser bastante extenso, vamos a ir por partes.

Funciones para manejo de URLs

En esta ocasión vamos a empezar por lo fácil: dar una descripción de las funciones auxiliares que nos permiten manipular direcciones URL (*Uniform Resource Locator*).

Empezamos:

InternetCanonicalizeUrl

Esta función nos permite convertir una cadena con una URL a su equivalente pero con caracteres seguros. Las cadenas URL no pueden contener cualquier tipo de carácter, y existe un método para buscar qué caracteres no son seguros y sustituirlos por la secuencia equivalente. El ejemplo que todos conocemos es el carácter espacio y su secuencia "%20".

Hay que tener cuidado de no utilizar esta función con una cadena previamente codificada, porque no se realiza ningún tipo de comprobación y la re-codificará de nuevo.

```
function InternetCanonicalizeUrl(
            lpszUrl:     PChar;
            lpszBuffer:  PChar;
            var lpdwLongitudBuffer: LongWord;
            dwOpciones:  LongWord
        ): LongBool;
```

- **lpszUrl**: un puntero a una cadena con la URL a codificar.
- **lpszBuffer**: un puntero a una cadena donde se almacenará la URL resultante. Este puntero debe tener memoria previamente reservada.
- **lpdwLongitudBuffer**: un puntero a un valor que indica la longitud de caracteres que se han reservado para lpszBuffer. Cuando la función retorna, almacenará en este valor el número de caracteres copiados en lpszBuffer si la función se ejecutó correctamente, o el número de bytes necesitados si la función retornó error.
- dwOpciones: una serie de banderas que configuran el comportamiento de la función:
    - ICU_BROWSER_MODE: indica que se comportará como la codificación que hace el navegador, es decir, después del carácter "?" ó "#" no codificará nada.

- ICU_NO_ENCODE: No codifica nada.
- ICU_DECODE: convierte todas las secuencias codificadas (%XX) en su correspondiente cadena decodificada. Para usar esta bandera debe hacerse en combinación con ICU_NO_ENCONDE ya que en caso contrario, no tendrá efecto.
- ICU_ENCODE_PERCENT: Codifica los signos de porcentaje, que por defecto no son codificados. Sólo está disponible a partir de IE 5.
- ICU_ENCODE_SPACES_ONLY: Sólo codifica los espacios.

La función retorna TRUE o FALSE dependiendo de su éxito.

Si utilizamos esta función a partir de IE4, la bandera ICU_BROWSER_MODE se utilizará siempre, por lo que si queremos codificar la cadena completa, debemos recurrir a otras funciones, como UrlCanonicalize de la librería shlwapi.dll

InternetCombineUrl

Permite combinar una dirección base con una ruta relativa, dando como resultado una URL absoluta codificada. En otras palabras, nos combina el nombre del host con la ruta relativa, pasando el resultado por InternetCanonizaliceUrl.

```
function InternetCombineUrl(
             lpszUrlBase:     PChar;
             lpszUrlRelativa: PChar;
             lpszBuffer:      PChar;
             var lpdwLongitudBuffer: LongWord;
             dwOpciones:      LongWord
         ): LongBool;
```

- **lpszUrlBase**: un puntero a una cadena con el host.
- **lpszUrlRelativa**: un puntero a una cadena con la ruta relativa.
- **lpszBuffer**: un puntero a una cadena con espacio suficiente para albergar el resultado.
- **lpdwLongitudBuffer**: Un puntero a una variable que almacena la longitud reservada, en caracteres, de lpszBuffer. Cuando la función retorna correctamente, almacenará en esta variable la longitud de los caracteres copiados, o el espacio necesario si no ha ocurrido un error.
- **dwOpciones**: indica las opciones que se utilizarán para realizar la llamada interna a InternetCanonicalizeUrl. Se admiten los mismos valores que en dicha función.

