16. Programación de sonido con los comandos MCI

16.1. Introducción

La programación de sonido bajo Windows se puede llevar a cabo con dos herramientas diferentes, los comandos MCI y las funciones de bajo nivel del API de Windows. En este capítulo estudiaremos la primera forma, dejando la programación de bajo nivel para el próximo capítulo. Aunque nos centraremos en Windows 95, la mayoría de lo indicado es directamente extrapolable a Windows 3.1.
En ambos capítulos asumimos ciertos conocimientos de programación por parte del lector, por lo que si carece de ellos, o sencillamente no le interesa este tema puede pasar directamente al capítulo 18.

16.2. Programación en Windows

La programación en Windows tenía fama (justificada) de ser complicada, pero los temores comenzaron a desvanecerse con la llegada de los lenguajes de programación visual. Hoy en día existen varias alternativas para afrontar estas tareas sin excesivos quebraderos de cabeza. A continuación exponemos las más utilizadas, por orden creciente de complejidad (y de prestaciones).

1. Los lenguajes de autor, como *Director* o *Toolbook*, son excelentes herramientas para iniciarse en la programación multimedia en Windows sin necesidad de un gran bagaje informático. Su fuerte es la realización de interactivos, pero poseen enormes limitaciones para la realización de programas de propósito más general.
2. *Visual Basic* es un entorno de programación de Microsoft basado en el popular lenguaje BASIC, que se ha convertido, por su excelente relación entre esfuerzo y prestaciones, en el favorito de muchos programadores. Con *Visual Basic* se puede escribir casi cualquier programa para Windows, si bien con entornos más complejos los resultados podrán ser más eficientes.
3. *Delphi* es un entorno de Borland, basado en el lenguaje Pascal. La curva de aprendizaje no es muy superior a la de *Visual Basic*, y sus prestaciones son algo mejores. Goza sin embargo de una menor difusión.
4. Los entornos basados en C++ (*Microsoft Visual C++* y *Borland C++*), pueden ser considerados como los pesos pesados de la programación para Windows. Si un programa no puede realizarse con cualquiera de ellos, será sencillamente un programa irrealizable. Aunque las prestaciones superan a las de cualquier otro entorno, la curva de aprendizaje y el tiempo de desarrollo cobran cara esta superioridad.

A raíz de lo expuesto, y en la medida de lo posible, hemos optado por utilizar Visual Basic en los ejemplos de este capítulo, con la certeza de que quien quiera llevarlos a cabo en un entorno basado en el lenguaje C no tendrá ninguna dificultad. En último término, e independientemente del lenguaje utilizado, serán frecuentes las llamadas a funciones C, ya que

gran parte de la programación de sonido (como de la programación multimedia en general), se realiza a través de llamadas a funciones de las API de Windows (que como el resto del sistema, están escritas en este lenguaje).

16.3. La programación multimedia en Windows

La programación de sonido (tanto de audio digital como de MIDI) es un caso particular de la programación multimedia. Existen actualmente numerosas obras que tratan este tema en profundidad, y sería absurdo pretender competir con ellas desde las limitadas páginas de este capítulo, por lo que, si desea ahondar en esta cuestión, sin duda hará bien en adquirir alguno de estos libros[1]. A pesar de ello, la información que aquí se da es autocontenida, e incluso en algunos puntos (especialmente en los referentes a la programación MIDI de bajo nivel, abordada en el próximo capítulo), trata cuestiones que no suelen aparecer en la mayoría de manuales.

Si tal como indicábamos, la programación en Windows era hasta hace poco bastante compleja, lo cierto es que comparativamente, la programación multimedia ha sido, desde el principio, relativamente sencilla. La razón de ello, está en la normalización y la simplificación que supusieron las extensiones multimedia añadidas a la versión 3.0 de Windows, e integradas definitivamente en la versión 3.1.

Estas extensiones posiblemente fueran lo mejor de un sistema un tanto precario (16 bits, multitarea cooperativa, etc.), si lo comparamos con otros entornos GUI[2] coetáneos, y constituyeron probablemente una de las principales puntas de lanza para el imparable avance de Windows. En este sentido, Windows 95, que puede ya competir sin vergüenza con cualquier sistema operativo actual, ha aportado mejoras internas, pero muy pocas modificaciones substanciales en el tema del multimedia.

A grosso modo, el programador de Windows puede abordar el multimedia a través de dos enfoques diferentes: el MCI y las funciones de bajo nivel. El MCI es el método más sencillo (y también el más documentado) y será el que trataremos primero. Cuando sus posibilidades se queden cortas, habrá que recurrir a las funciones de bajo nivel, que estudiaremos en el próximo capítulo.

16.4. Introducción a los comandos MCI

Las siglas MCI corresponden a *Media Control Interface*, un conjunto de herramientas de programación con las que se logra una funcionalidad muy similar a la del reproductor de medios de Windows (de hecho, esta aplicación las utiliza casi exclusivamente).

Uno de los puntos más relevantes del MCI, es que trata del mismo modo todos los dispositivos multimedia (reproductor de CD, vídeo analógico y digital, videodisco, audio

[1] Le recomendamos *Microsoft Windows Multimedia Programmer´s Reference* y *Microsoft Windows Multimedia Programmer´s Workbook*, de Microsoft Press, ambos absolutamente exhaustivos, o el más accesible *Programación Multimedia* de Anaya Multimedia.

