17. Programación MIDI de bajo nivel

17.1. Introducción

Al final del capítulo anterior exponíamos las limitaciones de los comandos MCI en lo que al MIDI se refiere. En este capítulo estudiaremos unas cuantas funciones incluidas en el API de Windows (tanto en la versión 3.1 como en Windows 95), que permiten superar esas restricciones. Verá como a pesar de que esta información queda excluida de la mayoría de manuales de programación multimedia, no implica dificultades muy superiores.
Tras unos apartados introductorios y más teóricos, realizaremos un pequeño programa que a pesar de su sencillez, pondrá en práctica la mayoría de conceptos y mensajes MIDI tratados en esta obra.
La programación con el API de Windows termina inevitablemente con llamadas a funciones C incluidas en el sistema. Por ello, el lenguaje C sería el más indicado para completar el código. Sin embargo, siguiendo la filosofía expuesta en el anterior capítulo, creemos que la exposición de los ejemplos en Visual Basic puede ser de utilidad para un mayor número de programadores, con la certeza de que para quien lo desee, la traducción al lenguaje C (o C++) no revestirá ninguna dificultad.
Asimismo, bastantes de los conceptos explicados podrán ser de utilidad para programadores de otras plataformas (aunque el esfuerzo requerido para llevar cualquier idea a la práctica será bastante superior).

17.2. El API multimedia de Windows

En Windows, las siglas API (*Application Programming Interface*) hacen referencia a las librerías de funciones utilizadas por el sistema, documentadas y utilizables desde cualquier lenguaje que permita realizar llamadas externas a funciones C.
Uno de estos API es el de multimedia, incluido con Windows desde la versión 3.1. Para trabajar con él es necesario incluir el fichero de cabecera "mmsystem.h"[1].
Sus funciones permiten trabajar con varios tipos de dispositivos, con un control muy superior al ofrecido por los comandos MCI. Los dispositivos accesibles son el de audio digital (o de ficheros de onda), el MIDI, el joystick, el reloj de tiempo y, desde la versión 95, el video digital y el mezclador de audio.
Un estudio detallado de todas estas funciones requeriría un espacio superior al de toda esta obra. Microsoft dispone de varios voluminosos manuales de referencia[2], y también es posible acceder a casi toda la información desde la biblioteca de ayuda interactiva de Visual C++.

---

[1] En *Visual Basic* no se utilizan ficheros de cabecera, por lo que será necesario incluir los prototipos de las funciones y la definición de las estructuras que vayamos a utilizar, con la ayuda del *API Text Viewer*.

[2] *Microsoft Windows Multimedia Programmer´s Reference* y *Microsoft Windows Multimedia Programmer´s Workbook*, de Microsoft Press.

17.2.1. Dispositivos accesibles desde el API multimedia

En este capítulo nos limitaremos a estudiar el MIDI y, aunque no lo haremos en su totalidad, llegaremos a escribir un programa completo y operativo. Antes de comenzar, daremos algunas indicaciones sobre los restantes dispositivos.

- **El dispositivo de ficheros de onda**. En este caso, el uso de las funciones de bajo nivel sólo será necesario cuando se desee un control sobre los ficheros de onda, similar al que puede ofrecer un editor gráfico de audio, como los estudiados en el capítulo 5**,**"Edición de sonido por ordenador". Para aplicaciones más sencillas, los comandos MCI son normalmente suficientes, aunque algo más lentos.
- **El joystick** es un dispositivo simple y también lo es (y mucho) su programación, por lo que si siente tentaciones y dispone un entorno de programación como *Visual C++*, no dude en consultar la ayuda interactiva, ¡verá lo sencillo que resulta! Básicamente, la programación del joystick consiste en poco más que abrir el dispositivo, y consultar periódicamente sus coordenadas y el estado de sus botones. Lo que sí que puede resultar muy complicado, es ¡que hacer después con esta información!
- **El timer** (reloj) multimedia, es imprescindible para toda programación que requiera un control del tiempo preciso (como un secuenciador MIDI). Visual Basic incorpora también un control *timer*, pero su resolución es de aproximadamente 50 ms (aunque en sus propiedades, el intervalo se puede ajustar aparentemente en milisegundos), lo cual es insuficiente para muchas aplicaciones rigurosas. A título informativo, diremos también que el reloj por defecto de *Visual C++* ofrece esta misma resolución. Si deseará programar un secuenciador, o sincronizar eventos MIDI con un fichero de video, necesitará controlar el *timer* especializado. En nuestro ejemplo no lo utilizaremos (no escribiremos un secuenciador).
- **El video digital**. Lo comentado para el dispositivo de ficheros de onda es aplicable también a este dispositivo. Si no pretende escribir una nueva versión de *Adobe Premiere* , las prestaciones de los comandos MCI serán más que suficientes.
- **El mezclador** ofrece un control directo sobre cualquier dispositivo de tipo *mixer* instalado por el software de una tarjeta de sonido. Esta posibilidad es nueva de Windows 95, y no reviste mayores complicaciones que el dispositivo MIDI que trataremos a continuación.