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

InternetCrackUrl

Tranquilos, que esta función no tiene nada que ver con los hackers ni nada parecido, símplemente se encarga de "despedazar" una URL devolviendo sus componentes:

```
function InternetCrackUrl(
            lpszUrl:       PChar;
            dwLongitudUrl: LongWord;
            dwOpciones:    LongWord;
            var lpComponentesUrl: URL_COMPONENTS
         ): LongBool;
```

- **lpszUrl**: un puntero a una cadena con la dirección URL a "despedazar".

- **lpszLongitudUrl**: cero o el número de caracteres de la URL, si la cadena es UNICODE.

- **dwOpciones**: uno de los siguientes valores:

    - ICU_DECODE: convierte las cadenas codificadas en su original. Sólo se puede usar cuando se asigna un buffer a los campos de la estructura URL_COMPONENTS.

    - ICU_ESCAPE: convierte las secuencias de escape a su valor original. Sólo se puede usar cuando se asigna un buffer a los campos de la estructura URL_COMPONENTS.

- **lpComponentesUrl**: la dirección de una estructura de tipo URL_COMPONENTS en la que se retorna los valores de cada uno de los elementos de la URL. La estructura tiene los siguientes campos:

```
URL_COMPONENTS = record
        dwStructSize:      LongWord;
        lpszScheme:        PChar;
        dwSchemeLength:    LongWord;
        nScheme:           TInternetScheme;
        lpszHostName:      PChar;
        dwHostNameLength:  LongWord;
        nPort:             INTERNET_PORT;
        lpszUserName:      PChar;
        dwUserNameLength:  LongWord;
        lpszPassword:      PChar;
        dwPasswordLength:  LongWord;
        lpszUrlPath:       PChar;
        dwUrlPathLength:   LongWord;
        lpszExtraInfo:     PChar;
        dwExtraInfoLength: LongWord;
end;
```

  Los significados de los campos son:

    - **dwStructSize**: el tamaño en byes de la estructura. Símplemente nos basta con asignar el valor sizeof(URL_COMPONENTS).

    - **nScheme**: un valor enumerado que indica el tipo de esquema (protocolo) de la URL: http, ftp, gopher, mailto, etc.

    - **nPort**: el número de puerto de la URL. Si no aparece ninguno, cada protocolo cuenta con su número de puerto por defecto.

- **El resto de campos**: son pares de atributos para cada componente de la URL: el primero de ellos (lpszXXX) es un puntero a una cadena donde se almacena el componente. El segundo (dwXXX) es la longitud del buffer donde apunta el puntero.

  Si el puntero es **nil** y la longitud 0, no se retornará ningún valor para ese componente.

  Si el puntero es **nil** pero la longitud es distinta de 0, se almacena en el puntero la dirección donde empieza el componente dentro de la propia URL, es decir, nos retorna un puntero a una subcadena de lpszUrl y nos copia la longitud del componente en la variable longitud.

  Y finalmente, podemos almacenar en el puntero la dirección de un buffer, previamente reservado por nosotros, y establecer la variable longitud al número de caracteres que hemos reservado para este buffer en cuestión. Es este caso, la función nos copiará en cada uno de los buffers, el componente correspondiente, hasta el máximo indicado por la variable longitud.

  Estos son campos y su resultado, aplicando la siguiente URL:

  http://usuario:clave@www.servidor.com/ruta/recurso.html?param=valor

  - **lpszScheme y dwSchemeLength**: puntero y longitud a esquema (protocolo). En el ejemplo, el valor resultante es "http".
  - **lpszHostName y dwHostNameLength**: puntero y longitud al servidor. En el ejemplo, el valor es "www.servidor.com".
  - **lpszUserName y dwUserNameLength**: puntero y longitud al nombre de usuario. En el ejemplo, el valor es "usuario".
  - **lpszPassword y dwPasswordLength**: puntero y longitud a la clave de usuario. En el ejemplo, el valor es: "clave".
  - **lpszUrlPath y dwUrlPathLength**: puntero y longitud a la ruta relativa de la URL. En el ejemplo, el valor es: "/ruta/recurso.html".
  - **lpszExtraInfo y dwExtraInfoLength**: puntero y longitud a información adicional, como pueden ser los parámetros (a partir del carácter ?) o el ancla local (a partir del carácter #). En nuestro ejemplo, su valor es "?param=valor".