[2] *Graphical User Interface*, o Interfaz gráfico de usuario.

digital y MIDI). Esto supone una simplificación pero también es una importante fuente de confusiones, ya que muchos de los dispositivos involucrados guardan poca relación entre sí. Para acabar de liar al neófito, diremos que existen dos interfaces de uso de los comandos MCI (es como si dispusiéramos de dos gramáticas para un único vocabulario), el interfaz de cadenas de comandos (*command-string interface*) y el interfaz de mensajes de comandos (*command-message interface*). Ambos pueden además mezclarse en una misma aplicación, aunque en aras de una mayor claridad, es preferible no hacerlo.

16.5. Los dispositivos MCI

La lista de dispositivos MCI disponibles, varía de un ordenador a otro en función de los *drivers* instalados. Consulte el icono **Multimedia** en el panel de control para comprobar los que incorpora su sistema. Active la pestaña de modo de visualización **Avanzado** y seleccione **Dispositivos de control multimedia**; le aparecerá una lista similar a la de la figura 16.1.

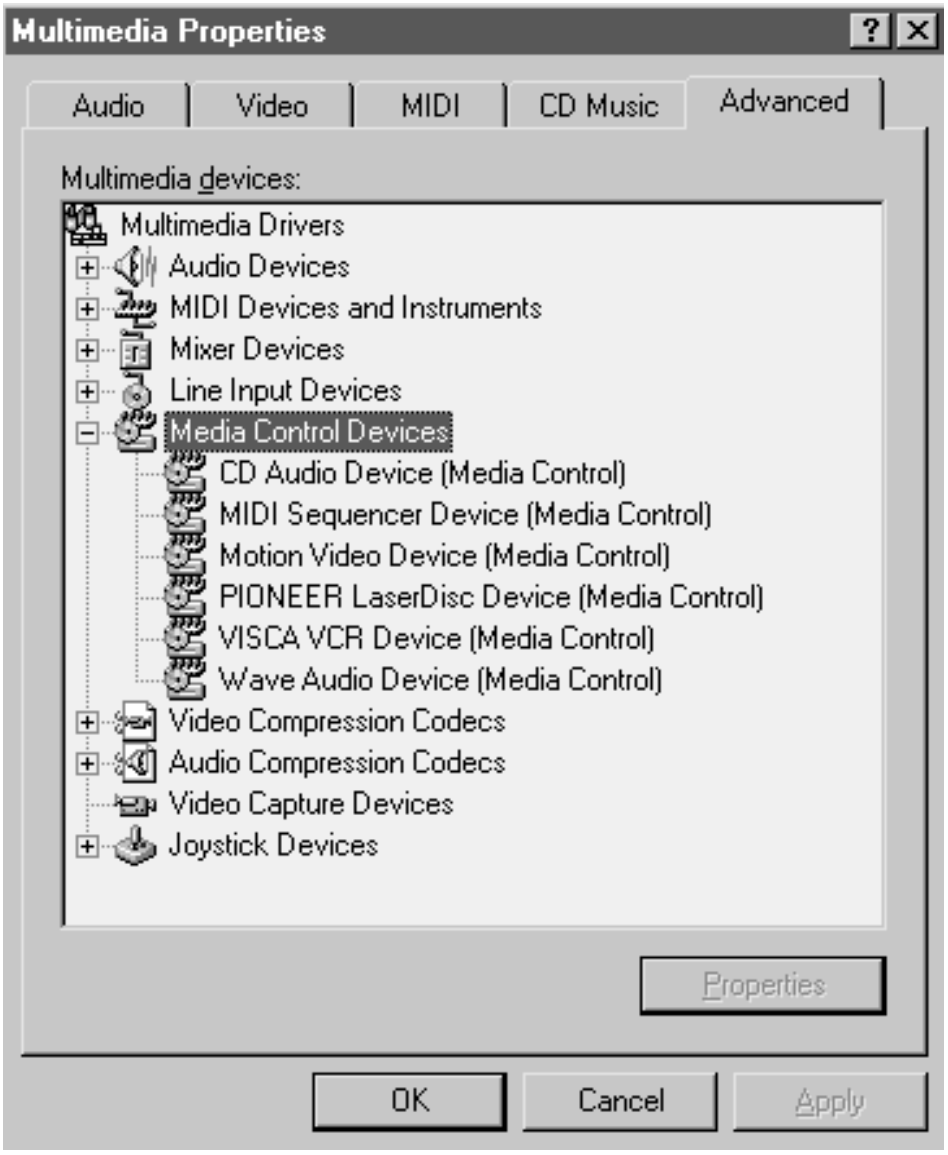

Figura 16.1. Lista de dispositivos MCI instalados.

La lista de dispositivos disponibles coincidirá con las entradas de la sección [mci] en el fichero *system.ini*, que puede ofrecer un aspecto similar al siguiente:

```
[mci]
cdaudio=mcicda.drv
sequencer=mciseq.drv
waveaudio=mciwave.drv
avivideo=mciavi.drv
```

La lista del ejemplo es una lista básica que el instalador de Windows coloca, si detecta la presencia de una tarjeta de sonido y de una unidad de CD-ROM. En el caso de otros dispositivos menos frecuentes, como lectores de videodisco, scanners, etc. la instalación de los drivers y la modificación del *system.ini*, se llevará a cabo desde el software que acompañe al aparato. A continuación se muestra en la tabla 16.1, la lista completa de todos los posibles dispositivos MCI:

Dispositivo	Constante	Descripción
animation	MCI_DEVTYPE_ANIMATION	Dispositivo de animación (Video for Windows o Quick Time)
cdaudio	MCI_DEVTYPE_CD_AUDIO	Reproductor de CD audio
dat	MCI_DEVTYPE_DAT	Reproductor de cintas DAT
digitalvideo	MCI_DEVTYPE_DIGITAL_VIDEO	Vídeo digital en una ventana
overlay	MCI_DEVTYPE_OVERLAY	Vídeo analógico en una ventana
scanner	MCI_DEVTYPE_SCANNER	Scanner de imágenes
sequencer	MCI_DEVTYPE_SEQUENCER	Secuenciador MIDI
vcr	MCI_DEVTYPE_VCR	Magnetoscopio de vídeo
videodisk	MCI_DEVTYPE_VIDEODISC	Reproductor de vídeo disco
waveaudio	MCI_DEVTYPE_WAVEFORM	Ficheros de onda

Tabla 16.1. Dispositivos MCI

Los nombres de la columna **dispositivo** son los que utilizaremos en las cadenas de comandos, mientras que las **constantes** se utilizan en los mensajes de comandos (véase 16.7). Aunque no comentaremos nada sobre los dispositivos no-sonoros, se puede comprobar que las posibilidades de control que ofrecen los comandos MCI son poco desdeñables.

16.6. El interfaz de cadenas de comandos

16.6.1. Introducción

Este interfaz es el más sencillo de los dos, y el utilizado en lenguajes como Visual Basic. La estructura típica de una cadena de comandos es de la forma:

comando objeto [modificadores adicionales]

- *comando* indica la acción a ejecutar,

- *objeto* es el nombre de un dispositivo o un fichero asociado,
- y los modificadores adicionales, varía en número y tipo, en función del comando y del objeto al que se aplican.

A continuación se indican en la tabla 16.2 algunos de los comandos MCI de uso más frecuente[3]. Los nombres de la segunda columna corresponden a las constantes utilizadas por la interfaz de mensajes (véase 16.7).

Comando	Constante	Descripción
open	MCI_OPEN	abre un dispositivo MCI
close	MCI_CLOSE	cierra un dispositivo MCI
play	MCI_PLAY	inicia la reproducción
record	MCI_RECORD	inicia la grabación
stop	MCI_STOP	finaliza la reproducción o la grabación
pause	MCI_PAUSE	pausa en reproducción o en grabación
resume	MCI_RESUME	reinicia la reproducción o la grabación
set	MCI_SET	asigna un valor a un modificador
status	MCI_STATUS	pide información sobre el estado del dispositivo
capability	MCI_GETDEVCAPS	pide información sobre las posibilidades del dispositivo

Tabla 16.2. Comandos MCI más frecuentes

Aunque la estructura de las cadenas sea la misma, no todos los comandos son aplicables a todos los dispositivos. Un CD Audio no puede, por ejemplo, grabar, por lo que la cadena **record cdaudio**, es incorrecta.

Veamos un ejemplo con el dispositivo de CD audio

```
open cdaudio                  //Abre el dispositivo de CD audio.
set cdaudio timeformat tmsf   //El formato será
                              //pista:min:seg:frame
play cdaudio from 6 to 7      //Reproduce la pista 6 entera
......
close cdaudio                 //Cierra el dispositivo
```

- Las cadenas no tienen por qué ir seguidas, ya que podrían estar activadas, por ejemplo, por acciones del ratón, pero algunas sí que deben estar separadas (en este caso **play** y **close**).
- Cada dispositivo tiene un formato de tiempo por defecto. En realidad la segunda línea, no es necesaria puesto que el formato por defecto del CD audio ya es éste, pero utilizaríamos esta instrucción si quisiéramos modificarlo (directamente a segundos, por ejemplo).
- Asimismo, si no pusiésemos ningún rango en la instrucción **play**, el CD se reproduciría de principio a fin.
- En los dispositivos que como **animation**, **sequencer** o **waveaudio** utilizan ficheros, el nombre del fichero se debe indicar en algún comando (normalmente **open**). Veamos un ejemplo con un fichero de onda:

[3]Existen más de 40 comandos diferentes.

```
open c:\waves\cascabel.wav type waveaudio alias cascabel
//Abre un fichero, se le indica que es de tipo waveaudio, y se le
//asigna un alias para podernos referir a él en comandos sucesivos.

set cascabel time format samples //Se indica un formato de tiempo

play cascabel from 1 to 1000
//Reproduce de la muestra 1 a la 1000
...........
close cascabel //Cierra el dispositivo
```

Esta es la forma más general, pero se puede simplificar enormemente. El tipo de dispositivo no es necesario cuando se trabaja con ficheros, ya que el sistema es capaz de detectar el tipo automáticamente, pero sólo si la extensión está definida en la sección [mci extensions] del fichero *system.ini*, en la forma:

```
[mci extensions]
wav=waveaudio
mid=sequencer
```

El **alias** tampoco es imprescindible, aunque nos permitirá trabajar con varios dispositivos del mismo tipo (varios ficheros de sonido). El formato de tiempo tampoco es necesario en este caso, pues éste es el formato defecto del dispositivo **waveaudio**. Si no indicamos inicio y final de la reproducción, el fichero se reproducirá entero. En muchos casos, si un comando **play** no viene precedido de un comando **open**, MCI intenta abrir el dispositivo directamente, lo que significa que las cinco cadenas del ejemplo anterior podrían simplificarse hasta:

```
play c:\waves\cascabel.wav
```

16.6.2. Posibilidades de audio de las cadenas de comandos

Básicamente, las acciones realizables en el campo del sonido son las siguientes:

- acceder a cualquier punto de un CD audio
- reproducir total o parcialmente un CD audio
- reproducir total o parcialmente ficheros de forma de onda
- grabar ficheros de forma de onda
- obtener información sobre el dispositivo de forma de onda
- reproducir total o parcialmente ficheros MIDI
- obtener información sobre el dispositivo de MIDI
- obtener información de retorno sobre el éxito de las operaciones

Reproducción parcial de ficheros
La reproducción parcial de ficheros es especialmente útil en el caso de los ficheros de onda. Supongamos que un programa que ha diseñado debe disparar varios efectos sonoros. La manera más directa de gestionarlos es abriendo y cerrando distintos ficheros, pero también puede incluir todos los efectos en un único fichero, utilizando un programa editor de audio. En este caso tan sólo serán necesarias una instrucción **open** al inicio de la aplicación y una instrucción **close** al final; para disparar los diferentes sonidos deberá utilizar instrucciones **play**

con diferentes valores de inicio y final. Esta forma de trabajo es más rápida y eficaz, siempre que el fichero con todos los sonidos no sea excesivamente grande.

Grabación de audio
Mediante comandos MCI, también es posible grabar ficheros de audio. Los comandos a utilizar podrían ser los siguientes:

```
open new type waveaudio alias grabacion
record grabacion
.........................
stop grabacion
save grabacion nombre.wav
close grabacion
```

En el caso del MIDI, no es lamentablemente posible realizar grabaciones (véase el apartado 16.10,"Limitaciones de los comandos MCI"). El porqué de esta restricción, deberíamos preguntárselo a Microsoft.

Obtención de información
Utilizando cadenas MCI, podemos conocer datos sobre el dispositivo de audio digital (resolución máxima, posibles frecuencias de muestreo, número de canales, etc.) así como sobre el dispositivo MIDI (síntesis utilizada, polifonía, número de voces, etc.). Estas consultas son de gran utilidad a la hora de escribir programas flexibles, que se adapten a las posibilidades del hardware instalado.

Otras posibilidades
El número total de comandos supera la cuarentena (muchos únicamente son aplicables a determinados dispositivos), y aunque tan sólo hemos arañado la superficie, esperamos que esta introducción le haya dado una idea sobre las posibilidades y el manejo básico de las cadenas MCI. Para obtener una información más detallada, podrá consultar la información que suele venir con el paquete de programación que vaya a utilizar. En cualquier caso, antes de escribir ninguna línea, lea detenidamente el siguiente apartado, pues lo indicado hasta el momento es absolutamente genérico y no forma parte de ningún lenguaje en particular.

16.6.3. Uso de las cadenas MCI desde Visual Basic

El interfaz de mensajes de comandos utiliza estructuras del lenguaje C, lo cual dificulta su uso desde otros que no sean el C y el C++. Por ello, en Visual Basic o los lenguajes de autor que soporten MCI (*Macromind Director*, *Asymetrix Toolbook*, etc.), el interfaz a utilizar es normalmente el de cadenas de comandos.

Desde *Lingo*[4] (el lenguaje de programación de *Director*), por ejemplo, la sintaxis sería la siguiente: `mci "cadena de comando".`

Desde Visual Basic, en lugar de utilizar la palabra clave *mci*, se debe utilizar una de las dos funciones C del API de Windows: *mciExecute()* o *mciSendString()*.

[4]Puede consultar *Director 4.0* y *Programación en Lingo*, ambos en la serie Guías Prácticas de Anaya Multimedia.

- La función *mciSendString()* es más completa por cuanto permite enviar un mensaje de retorno a la ventana indicada por el programador, con información sobre la ejecución de la orden, pero también requiere un mayor número de parámetros.
- La función *mciExecute*(), es más sencilla, ya que recibe un único parámetro con la cadena a ejecutar, aunque si se produjera algún error no le avisará; simplemente no realizará la instrucción.

Cuando desde Visual Basic se realiza una llamada a una función C, se debe indicar su prototipo en la sección **(declarations)** de un fichero tipo .bas. Si dispone de Visual Basic 4.0 o superior, puede utilizar la utilidad *API text Viewer* para copiar estos prototipos. En el caso de que este programando una aplicación de 32 bits en Windows 95[5], la línea a añadir para la función *mciExecute()* tendría el siguiente aspecto:

```
Declare Function mciExecute Lib "winmm.dll" (ByVal lpstrCommand
As String) As Long
```

Al realizar la llamada, simplemente tendrá que utilizar como argumento una cadena MCI.
Un ejemplo sencillo podría ser: `mciExecute("play c:\waves\cascabel.wav")`

Tenga en cuenta que si desea utilizar variables en la instrucción (nombres de ficheros, puntos de inicio o final, etc.) deberá formatear previamente la cadena. Estas líneas podrían escribirse en Visual Basic:

```
Dim cadena as String
Dim fichero as String
Dim inicio as Integer
Dim final as Integer

...fichero, inicio y final deberán tomar los valores
necesarios...

cadena = "play "+ fichero+ " from "+ CStr(inicio)+ " to "+
CStr(final)
mciExecute(cadena)
```

¡Observe los espacios que rodean los literales entrecomillados!

