Capítulo 1

Delphi el API de Windows

Delphi ha traído una nueva era a la programación en Windows. Nunca antes ha sido tan sencillo crear aplicaciones potentes y robustas para el entorno Windows en tan cortos plazos de tiempo. Ahora en su quinta versión, Delphi se conoce mundialmente como el entorno de desarrollo visual por excelencia para Windows. Ninguna otra herramienta de programación puede siquiera acercarse a la potencia, facilidad de uso y calidad de los ejecutables de Delphi.

Uno de los puntos fuertes de Delphi es la Librería de Componentes Visuales (Visual Component Library), el modelo de objetos de Borland. Este modelo de objetos ha permitido al equipo de desarrolladores de Delphi encapsular la casi totalidad del tedioso proceso de programación para Windows en componentes de fácil utilización.

Los anteriores lenguajes de programación para Windows exigían al desarrollador escribir grandes cantidades de código sólo para exprimir de Windows un mínimo de funcionalidad. El simple acto de crear una ventana e interceptar acciones de menú ocupaba páginas enteras de código. La excelente encapsulación por parte de Delphi de las tediosas exigencias de la programación para Windows ha convertido lo que una vez fue una tarea monótona en una experiencia divertida y excitante.

El API de Windows vs. la VCL

El equipo de desarrollo de Delphi hizo un trabajo excelente al encapsular en la VCL la amplia mayoría de la funcionalidad del vasto API de Windows. Sin embargo, debido a las dimensiones de este API, sería imposible e impráctico encapsular cada una de sus funciones en un objeto de Object Pascal. Para alcanzar ciertos objetivos o resolver un problema específico, un programador puede verse obligado a utilizar funciones de más bajo nivel del API de Windows que sencillamente no están encapsuladas en ningún objeto Delphi. Puede ser necesario, por otra parte, extender la funcionalidad de un objeto Delphi, y si este objeto encapsula alguna parte concreta del API de Windows, la extensión deberá realizarse, probablemente, utilizando también en gran medida los recursos del API.

Tipos de datos de Windows

Las funciones del API de Windows utilizan ciertos tipos de datos que pueden no ser familiares para los programadores ocasionales de Delphi. Estos tipos se importan de ficheros de cabecera originales en C que definen la estructura de las funciones del API de Windows. En su gran mayoría, estos nuevos tipos de datos son sencillamente tipo$_s$ de datos de Pascal que han sido renombrados para hacerlos similares a los tipos de datos originalmente utilizados en lenguajes de programación anteriores. Esto se hizo así para facilitar que los programadores experimentados comprendieran los tipos de parámetros y valores de retorno de las funciones del API, y que los prototipos de las funciones fuesen muy parecidos a los mostrados en la documentación del API de Windows, para evitar posibles confusiones. La siguiente tabla muestra los tipos de datos más comunes de Windows y sus equivalentes en Object Pascal.

Tabla 1-1: Tipos de datos de Windows

Tipo Windows	Tipo Object Pascal	Descripción
LPSTR	PAnsiChar	Puntero a cadena de caracteres.
LPCSTR	PAnsiChar	Puntero a cadena de caracteres.
DWORD	LongWord	Número entero sin signo.
BOOL	LongBool	Valor booleano.
PBOOL	^BOOL	Puntero a valor booleano.
PByte	^Byte	Puntero a valor byte.
PINT	^Integer	Puntero a valor entero
PSingle	^Single	Puntero a número de punto flotante de precisión simple.
PWORD	^Word	Puntero a valor de 16 bits.
PDWORD	^DWORD	Puntero a valor de 32 bits.
LPDWORD	PDWORD	Puntero a valor de 32 bits.
UCHAR	Byte	Valor de 8 bits (puede representar un carácter).
PUCHAR	^ Byte	Puntero a valor de 8 bits.
SHORT	Smallint	Número entero de 16 bits.
UINT	LongWord	Número entero de 32 bits (sin signo).
PUINT	UINT	Puntero a entero de 32 bits.
ULONG	Cardinal	Número entero de 32 bits (sin signo).
PULONG	^ULONG	Puntero a entero de 32 bits.
PLongInt	^LongInt	Puntero a valor de 32 bits.
PInteger	^Integer	Puntero a valor de 32 bits.
PSmallInt	^Smallint	Puntero a valor de 16 bits.
PDouble	^Double	Puntero a número de punto flotante de doble precisión.
LCID	DWORD	Identificador local.
LANGID	Word	Identificador de lenguaje.