Esta función también retorna un valor booleano indicando su éxito o fracaso.

InternetCreateUrl

Esta función hace la operación inversa a la anterior, es decir: nos compone una cadena de URL a partir de sus componentes individuales.

```
function InternetCreateUrl(
            var lpComponentesUrl: URL_COMPONENTS;
            dwOpciones:           LongWord;
            lpszUrl:              PChar;
            var dwLongitudUrl:    LongWord
          ): LongBool;
```

No voy a entrar en más detalles, porque los parámetros se usan del mismo modo que con InternetCrackUrl, pero de forma inversa. Sólo hay que decir, que dentro de la estructura URL_COMPONENTS, se ignorarán aquellos componentes cuyo puntero se haya establecido a **nil**. Hay que ser cuidadoso también con la manipulación de los punteros a carácter, ya que es muy normal equivocarse.

La función retornará FALSE en caso de error y GetLastError nos retorna ERROR_INSUFFICIENT_BUFFER si el buffer lpszUrl es demasiado pequeño, o si hemos cometido algún error manipulando los punteros de caracteres.

Funciones de estado de la conexión

Uno de los principales problemas a la hora de conectarnos a internet, suele ser los distintos tipos de configuración y la comprobación de la conexión. Hasta ahora, podíamos apoyarnos en el API RAS (Remote Access Service) para consultar la configuración del módem, aunque a partir de Wininet, se nos ofrece una serie de funciones para estas tareas.

Antes de empezar, hay que dejar claro que sólo hay una manera de comprobar al 100% la existencia una conexión: intentando acceder a un recurso remoto (por ejemplo con un *ping*), pero esto suele ser una tarea lenta, en la que tenemos que dejar que se agote un tiempo de espera (*timeout*), para asegurarnos de que no hay nadie al otro lado escuchando nuestras peticiones. Para evitar este tiempo de espera, existen otras funciones que intentan averiguar el estado de conexión o la posibilidad de acceso a internet, sin intentar realizar una petición a la red. Concretamente se considera que un equipo no tiene conexión a internet cuando no tiene ni tarjeta de red ni módem configurado. Estas son las posibilidades, según la gente de Microsoft:

1. **IsNetworkAlive / IsDestinationReachable**: estas funciones sólo están disponibles si tenemos instalada cualquier versión de Internet Explorer a partir de la 5. Nos permiten comprobar una tipo de conexión o incluso hacer un ping a una URL determinada para garantizar la conexión. El principal inconveniente es que estas funciones no permiten la existencia de *firewalls* en la conexión, por lo que muchas redes de empresa no permitirían su uso.

2. **RasEnumConnections**: para conexión vía módem, podemos utilizar esta función del API RAS que nos permite consultar la configuración de los módems que tengamos instalados en nuestro sistema.

3. **InternetAttemptConnect**: esta función nos permite hacer un primer intento de conexión a internet. Si el usuario configuró su sistema como con "auto-marcado" (Panel de control - Opciones de internet - Conexiones - Marcar siempre la conexión predeterminada), esta función mostrará la ventana de marcado del módem. Si este intento falla, se activará la bandera global de conexión "Trabajar sin conexión".

4. **InternetCheckConnection**: según la documentación oficial, esta función intenta acceder a una URL dada haciendo un *ping*. La práctica (y sus propios autores) nos dice que no funciona, y que debe evitarse su uso (un buen trabajo de Microsoft).

5. **InetIsOffline**: esta función está exportada en la librería URL.DLL y según la documentación, nos indica que estamos conectados si la conexión está activa *o si todavía* no ha habido ningún intento de conexión. Por si fuera poco, la función nos informa de que estamos conectados símplemente si hay una tarjeta de red configurada en el sistema, aunque el cable esté desconectado. Por todo esto, se desaconseja el uso de esta función.