16.7. La interfaz de mensajes de comandos desde el lenguaje C

Lo que sigue es especialmente útil para programadores en C o C++, por lo que si no es éste su lenguaje, puede pasar directamente al siguiente apartado.
Cuando se utiliza el interfaz de cadenas, una llamada a las funciones *mciSendString()* o *mciExecute()*, fuerza al sistema a interpretar la cadena y desglosar cada uno de los parámetros que la componen. Con estos valores el sistema realiza internamente una nueva

[5] En el caso de que trabaje con Windows 3.1 o con Windows 95 a 16 bits, sería ligeramente diferentes.

llamada, esta vez a la función *mciSendCommand()*, que es la función utilizada directamente por el interfaz de mensajes.
La alternativa del interfaz de mensajes es por lo tanto más directa, aunque también más difícil de utilizar si no se programa en C, dado que utiliza estructuras propias de este lenguaje. Ambos interfaces ofrecen la misma funcionalidad y poseen, por así decirlo, el mismo "vocabulario", pero donde el interfaz de cadenas, utiliza cadenas de texto, el de mensajes, utiliza datos estructurados y constantes predefinidas en el fichero de cabecera *"mmsystem.h"*. Cada uno de los comandos, dispositivos o modificadores que pueden formar parte de una cadena MCI, tienen un valor numérico asociado, accesible desde una constante de C. Así por ejemplo, el comando **open** tiene asociada la constante MCI_OPEN, y el dispositivo **waveaudio**, la constante MCI_DEVTYPE_WAVEFORM (véase las tablas 16.1 y 16.2).
En un principio, el uso del interfaz de mensajes puede resultar menos inmediato pues requiere el conocimiento de decenas de nombres poco mnemotécnicos. A la larga, es más sencillo para el programador en C, pues le evita el uso exhaustivo de funciones de manejo de cadenas, a la hora de formatear los mensajes. Este interfaz ofrece asimismo al programador una mayor información sobre el estado de las operaciones, ya que permite el uso de funciones de tipo *callback*, que mandan automáticamente mensaje de seguimiento a una ventana de nuestra elección.

El prototipo de la función *mciSendCommand()* es el siguiente:

```
DWORD mciSendCommand(UINT wIDDispositivo, UINT wComando,
      DWORD dwParam1, DWORD dwParam2);
```

- *wIDDispositivo*, es un entero sin signo, que identifica al dispositivo.
- *wComando*, indica el comando a realizar, mediante una de las constante del tipo MCI_XXX, como las mostradas en la tabla 16.2.
- *dwParam1* es un entero largo en el que se colocan constantes de tipo bandera (flags) que indican diferentes formas de llevar a cabo la acción (esperar, avisar, etc.)
- *dwParam2* está definido como entero largo, pero contiene en realidad la dirección de una estructura C con información adicional para el comando (inicio, final, etc.). El tipo de estructura varía con el valor de *wComando*, ya que cada uno posee una estructura asociada.

El siguiente fragmento en C ilustra la apertura y reproducción de un fichero de audio. Este es un ejemplo más completo que los expuestos en 16.6, ya que incluye un control absoluto sobre los resultados de las operaciones.

```
// Abre y reproduce un fichero de audio en forma de onda.
// La función retorna tras el inicio de la reproducción.
// Devuelve 0L si todo ha ido bien, o un código de error.

DWORD playWaveFile(HWND hWndNotifica,
                       LPSTR lpstrNombreFichero)
// hWndNotifica es el Handle a la ventana que recibe la notificación
// lpstrNombreFichero es una cadena con el nombre del fichero
{
      UINT wIDDispositivo;
      DWORD dwRetorno;
      MCI_OPEN_PARMS mciOpenParams;
```

```
        MCI_PLAY_PARMS mciPlayParams;

        //Abrimos dispositivo, indicando tipo y fichero
        mciOpenParams.lpstrDeviceType = "waveaudio";
        mciOpenParams.lpstrElementName = lpstrNombreFichero;
        if (dwRetorno = mciSendCommand(0, MCI_OPEN,
              MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
              (DWORD)(LPVOID)&mciOpenParams))
              return (dwRetorno);    //Error en la apertura

        //Apertura correcta. Obtenemos el identificador
        wIDDispositivo = mciOpenParms.wDeviceID;

        //Se inicia la reproducción. Cuando ésta termine, la ventana
        //recibirá un mensaje MM_MCINOTIFY
        mciPlayParams.dwCallback = (DWORD) hWndNotify;
        if (dwRetorno = mciSendCommand(wIDDispositivo, MCI_PLAY,
              MCI_NOTIFY, (DWORD)(LPVOID)&mciPlayParams))
        {
              //Error en reproducción
              mciSendCommand(wIDDispositivo, MCI_CLOSE, 0, NULL);
              return (dwRetorno);
        }
        return (0L);
}
```

En este ejemplo aparecen dos tipos de estructuras, MCI_OPEN_PARMS y MCI_PLAY_PARMS, de las que sólo hemos utilizado tres campos de la primera. En la segunda, podríamos haber indicado los puntos de inicio y final de la reproducción. Aunque el fragmento parece bastante más complejo que las líneas escritas en ejemplos anteriores, los programadores en C sin duda admitirán que esta complejidad no es exagerada.