Tipo Windows	Tipo Object Pascal	Descripción
THandle	Integer	Identificador (manejador) de objeto. Muchas funciones del API de Windows retornan un valor de este tipo, que identifica a un objeto dentro de las tablas internas de objetos de Windows.
PHandle	^THandle	Puntero a manejador.
WPARAM	LongInt	Parámetro de mensaje de 32 bits. En versiones anteriores de Windows era un tipo de datos de 16 bits.
LPARAM	LongInt	Parámetro de mensaje de 32 bits.
LRESULT	LongInt	Valor de retorno de función de 32 bits.
HWND	Integer	Manejador de ventana. Todos los controles, ventanas principales o hijas, etc. tienen su manejador de ventana correspondiente, que les identifica dentro de las tablas internas de Windows.
HHOOK	Integer	Manejador de un "gancho" *(hook)* de sistema instalado.
ATOM	Word	Índice de una cadena de caracteres dentro de la tabla de átomos locales o globales.
HGLOBAL	THandle	Manejador que identifica un bloque de memoria global reservado dinámicamente. En las versiones de Windows de 32 bits, no hay distinción entre la memoria dinámica reservada global o localmente.
HLOCAL	THandle	Manejador que identifica un bloque de memoria local reservado dinámicamente. En las versiones de Windows de 32 bits, no hay distinción entre la memoria dinámica reservada global o localmente.
FARPROC	Pointer	Puntero a procedimiento. Se utiliza normalmente como tipo de parámetro en funciones que requieren que se suministre la dirección de una función de respuesta *(callback function).*
HGDIOBJ	Integer	Manejador de objeto gráfico. Los contextos de dispositivo, plumas, brochas, etc. tienen un manejador de este tipo que los identifica dentro de las tablas internas de Windows.
HBITMAP	Integer	Manejador de objeto de mapa de bits de Windows.
HBRUSH	Integer	Manejador de objeto de brocha de Windows.
HPEN	Integer	Manejador de objeto de pluma de Windows.
HDC	Integer	Manejador de contexto de dispositivo de Windows.
HPALETTE	Integer	Manejador de objeto de paleta de colores de Windows.
HFONT	Integer	Manejador de objeto de fuente lógica de Windows.
HICON	Integer	Manejador de objeto de icono de Windows.
HMENU	Integer	Manejador de objeto de menú de Windows.
HMETAFILE	Integer	Manejador de objeto de metafichero de Windows.
HENHMETAFILE	Integer	Manejador de objeto de metafíchero mejorado de Windows.
HRGN	Integer	Manejador de objeto de región de Windows.
HINST	Integer	Manejador de objeto de instancia.

Tipo Windows	Tipo Object Pascal	Descripción
HMODULE	HINST	Manejador de objeto de un módulo.
HRSRC	Integer	Manejador de objeto de recurso de Windows
HKL	Integer	Manejador de configuración de teclado.
HFILE	Integer	Manejador de fichero abierto.
HCURSOR	HICON	Manejador de objeto de cursor de Windows
COLORREF	DWORD	Valor de referencia de color de Windows, que contiene valores para los componentes rojo, verde y azul de un color.

Manejadores

Un concepto fundamental de la programación Windows es el de manejador de objeto. Muchas funciones devuelven un manejador de objeto que la función crea o carga desde un recurso. Funciones como CreateWindow y CreateWindowEx devuelven un manejador de ventana. Otras funciones devuelven un manejador de fichero abierto, o un manejador de un heap recién reservado en memoria, corno HeapCreate. Internamente, Windows mantiene la información necesaria sobre todos esos objetos, y los manejadores sirven como enlace entre el objeto y la aplicación. Éste es el mecanismo que permite que una aplicación se comunique con el sistema operativo. A través de los manejadores, una aplicación puede fácilmente referirse a cualquiera de esos objetos, y el sistema operativo sabrá instantáneamente qué objeto desea manipular la aplicación.

Constantes

Las funciones del API de Windows declaran miles de constantes diferentes para que sean utilizadas como valores de parámetros. En el fichero **Windows.PAS** se definen constantes para todo tipo de usos, desde valores de colores hasta valores de retorno. Las constantes definidas para cada función del API se listan junto con la función en cuestión dentro de ese fichero. Sin embargo, **Windows.PAS** puede ofrecer más información con relación a las constantes asociadas a cualquier función particular; por ello, una buena regla a tener siempre en cuenta es la de comprobar este fichero al utilizar funciones complejas.