6. **InternetGetConnectedState**: a partir de IE 4 se nos ofrece esta función que nos permite distinguir configuraciones típicas como conexión por módem o conexión vía LAN, aunque no gestiona bien configuraciones complejas como LAN + router con auto-marcado. Además, esta función es capaz de retornarnos el valor de la bandera "Trabajar sin conexión".

7. **La bandera "Trabajar sin conexión"**: también a partir de IE 4, se gestiona una variable global (almacenada en el registro) que nos indica si debemos trabajar con o sin conexión a internet. Esta bandera suele actuar de un modo complementario con el resto de funciones y su valor puede consultarse o modificarse a través de la función InternetQueryOption. Para asegurarnos que pasamos a un estado "Trabajar con conexión", debemos llamar a la función InternetGoOnline. Esta bandera, junto con el uso de la función InternetGetConnectedState, parece ser el método más fiable de comprobar la conexión (si intentar un acceso a la red), de hecho, es el método que utiliza Microsoft para sus aplicaciones (IE, Outlook, etc.).

Ahora vamos a describir con más profundidad la sintaxis de algunas de estas funciones:

IsNetworkAlive

Esta función sólo la tendremos disponible a partir de IE 5, por lo que debemos asegurarnos de su existencia antes de usarla. Para ellos, lo mejor que podemos hacer es una carga dinámica de la librería Wininet.dll y de esta función. Para aprender más sobre la carga de librerías, podéis consultar el artículo de C++ Builder sobre "Librerías de enlace dinámico", aparecido en el número 4 de la revista Síntesis (www.grupoalbor.com).

```
function IsNetworkAlive(
            var dwTipoConexión: LongWord
  ): LongBool;
```

Esta función retornará TRUE o FALSE dependiendo si considera al sistema preparado para establecer una conexión a internet, según el método indicado en el parámetro. Hay que recordar que esta función no detectará una conexión correcta detrás de firewalls corporativos. Los valores permitidos para dwTipoconexión son:

- NETWORK_ALIVE_LAN: el sistema cuenta con tarjeta de red y conexión a red local.

- NETWORK_ALIVE_WAN: el sistema cuenta con una conexión remota activa.

IsDestinationReachable

También sólo disponible desde IE5, nos permite comprobar la conexión física a un host así como consultar la calidad de la conexión establecida.

```
function IsDestinationReachable(
            const lpszDestino: PChar;
            lpCalidad: LPQOCINFO
         ): LongBool;
```

Los parámetros son los siguientes:

- **lpszDestino**: una URL a la que se hará un *ping*.
- **lpCalidad**: un puntero a una estructura de tipo QOCINFO (Quality Of Connection INFO) con el siguiente aspecto:

```
QOCINFO = record
        dwSize:     LongWord;
        dwFlags:    LongWord;
        dwInSpeed:  LongWord;
        dwOutSpeed: LongWord;
end;
```

  El significado de cada uno de los campos es:

  - **dwSize**: como siempre, el tamaño de la estructura: sizeof(QOCINFO).
  - **dwFlags**: nos da información sobre el tipo de conexión disponible. Los valores permitidos son los mismos que para el parámetro dwTipoConexión de la función IsNetworkAlive.
  - **dwInSpeed**: velocidad de recepción de datos, en bytes por segundo.
  - **dwOutSpeed**: velocidad de envío de datos, en bytes por segundo.

La función nos retornará TRUE si el equipo tiene una conexión a internet abierta.

InternetAttemptConnect

Esta función se usa para hacer el primer intento de conexión, y permitir al sistema que marque el módem si así está configurado. La función es muy sencilla:

```
function InternetAttemptConnect(
            dwReservado: LongWord
         ): LongWord;
```

Como parámetro debemos pasar siempre un 0, y la función nos retornará el valor ERROR_SUCCESS si todo ha ido bien, o un código de error en caso contrario.