Resumen de su manejo

La forma de trabajo es siempre parecida, por lo que, una vez haya escrito un par de funciones, la codificación se volverá bastante rutinaria. A continuación indicamos algunas de las acciones más frecuentes y la forma de llevarlas a cabo:

1. Abrir un dispositivo con el comando MCI_OPEN y los parámetros necesarios (dispositivo y fichero) en la estructura MCI_OPEN_PARMS. En este caso, el parámetro *wIDDispositivo* se deja a cero.
2. Almacenar en una variable, el identificador del dispositivo abierto, retornado en el campo *wDeviceID* de MCI_OPEN_PARMS, en la llamada anterior.
3. A partir de la obtención de este identificador, lo podremos utilizar para cualquier otra operación, como por ejemplo:
4. Consultar el estado del dispositivo, con el comando MCI_STATUS y la estructura MCI_STATUS_PARMS.
5. Modificar algún parámetro del dispositivo, con el comando MCI_SET y la estructura MCI_SET_PARMS.
6. Reproducir un fichero con el comando MCI_PLAY y la estructura MCI_PLAY_PARMS.
7. Grabar un fragmento (sólo aplicable al dispositivo de onda) con el comando MCI_RECORD y la estructura MCI_RECORD_PARMS.

8. También se pueden utilizar los comandos MCI_PAUSE, MCI_RESUME y MCI_STOP, para parar temporalmente, reanudar, o parar definitivamente la reproducción o la grabación. En este caso no se utiliza ninguna estructura, dejando por lo tanto a cero el último parámetro de la función.
9. Salvar un fichero a disco con el comando MCI_SAVE y la estructura MCI_SAVE_PARMS.
10. Cerrar un dispositivo con el comando MCI_CLOSE. No se utiliza ninguna estructura, por lo que el último parámetro se deja a cero.

En caso de que se produzca algún error, se puede utilizar la función *mciGetError()*, que devuelve una cadena con el mensaje de error correspondiente al código numérico.

Una descripción exhaustiva de todas los comandos y estructuras involucradas en la interfaz de mensajes excede las pretensiones de este capítulo, por lo que si desease profundizar en esta línea, podrá consultar la información que acompaña a su compilador[6]. Conviene recordar por último que, aunque menos eficaz, el interfaz de cadenas sigue siendo accesible desde el lenguaje C, a través de las funciones *mciSendString()* o *mciExecute()*,

16.8. El MMControl de Visual Basic

16.8.1. Descripción

Visual Basic ofrece una alternativa adicional al uso de cadenas de comandos. Se trata del control multimedia *MMControl*, incluido en la versión 3.0 como *mci.vbx*, o como *mci32.ocx* a partir de la versión 4.0. Este control es accesible desde la barra de herramientas de Visual Basic. Si al abrir esta ventana, no le apareciera el icono marcado en la figura 16.2, posiblemente tenga que modificar la configuración del programa mediante la opción de menú **Tools | Custom Controls**.

Figura 16.2. Control multimedia en la barra de herramientas de Visual Basic

[6]En la ayuda "on line" de Visual C figuran decenas de páginas relacionadas con los comandos MCI.

Cuando incluya este control en una *Form*, el aspecto que mostrará por defecto será similar al de la figura 16.3. Sus botones, que siguen el paradigma de las pletinas de casetes, nos dan una idea intuitiva de sus posibilidades, entre las que se incluyen las de rebobinar, reproducir, parar, avance rápido, etc. La funcionalidad de este control es muy similar a la del reproductor de medios de Windows, aunque a diferencia de éste último, permite además *grabar* ficheros de audio de forma de onda.
Su elección frente a las cadenas MCI es aconsejable cuando en el programa a realizar desee incluir un interfaz de usuario interactiva para el control de los diferentes dispositivos multimedia.

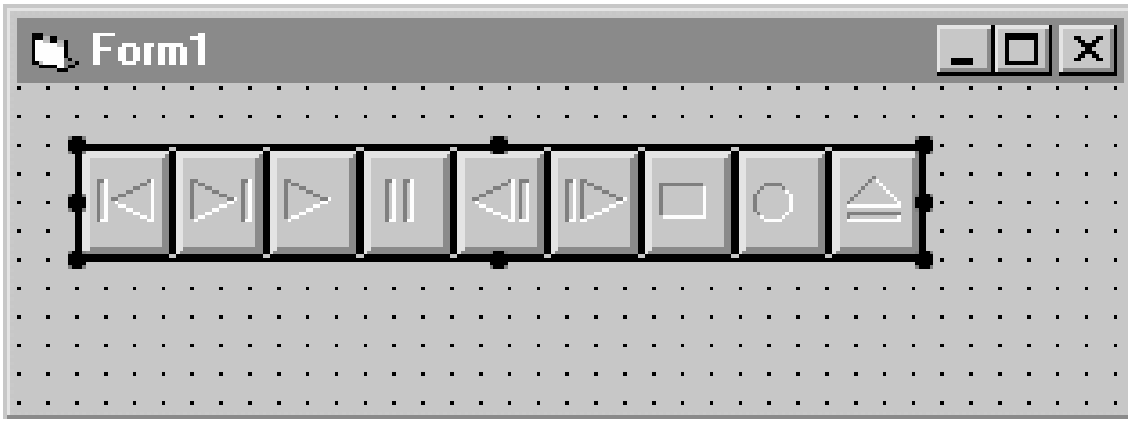

Figura 16.3. Aspecto del control multimedia

- Prev (previo)
- Next (siguiente)
- Play (reproducción)
- Pause (pausa)
- Back (atrás)
- Step (adelante)
- Stop (parada)
- Record (grabación)
- Eject (expulsar)