17.3. Introducción a las funciones MIDI de bajo nivel

A grandes rasgos, las posibilidades del API multimedia no difieren demasiado de un dispositivo a otro, por lo que, aunque nos centraremos en las funciones y estructuras utilizadas con los dispositivos MIDI, gran parte de la información es aplicable a otros dispositivos, cambiando unas pocas letras en los nombres de las funciones y las estructuras.
Las funciones disponibles pueden agruparse en los siguientes temas o familias:

- petición de información sobre dispositivos disponibles
- apertura y cierre de dispositivos
- envío de eventos individuales
- recepción de eventos individuales

- creación y manejo de bloques de memoria con formato
- gestión de errores
- grabación

Una aplicación MIDI profesional posiblemente tenga que utilizar funciones de todos estos grupos, pero en nuestro ejemplo nos limitaremos a detectar los dispositivos, abrir uno de ellos y enviarle mensajes individuales, por lo que tan sólo utilizaremos funciones de los tres primeros grupos.

17.4. Petición de información sobre los dispositivos disponibles

A diferencia de los comandos MCI, mediante las funciones de bajo nivel es posible acceder a cualquiera de los dispositivos o puertos disponibles en el sistema. Para detectar estos dispositivos accesibles se utilizan dos funciones de manejo muy simple:

UINT midiInGetNumDevs(void)
UINT midiOutGetNumDevs(void)

que devuelven respectivamente el número de dispositivos MIDI disponibles, de entrada y de salida. Entre los dispositivos de salida se incluye también el *MIDI Mapper*.

17.5. Prestaciones de los dispositivos disponibles

Una vez conocido el número de dispositivos instalados en el sistema, es posible determinar las prestaciones de cada uno de ellos. Para ello se utilizan dos funciones y dos estructuras en las que se almacena la información solicitada.
Las dos estructuras se definen (en C) como:

```
typedef struct {
    WORD      wMid;               //identificador fabricante
    WORD      wPid;               //identificador producto
    MMVERSION vDriverVersion;     //versión del driver
    CHAR      szPname[MAXPNAMELEN];//nombre del producto
    DWORD     dwSupport;          //reservado
} MIDIINCAPS;

typedef struct {
    WORD      wMid;               //identificador fabricante
    WORD      wPid;               //identificador producto
    MMVERSION vDriverVersion;     //versión del driver
    CHAR      szPname[MAXPNAMELEN];//nombre del producto
    WORD      wTechnology;        //tecnología (FM,Wavetable,etc.)
    WORD      wVoices;            //num. voces (multitímbrica)
    WORD      wNotes;             //num. notas (polifonía)
    WORD      wChannelMask;       //canales a los que responde
    DWORD     dwSupport;          //posibilidades adicionales
} MIDIOUTCAPS;
```

- *wTechnology* y *dwSupport* utilizan constantes definidas también en "mmsystem.h".
- En *wChannelMask*, cada bit representa uno de los 16 posibles canales MIDI.

Para definir éstas en Visual Basic, basta con consultar y copiar la información que ofrece la utilidad *API Text Viewer*, incluida con la versión 4.0. Si dispone de una versión anterior, puede copiar las definiciones del código ejemplo, incluidas en el apartado 17.10. De toda la información contenida en estas estructuras, en el ejemplo utilizaremos únicamente el campo *szPname* para obtener el nombre del dispositivo.