Hay que tener en cuenta que esta función nos abrirá la ventana de marcado del módem, si hemos configurado la opción de configuración del acceso telefónico a redes (en "Panel de control - Opciones de internet - Conexiones"), como "Marcar siempre la conexión predeterminada". Si en dicha ventana se pulsa el botón cancelar o "Trabajar sin conexión", se activará esta bandera global.

InternetGetConnectState

Esta función nos consultará el estado de conexión de nuestro sistema.

```
function InternetGetConnectedState(
             lpdwTipoConexion: PLongWord;
             dwReservado:      LongWord
         ): LongBool;
```

El segundo argumento es un valor de 32 bits que debe valer siempre 0, y el primer es un puntero a otro valor de 32 bits que contendrá el tipo de conexión. Los distintos valores combinados que se retornarán, son:

- INTERNET_CONNECTION_CONFIGURED: indica que hay algún enlace a internet configurado, pero no es seguro que esté activo en este momento.
- INTERNET_CONNECTION_LAN: se utiliza una red local para acceder a internet.
- INTERNET_CONNECTION_MODEM: se utiliza un módem para acceder a internet.
- INTERNET_CONNECTION_OFFLINE: la bandera de "Trabajar sin conexión" está activada, valor que puede haber adquirido desde cualquier otra aplicación que acceda a ella.
- INTERNET_CONNECTION_PROXY: se utiliza un proxy para acceder a internet.
- INTERNET_RAS_INSTALLED: El sistema tiene algún sistema de acceso remoto configurado.

La función retornará TRUE o FALSE, dependiendo del éxito en su ejecución.

InternetGoOnline

Esta función nos activa la bandera "Trabajar sin conexión", pidiendo permiso al usuario a través de una ventana de diálogo estándar. La ventana en cuestión podéis verla en la siguiente figura:


```
function InternetGoOnline(
             lpszURL:     PAnsiChar;
             hwndPadre:   HWND;
             dwReservado: LongWord
         ): LongBool;
```

Los parámetros tienen los siguientes significados:

- **lpszURL**: la URL a la que se quiere conectar.

- **hwndPadre**: la ventana padre del cuadro diálogo que puede mostrarse. El cuadro de diálogo de la imágen anterior es una ventana modal, por lo que debe indicarse el descriptor de la ventana padre. Un posible valor correcto puede ser el retornado por alguna de las funciones GetActiveWindow, GetDesktopWindow o GetForegroundWindow.

- **dwReservado**: debe ser 0.

Como siempre, se retornará TRUE o FALSE dependiendo de si el usuario ha pulsado en el botón "Conectar" o "Seguir desconectado" de la ventana.

Funciones para el marcado del módem

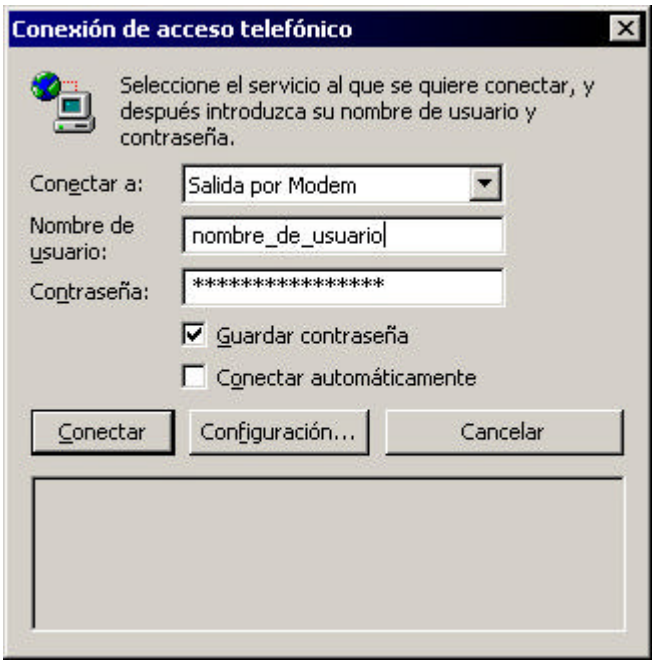
Una vez que tenemos claro que nuestro sistema se conecta a través del módem debemos establecer la llamada telefónica. Para ello podemos hacer uso de la función InternetAttemptConnect, que nos mostrará la ventana de marcado, o si queremos un mayor control, llamar a alguna de las siguientes funciones:

InternetAutodial

Establecerá una conexión con el módem configurado como predeterminado (a partir de IE 4). Esta función sólo mostrará la ventana de marcado (según aparece en imágen de más abajo) cuando así se haya configurado en las opciones del sistema (Panel de control - Opciones de internet - Conexiones).

```
function InternetAutodial(
             dwOpciones:  LongWord;
             hwndPadre:   DWORD
         ): LongBool;
```

```
Nota: en la conversión de la función
hecha en Delphi 5, existe un error
en el segundo parámetro, que se ha
definido como DWORD en vez de HWND.
```



El segundo parámetro será el descriptor de la ventana padre, al igual que en la función InternetGoOnline. El primer parámetro será un valor dentro de los siguientes posibles:

- INTERNET_AUTODIAL_FORCE_ONLINE: desactiva la bandera de "Trabajar sin conexión".

- INTERNET_AUTODIAL_FORCE_UNATTENDED: intenta hacer una conexión desatendida, marcando directamente el módem (aunque muestra una ventana de diálogo).

Esta función, como todas las demás, retorna un valor boolean indicando el éxito de la ejecución.

InternetAutodialHangup

Cuelga una llamada establecida con InternetAutodial:

```
function InternetAutodialHangup(
            dwReservado: LongWord
        ): LongBool;
```

El único parámetro debe pasarse como un 0, y el retorno nos indicará si se ha podido colgar la llamada o no.

InternetDial

Dado el carácter limitado de las funciones anteriores, se puede establecer el marcado del módem con esta otra función, que además nos permite utilizar distintos módems o configurar el comportamiento.

```
function InternetDial(
            hwndPadre:      HWND;
            lpszIdConexión: PAnsiChar;
            dwOpciones:     LongWord;
            lpdwConexión:   PLongWord;
            dwReservado:    LongWord
        ): LongWord;
```

Los parámetros tienen los siguientes significados:

- **hwndPadre**: el descriptor de la ventana padre, al igual que en InternetGoOnline.
- **lpszIdConexión**: una cadena con el nombre de la conexión a marcar. Se puede pasar **nil** para utilizar la conexión predeterminada, o bien utilizar la función RasEnumConnections para obtener los nombres de las conexiones disponibles.
- **dwOpciones**: son las opciones de marcado. Puede utilizarse cualquiera de los siguientes valores:
    - INTERNET_AUTODIAL_FORCE_ONLINE: Fuerza a desactivar la bandera "Trabajar sin conexión".
    - INTERNET_AUTODIAL_FORCE_UNATTENDED: Intenta realizar el marcado sin la intervención del usuario. En caso de que sea necesaria esta intervención, la función fallará.
    - INTERNET_DIAL_FORCE_PROMPT: Ignora la configuración de "Marcar automáticamente" y muestra siempre la ventana de marcado.
    - INTERNET_DIAL_UNATTENDED: Intenta realizar el marcado sin mostrar ningún tipo de interfaz. Si esto no es posible, mostrará algún tipo de ventana para que el usuario configure la conexión.
    - INTERNET_DIAL_SHOW_OFFLINE: Muestra el botón "Trabajar sin conexión" en vez del botón "Cancelar".

- **lpdwConexión**: es un puntero a una variable donde se almacenará el número asignado a la conexión. Este valor debe conservarse hasta la llamada a InternetHangup.

- **dwReservado**: Debe ser 0.