16.8.2. Manejo del control multimedia

Como sabemos, no todos los dispositivos multimedia permiten las mismas acciones, por lo que dependiendo del tipo de dispositivo al que se aplique el control, algunos botones se desactivarán automáticamente. ¿De que forma indicarle al control el tipo de dispositivo al que se deberá asociar?
Como todo elemento de este lenguaje, este control tiene unas propiedades, unos métodos, y es capaz de responder a determinados eventos. Si observásemos la lista de propiedades en la ayuda que presenta Visual Basic, descubriríamos inicialmente con pavor que posee más de 60. Pero tras un repaso más detallado nos percataremos de que la mayoría hacen referencia al aspecto del control (tamaño, posición, etc.) y son comunes a la mayoría de los objetos del

lenguaje. A continuación enumeramos algunas de las principales propiedades específicas de este control:

- Command
- DeviceType
- Filename
- TimeFormat
- From
- To

Como es frecuente en Visual Basic, muchas de ellas son asignables en la fase de diseño, desde la ventana de propiedades (véase figura 16.4), o bien desde el propio código, lo cual permite la modificación dinámica de muchos parámetros durante la ejecución del programa.

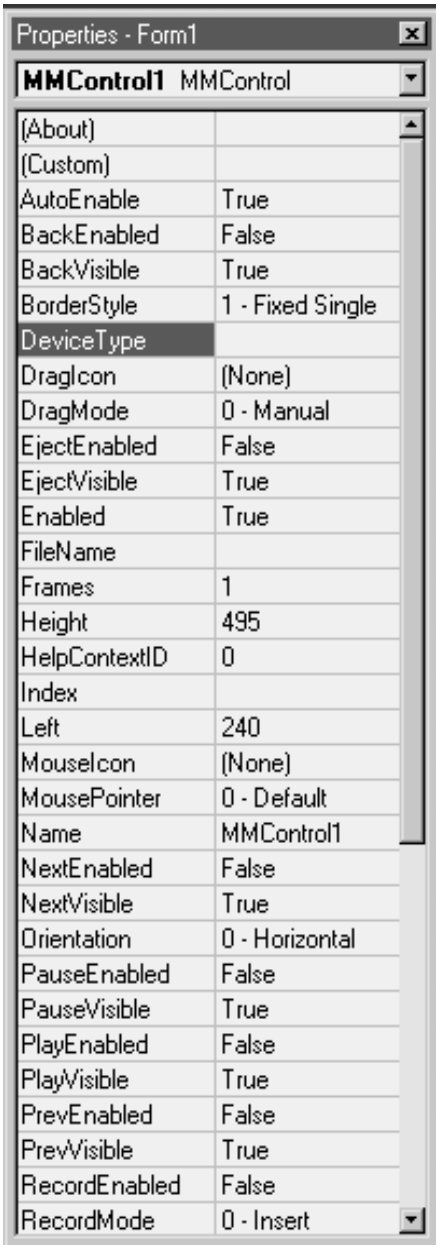

Figura 16.4. Ventana de propiedades del control multimedia

De la lista anterior, las propiedades más importantes son las tres primeras, que se corresponden directamente con los componentes de una cadena de comandos MCI básica. Supongamos que nuestro control recién añadido a la *Form*, se llama *MMControl1* (este es el nombre que VB le asigna por defecto). La instrucción *mciExecute("play c:\waves\cascabel.wav¨)* que mostramos como ejemplo en 16.6.3, podría llevarse a cabo con el control multimedia, escribiendo las siguientes líneas:

```
MMControl1.DeviceType = "WaveAudio"
MMControl1.Filename = "c:\waves\cascabel.wav"
MMControl1.Command = “open”
MMControl1.Command = “play”
```

16.8.3. Uso de varios controles simultáneos

Windows prohíbe la apertura simultánea de varios dispositivos idénticos, pero esta restricción no es aplicable a dispositivos diferentes, por lo que es perfectamente posible incluir y abrir varios controles multimedia en una misma *Form* de Visual Basic, si cada uno de ellos se asigna a un dispositivo diferente (audio digital, MIDI, vídeo, etc.). De esta forma el máximo número de controles accesibles simultáneamente se corresponderá con el número de dispositivos MCI instalados en el sistema.

16.8.4. Consideraciones adicionales

- La lista de posibles comandos es algo más reducida que en el interfaz de cadenas, ya que algunos han pasado a constituir directamente una propiedad del control. Para una descripción detallada de las propiedades y de los comandos puede consultar la ayuda interactiva de Visual Basic.
- Igual que sucedía con las cadenas MCI, la primera línea que indica el tipo de dispositivo, no será normalmente necesaria, si la extensión .wav figura como entrada en el apartado [mci extensions] del system.ini.
- Algo similar ocurre con *TimeFormat*.; cada dispositivo tiene un formato de tiempo por defecto, por lo que si no deseamos modificarlo, no será necesaria la especificación de esta propiedad.
- Tampoco será necesario especificar siempre todos los campos que componían una cadena MCI. Todas las propiedades se mantienen vigentes hasta que no se modifiquen, por lo que mientras trabaje con un mismo fichero, podrá escribir varios comandos correlativos, como por ejemplo:

```
MMControl1.Command = "play"
.......
MMControl1.Command = "pause"
.......
MMControl1.Command = "resume"
```

16.8.5. El interfaz de usuario

El control multimedia es especialmente interesante en el desarrollo de aplicaciones interactivas en las que se desee brindar al usuario cierto control sobre los dispositivos multimedia. De acuerdo con la filosofía de Visual Basic, la acción sobre un determinado botón genera un evento, y cada uno de estos eventos realiza una llamada a una subrutina que el programador puede completar como desee. Cada botón incluye cuatro subrutinas, que se activan automáticamente, dependiendo de la acción que sobre él se realice. Para el botón **Play**, por ejemplo, las subrutinas asociadas son: **PlayClick, PlayCompleted, PlayGotFocus** y **PlayLostFocus.**
Cada una de estas rutinas (en particular la de tipo *XXClick*, donde XX es el nombre de un botón de entre los indicados en la figura 16.3) produce la acción esperada sin necesidad de escribir una sola línea de código, por lo que para poner a punto un programa reproductor de CD audio totalmente operativo (con avance, retroceso, parada, etc.), ¡bastaría con escribir las tres líneas indicadas en la figura 16.5!