Las dos funciones utilizadas para acceder a esta información son:

- `ULONG midiInGetDevCaps(UINT uDeviceID, LPMIDIINCAPS lpMidiInCaps, UINT cbMidiInCaps);`
- `ULONG midiOutGetDevCaps(UINT uDeviceID, LPMIDIOUTCAPS lpMidiOutCaps, UINT cbMidiOutCaps);`

En ambos casos:

- Las funciones devuelven cero si no se ha producido ningún error.
- *uDeviceID* es el identificador del dispositivo a consultar, y varía entre cero y el número total de dispositivos menos uno.
- La información se almacena en cada una de las estructuras, pasadas por referencia.
- El último parámetro corresponde al tamaño (en bytes) de esta estructura.

Si quisiéramos acceder a todos los dispositivos MIDI instalados, utilizaríamos un bucle desde cero hasta el valor obtenido con las funciones del apartado anterior menos uno (véase la subrutina *PreparaMidi()* en el apartado 17.11).

17.6. Apertura y cierre de dispositivos

Antes de utilizar cualquier dispositivo de entrada o de salida, es necesario abrirlo utilizando las funciones:

- `ULONG midiInOpen(LPHMIDIIN lphMidiIn, UINT uDeviceID, DWORD dwCallback, DWORD dwCallbackInstance, DWORD dwFlags);`
- `ULONG midiOutOpen(LPHMIDIOUT lphmo, UINT uDeviceID, DWORD dwCallback, DWORD dwCallbackInstance, DWORD dwFlags);`

En ambos casos:

- Las funciones retornan cero si no se han producido errores
- Las funciones cargan el *handle* del dispositivo abierto, en el segundo parámetro. A partir de este momento, ésta será la variable con la cual accederemos al dispositivo.
- El segundo parámetro es el índice correspondiente al dispositivo que deseamos abrir.
- El tercer parámetro es el *handle* de la ventana que recibirá los mensajes.
- El cuarto parámetro no se utiliza en Visual Basic, por lo que pondremos un cero.

- El quinto parámetro indica la forma en que queremos recibir los mensajes de retorno a la ventana. En Visual Basic es conveniente poner &H10000

La apertura dará error si el dispositivo ya estuviera abierto previamente (si lo estuviera utilizando otra aplicación, o la aplicación anterior hubiese olvidado cerrarlo).

Para cerrar un dispositivo se utilizan simplemente las dos funciones.

- `ULONG midiInClose(HMIDIIN hmi);`
- `ULONG midiOutClose(HMIDIOUT hmo);`

pasándoles el *handle* del dispositivo a cerrar.

17.7. Envío de mensajes individuales

Una vez abierto un dispositivo de salida, una de las operaciones más sencillas a realizar, pero a la vez más operativas, consiste en enviar mensajes MIDI. El envío de cada mensaje (un *Note On*, un *Note Off*, un cambio de programa, etc.) requerirá una nueva llamada a la función

```
ULONG midiOutShortMsg(HMIDIOUT hmo, DWORD dwMsg);
```

- Si el envío se realizó con éxito, la función retorna un cero.
- *dwMsg* contendrá el mensaje que deseemos mandar. Dado que esta variable tiene cuatro bytes, podremos utilizar esta función para mandar cualquier tipo de mensaje MIDI (todos tienen dos o tres bytes), salvo los de sistema exclusivo. Los mensajes se deben formatear de modo que el primer byte del mensaje ocupe el byte menos significativo de la variable. Esto se podría conseguir en C utilizando uniones u operadores de bits. En Visual Basic habrá que hacer un poco de "aritmética" binaria.

17.8. Formateo de mensajes en Visual Basic

Note On

Supongamos que deseamos enviar un mensaje de *Note On*. Recuerde que este mensaje está formado por tres bytes, donde el primero tiene un valor hexadecimal de 9N, siendo N el número de canal (véase 8.4), mientras que el segundo indicará la nota y el tercero la velocidad. Una rutina de Visual Basic encargada de enviar este tipo de mensajes, podría ser como la de la figura 17.1.