La función retorna ERROR_SUCCESS si todo ha ido correctamente, o uno de los siguientes errores:

- ERROR_INVALID_PARAMETER: Alguno de los parámetros es incorrecto.
- ERROR_NO_CONNECTION: Ha habido un problema.
- ERROR_USER_DISCONNECTION: el usuario ha pulsado en el botón "Cancelar" o "Trabajar sin conexión".

InternetHangup

Esta función debe utilizarse para colgar una conexión establecida con InternetHangup. Para ello, necesitamos conocer el número de conexión asignado por el sistema en la función InternetDial.

```
function InternetHangUp(
             dwConexión:  LongWord;
             dwReservado: LongWord
         ): LongWord;
```

En el primer parámetro debemos pasar el número de conexión, obtenido a través del cuarto parámetro de la función InternetDial. El segundo parámetro debe ser siempre 0.

La función retorna ERROR_SUCCESS si se ha conseguido colgar la conexión o un código de error en cualquier otro caso.

Funciones para comer galletas

Todos sabemos lo que son estas pequeñas *galletas* (*cookies*) que últimamente las vemos aparecer por cualquier lado. Para los olvidadizos, diremos que una *cookie* es un pequeño archivo almacenado por una página web en nuestro disco duro, para grabar en él datos de configuración, preferencias de usuario, contraseñas, etc. De este modo, la página en cuestión es capaz de reconocernos al entrar en la página, o saber si ha habido cambios desde nuestra última visita.

Desde el propio API Wininet también es posible gestionar estos pequeños archivos, sin tener que recurrir al navegador, Javascript, CGI o cualquier otro sistema de programación web.

Las *cookies* están asociadas a una URL concreta, así que por cada URL podremos tener un sólo archivo de configuración, y dentro de él, uno o varios pares de datos "variable-valor".

Existen dos tipos de *cookies*, las que tienen fecha de caducidad, y las que se eliminan al terminar una sesión de internet.

Las primeras (llamadas *cookies persistentes*) deben crearse indicando la fecha a partir de la cual dejan de ser válidas.

Las segundas (llamadas *cookies de sesión*) se crean sin ninguna fecha, y no se almacenarán en el disco duro, sino que permanecen en memoria hasta el final del proceso que las creó.

Las funciones para manipular las *cookies* son sencillas:

InternetSetCookie

Esta función nos permite almacenar una *cookie* en nuestro sistema a partir de la URL especificada. Para realizar esta operación, se consultará en los parámetros de seguridad configurados en Internet Explorer, por lo que es posible que aparezca una ventana de aviso o restricción si así está configurado.

```
function InternetSetCookie(
             lpszUrl:      PChar;
             lpszVariable: PChar;
             lpszValor:    PChar
         ): LongBool;
```

- **lpszUrl**: la URL asociada a la cookie. Si ya hay alguna cookie para esta URL, se utilizará el mismo archivo en el disco duro, en caso contrario, se creará uno nuevo.

- **lpszVariable**: nombre de la variable a almacenar. Este será un nombre que posteriormente utilizaremos para leer el valor. Si pasamos el valor **nil** (lo más habitual), se buscará el nombre dentro de la cadena lpszValor.

- **lpszValor**: valor asociado a la variable. Además del valor en sí, se pueden incluír otras palabras clave para configurar el comportamiento de la cookie. La sintaxis general de esta cadena es:

  <variable>=<valor>; [; expires=<fecha> [; domain=<nombre_dominio>]] [; path=<ruta>] [; secure] [; httponly]

  Los valores encerrados entre < y > indican que debe sustituirse por un valor concreto, y los encerrados entre [y] son opcionales. Los distintas opciones tienen los siguientes significados:

  - **expires**: indica la fecha de caducidad para la *cookie*. El valor <fecha> debe ser una fecha y hora según el meridiano de Greewich y debe indicarse en formato DIA, DD-MMM-AAAA HH:MM:SS GTM, donde:

    - DIA: es el día de la semana, en inglés. Los posibles valores son: Mon, Tue, Wed, Thu, Fri, Sat y Sun.