Cadenas de caracteres

Todas las funciones del API de Windows que utilizan cadenas requieren un puntero a un *array* de caracteres terminados en nulo. Windows ha sido escrito en C, que no ofrece el tipo de cadena de Pascal. Las versiones iniciales de Delphi exigían que la aplicación reservara un buffer para la cadena y convirtiera la variable de tipo String a PChar. Sin embargo, a partir de Delphi 3 el formato interno de las cadenas y un nuevo mecanismo de conversión permiten utilizar una cadena como un PChar mediante una simple conversión de tipo (por ejemplo, *PChar(MiCadena),* donde *MiCadena* es una

variable declarada corno *MiCadena:* **String).** En la mayoría de los casos, esta conversión podrá utilizarse al llamar a una función del API de Windows que requiera un parámetro de tipo cadena de caracteres.

Importación de funciones de Windows

El API de Windows es enorme. Define funciones para casi cualquier utilidad o acción que un programador podría imaginar. Debido al generoso volumen del API de Windows, algunas funciones simplemente han sido "olvidadas" y no han sido importadas por las librerías de Delphi. Por cuanto todas las funciones del API de Windows son sencillamente funciones exportadas de DLLs, importar una nueva función del API de Windows es un proceso relativamente simple, siempre que se conozcan los parámetros de la misma.

Importar una nueva función del API de Windows es exactamente igual a importar cualquier otra función de una DLL. Por ejemplo, supongamos que la función *BroadcastSystemMessage* descrita en el capítulo "Funciones de Gestión de Mensajes" no estuviese importada en el código fuente incluido en Delphi[1]. Para importar esa función y poder utilizarla en una aplicación, bastaría con declararla como:

```
function BroadcastSystemMessage(Flags: DWORD; Recipients: PDWORD;
  uiMessage: UINT; wParam: WPARAM; 'Paran]: LPARAM): Longlnt; stdcall;

implementation

function BroadcastSystemMessage; external user32 name 'BroadcastSystemMessage' ;
```

Siempre que se conozcan los tipos de los parámetros exigidos por la función *y* el nombre de la DLL que contiene la función, cualquier función del API de Windows puede ser importada y utilizada por una aplicación Delphi. Es importante destacar que la directiva **stdcall** debe añadirse siempre al prototipo de la función, por cuanto éste es el mecanismo estándar mediante el cual Windows pasa los parámetros a la función a través de la pila.

Funciones importadas incorrectamente

Algunas funciones han sido importadas incorrectamente en el código fuente de Delphi. Esas excepciones se señalan en las descripciones individuales de las funciones. En la mayoría de los casos, las funciones que han sido incorrectamente importadas tienen relación con la posibilidad de pasar el valor **nil** como valor a un parámetro de tipo puntero, generalmente para recuperar el tamaño necesario para un *buffer* que la aplicación debe reservar dinámicamente una vez conozca la longitud necesaria. En

1. De hecho, no lo estaba en la versión de Windows.PAS incluida en Delphi 3. Esta omisión fue corregida en Delphi 4.

Delphi, algunas funciones de este tipo, han sido importadas con parámetros definidos como **var** o **const.** Estos tipos de Parámetros aceptan un puntero a un *buffer, pero* nunca pueden recibir **nil,** limitando de este modo la utilización de la función dentro de Delphi. Corno ocurre casi siempre con Delphi, el problema es muy fácil de resolver Simplemente vuelva a importar la función, según hemos descrito antes. Las funciones que han sido importadas incorrectamente se identifican en el libro en las descripciones individuales de cada función.

Funciones de respuesta

Otro concepto muy importante de la programación Windows es el de función de respuesta (callback limetion). Una función de respuesta es una función dentro de la aplicación que no es llamada directamente por ninguna otra función o procedimiento de la aplicación, sino que es llamada por el sistema operativo. Esto le permite a Windows comunicarse directamente con la aplicación, pasándole los parámetros requeridos por la función de respuesta en cuestión. La mayoría de las funciones de enumeración requieren algún tipo de función de respuesta definida por la aplicación que reciba la información que se enumera.