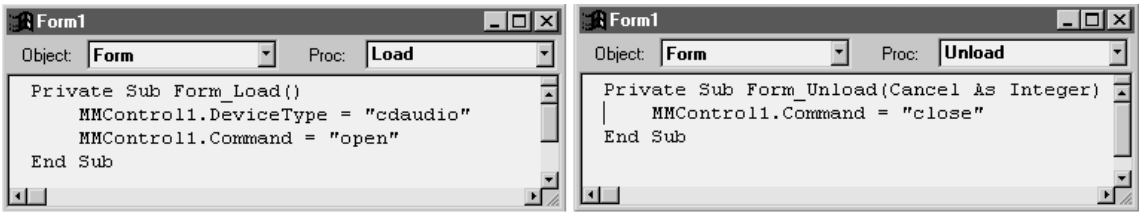

Figura 16.5. Un sencillo reproductor de CD audio

Si deseáramos personalizar un poco la aplicación podríamos añadir nuevo código en algunas de las subrutinas de los botones. Pero hay también ocasiones en las que el programador desea limitar la acción del usuario; para ello, es posible inhabilitar determinados botones o incluso hacerlos invisibles, modificando las propiedades, **ButtonEnabled** o **ButtonVisible**, donde *Button* es el nombre de cualquiera de los botones de la figura 16.3. Un ejemplo podría ser:

```
MMControl1.EjectEnabled = False
MMControl1.EjectVisible = False
```

También posible ocultar todo el control, mediante la instrucción:

```
MMControl1.Visible = False
```

Un control totalmente "invisible" ofrece de hecho la misma funcionalidad que el interfaz de cadenas de comandos, lo que plantea una pregunta obligada: ¿qué interfaz es más conveniente para el programador?
La respuesta debe ser matizada, ya que depende en gran parte de las preferencias de cada uno, aunque conviene tener en cuenta que el control utiliza bastantes más recursos del sistema que unas pocas instrucciones de cadenas, por lo que, si la aplicación no utiliza exhaustivamente las posibilidades MCI, la alternativa de cadenas será más eficaz.

16.9. La MCIWnd de Visual C++

Junto al uso de las cadenas de comandos o los mensajes de comandos, accesibles ambos a través de las funciones C indicadas anteriormente, a partir de la versión 2.0 de Visual C++ existe una nueva clase incluida en las MFC[7], la *MCIWnd* que ofrece una operatividad (que no un aspecto) similar al control multimedia de Visual Basic. Las consideraciones sobre su uso son similares a las indicadas para el control de Visual Basic, en el párrafo anterior. Si desea utiliza esta clase consulte la documentación que acompaña al paquete.

16.10. Limitaciones de los comandos MCI

El uso de los comandos MCI (desde cualquiera de los interfaces descritos), permite enriquecer notablemente cualquier aplicación para Windows con un coste bastante razonable. En el tema del sonido, que es el que nos ocupa, las posibilidades MCI son especialmente notables para los dispositivos de onda y el CD audio. En ambos casos, difícilmente necesitará mayores prestaciones. No se puede afirmar lo mismo con respecto al MIDI. Veamos cuales son las principales restricciones:

- **Imposibilidad de grabar mensajes MIDI**. Esta es una limitación importante, y en cierta forma, misteriosa. ¿Que impulsó a Microsoft a no contemplar la grabación de secuencias MIDI, cuando el dispositivo de ondas permite la grabación?
- **Imposibilidad de elección del puerto de salida.** Como se ha comentado anteriormente, incluso en sistemas con una sola tarjeta de sonido, es frecuente encontrar más de un puerto MIDI de salida. Esta limitación, aunque molesta, es totalmente coherente con la filosofía de las MCI que utilizan un único *driver* para cada tipo de dispositivo. En el caso del MIDI el driver utilizado es siempre el *Predeterminado* (o lo que es lo mismo, el *MIDI mapper*).
- **Imposibilidad de enviar mensajes individuales.** Sin necesidad de entrar en la creación de programas MIDI especializados (secuenciadores, etc.) que exceden las pretensiones de esta obra, la posibilidad de enviar mensajes MIDI independientes podría ser muy beneficiosa. Supongamos que una aplicación interactiva (o un juego) necesita cierto tipo de efectos sonoros (risas, gritos, portazos, lluvia, explosiones, etc.). Los cinco efectos indicados están todos incluidos en la especificación General MIDI, por lo que para activarlos bastaría con enviar tres mensajes (un cambio de programa, un note on y un note off) de dos y tres bytes respectivamente. Esta limitación obliga en su lugar a disparar ficheros de sonido de varios centenares de Kb.

Por ello, creemos oportuno tratar la programación MIDI de bajo nivel con funciones del API de Windows, en el próximo capítulo, en el que escribiremos un programa MIDI completo.

[7] *Las Microsoft Foundation Classes*, son las clases C++ que incorpora Microsoft Visual C++, y que constituyen la piedra angular de la programación en este lenguaje.