    - DD-MMM-AAAA, la fecha en este formato. Los meses deben indicarse con las tres primeras letras de su nombre en inglés, siendo los siguientes valores los posibles: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov y Dec.

    - HH:MM:SS: la hora según este formato. La parte de horaria debe indicarse en el formato 24 horas.

    - GMT: esto es una constante que debe aparecer al final de la cadena, indicando que la fecha representa un valor correcto para el meridiano de Greenwich.

  - **domain**: indica el dominio a partir del cual la cookie es válida. Por ejemplo, si indicamos como dominio "ideas.lawebdejm.com", la cookie será válida para cualquier URL dentro de este dominio y subdominios. Este parámetros sólo es válido para cookies persistentes por lo que será ignorado si no indicamos una fecha de caducidad.

- **path**: indica la ruta en el servidor a partir de la cual la cookie es válida. Por ejemplo, si indicamos la ruta "/ideas/delphi", la cookie será válida para las URLs de esta ruta, o cualquier otra dentro de esta.

- **secure**: indica que la cookie sólo se transmitirá y será válida si estamos accediendo a la URL a través del protocolo HTTPS (HTTP secure).

- **httponly**: indica que la cookie será codificada para leerse sólo desde los protocolos HTTP y HTTPS. En nuestro caso, esto significa que Wininet no tendría acceso a esta cookie.

La función retornará un valor booleano indicando éxito o fracaso.

InternetGetCookie

Esta función nos permite recuperar los valores de una cookie que previamente hayamos almacenado, ya sea a través de la función InternetSetCookie o desde una página web.

```
function InternetGetCookie(
            lpszUrl: PChar;
            lpszVariable: PChar;
            lpszValores: PChar;
            var lpdwTamaño: LongWord
          ): LongBool;
```

- **lpszUrl**: la URL asociada a la cookie. Es la URL a partir de la que se almacenó el valor.

- **lpszVariable**: No está implementado, ya que esta función retorna todas las variables de una cookie dada a través del parámetro lpszValores.

- **lpszValores**: apunta a un buffer en el que se copiarán los valores de la cookie. La sintaxis en que se retornan es <variable>=<valor> ; <variable>=<valor> ; <variable>=<valor>... Una vez que tenemos el valor de todas las variables, podemos tratar esta cadena desde nuestro programa, por ejemplo, buscando una variable concreta. Podemos evitar que se copien estos valores , pasando un puntero **nil**.

- **lpdwTamaño**: apunta a una variable en la que se indica el tamaño del buffer que pasamos en lpszValores. Si la función ha tenido espacio suficiente en el buffer, nos copiarán en esta variable el número de caracteres copiados. Si no ha tenido espacio suficiente, o hemos pasado lpszValores a **nil**, nos retornará el número de caracteres necesarios para copiar la cookie completa.

La función retornará TRUE o FALSE dependiendo de su éxito. En caso de error, GetLastError nos puede devolver alguno de estos errores, entre otros:

- ERROR_NO_MORE_ITEMS: No hay ninguna cookie asociada a la URL indicada.

- ERROR_INSUFFICIENT_BUFFER: el buffer pasado en lpszValores es demasiado pequeño. En este caso se nos copiará en la variable dwTamaño el espacio requerido.

Terminando...

Creo que como introducción ya está bien. No hemos entrado en los temas interesantes de Wininet, sino que hemos dado una panorámica de las funciones auxiliares que nos ayudarán a la hora de desarrollar aplicaciones con acceso a internet.

En el próximo artículo entraremos en harina, y abarcaremos todo el tema de acceso a internet a través de las función para el protocolo HTTP.

Los ejemplos

Todo lo que hemos ido explicando, se utiliza de modo práctico en el siguiente ejemplo:

Delphi 5
Un pequeño proyecto que demuestra el uso de las funciones sobre gestión de URLs y estado de la conexión:

Ejemplo en zip - www.lawebdejm.com/?id=22211

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