Las funciones de respuesta individuales tienen parámetros específicos que deben ser declarados exactamente por la aplicación. Esto es necesario para que Windows pase a la función la información correcta en el orden correcto. Un buen ejemplo de función que utiliza una función de respuesta es EnumWindows. Esta función recorre todas las ventanas de nivel superior en la pantalla, pasando el manejador de cada ventana a la función de respuesta definida por la aplicación. El proceso continúa hasta que todas las ventanas hayan sido enumeradas o la función de respuesta devuelva FALSE. La función de respuesta utilizada por EnumWindows se define corno:

```
EnumWindowsProc(
  hWnd: HWND;                  {manejador de ventana de alto nivel}
  IParam: LPARAM               {datos definidos por la aplicación}
): BOOL;                       {devuelve TRUE o FALSE}
```

Una función con este prototipo deberá definirse dentro de la aplicación, y un puntero a ella deberá pasarse como parámetro a la función *Enum Windows.* El sistema operativo llamará a la función de respuesta una vez por cada ventana de nivel superior en la pantalla, pasando cada vez el manejador de una de ellas como parámetro. Es importante destacar que la directiva **stdcall** debe ser añadida al prototipo de la función de respuesta, ya que ese es el mecanismo estándar de traspaso de parámetros que utiliza Windows. Por ejemplo, la función de respuesta anterior deberá tener el siguiente prototipo:

```
function EnumWindowsProc(hWnd: HWND; lParam: LPARAM) BOOL; stdcall;
```

Este poderoso mecanismo de *software* permite en muchos casos que una aplicación recupere información sobre el sistema que es almacenada sólo internamente por

Windows y que de otro modo sería totalmente inalcanzable. Para un ejemplo completo de utilización de funciones de respuesta, consulte la función *EnumWindows, y* muchas otras funciones a lo largo del libro.

Parámetros de funciones

La gran mayoría de las funciones del API de Windows simplemente reciben los parámetros estáticos que se le envían y realizan cierta tarea en base a los valores de los parámetros. Sin embargo, ciertas funciones devuelven valores que deberán ser almacenados en un *buffer, y* este *buffer* deberá pasarse a la función en forma de puntero. En la mayoría de los casos en que la descripción de una función específica que ésta devuelve algún valor a través de un *buffer* de cadena terminada en carácter nulo o de una estructura de datos, estos *buffers* o estructuras deberán ser reservados por la aplicación antes de llamar a la función.

En muchos casos, la documentación de un parámetro puede especificar que éste puede contener uno o más valores de una tabla. Esos valores se definen en forma de constantes, y pueden combinarse utilizando el operador **or.** Un valor concreto de ese parámetro generalmente identifica una máscara de bits, donde el estado de cada bit tiene un significado concreto para la función. Es por eso que las constantes pueden combinarse mediante operaciones de manipulación de bits. Por ejemplo, la función *CreateWindow* tiene un parámetro llamado *dwStyle* que puede aceptar un número variable de constantes combinadas mediante el operador **or.** Para pasar más de una constante a la función, al parámetro deberá asignársele una valor como "WS CAPTION **or** WS_CHILD **or** WS CLIPCHILDREN". Ello hará que se cree una ventana hija que tendrá una barra de título y no dibujará en el área ocupada por sus ventanas hijas al redibujarse.

A la inversa, cuando la documentación de una función específica que ésta devuelve uno o más valores definidos como constantes, el valor devuelto puede combinarse con cualquiera de las constantes utilizando el operador **and** para determinar si la constante está incluida en el valor de retorno. Si el resultado de la combinación es igual a la constante (por ejemplo, **if** (Result **and** WS_CHILD) = WS_CHILD **then...),** la constante está incluida en el valor devuelto por la función.

Unicode

Originalmente, el software necesitaba únicamente un byte para definir un carácter de un conjunto de caracteres. Ello permitía hasta 256 caracteres diferentes, lo que era más que suficiente para el alfabeto, los dígitos, los signos de puntuación y los símbolos matemáticos comunes. Sin embargo, debido al desarrollo de la comunidad global y la subsiguiente internacionalización de Windows y el software para Windows, se necesitaba un nuevo método para identificar caracteres. Muchos idiomas utilizan más de 256 caracteres diferentes en su escritura, mucho más de lo que puede describirse con un byte. Por todo ello surgió Unicode. Un carácter Unicode ocupa 16 bits, lo que

permite identificar 65.535 caracteres diferentes. Para acomodar el nuevo conjunto de caracteres, muchas funciones del API de Windows se ofrecen en dos versiones diferentes: ANSI y Unicode. Cuando revise el fichero **Windows.PAS**, verá funcione definidas con una 'A' o 'W' añadidas al final del nombre de la función, identificando así las versiones ANSI o *Wide* (Unicode) de la función. En este libro nos centraremos en la descripción de las versiones ANSI de las funciones del API. Sin embargo, las funciones Unicode generalmente difieren únicamente en el tipo de las cadenas pasada a la función, por lo que el texto de este libro describirá de forma adecuada el comportamiento de la versión Unicode de las funciones.