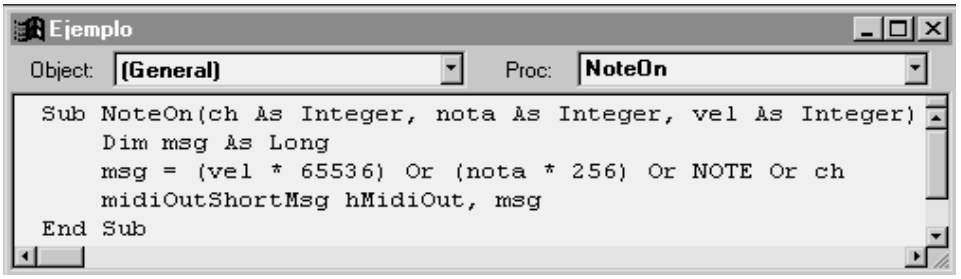

Figura 17.1. Rutina *NoteOn* en Visual Basic

En este ejemplo hemos optado por utilizar una variable global *hMidiOut*, para almacenar el *handle* al dispositivo de salida elegido. La rutina recibe tres parámetros (canal, nota y velocidad) y los agrupa, calculando la posición correcta de cada byte. Tenga en cuenta que al multiplicar por distintas potencias de dos (256 y 65536), estamos desplazando de 8 y 16 bits respectivamente cada uno de los parámetros recibidos. A continuación, al combinarlos con el operador lógico OR, cada uno de estos valores ocupará su byte correspondiente. Si no lo ve directamente, realice algunos cálculos binarios para convencerse.
A partir de aquí, para enviar cualquier otro mensaje de tres bytes (cambio de control, etc.), bastaría con cambiar el valor de la constante hexadecimal que indica el tipo de mensaje. Veamos un ejemplo más con un mensaje de dos bytes.

Cambio de programa
Este mensaje utiliza sólo dos bytes. El código hexadecimal asociado es el CN (donde N indica el canal deseado), y el segundo byte corresponde al número de programa MIDI elegido. Por consiguiente, la rutina *PChange* podría ser como la de la figura 17.2.

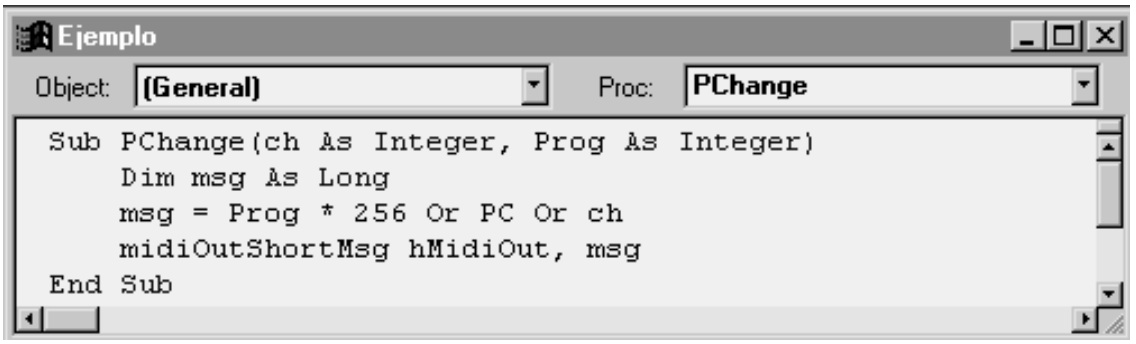

Figura 17.2. Rutina *Program Change* en Visual Basic

A partir de aquí, está ya en posesión de toda la información necesaria para realizar el programa de ejemplo.

17.9. Descripción del programa ejemplo

El programa que vamos a realizar, cuyo aspecto se puede apreciar en la figura 17.3, nos va a permitir:

1. Elegir un dispositivo MIDI de salida de entre todos los instalados en el sistema.
2. Elegir un canal MIDI de salida.
3. Disparar notas y acordes desplazando el ratón sobre la ventana.
4. Enviar cambios de volumen y de afinación (*pitch bend*) continuos, también con el ratón.
5. Modificar el programa elegido (instrumento).

En la lista descriptiva del interfaz gráfico, hemos indicado la función de cada elemento junto con los controles de Visual Basic utilizados. Las propiedades de todos estos controles son las predeterminadas, por lo que no es necesario especificar mayores detalles. Si acaso, recordar que para que varios botones de opción funcionen de forma exclusiva, deben estar

superpuestos a un control. En este caso hemos utilizado el control *Frame*. Por último, si programa en C pero no conoce Visual Basic, ¡no puede imaginarse lo sencillo que resulta hacer aplicaciones con este lenguaje!

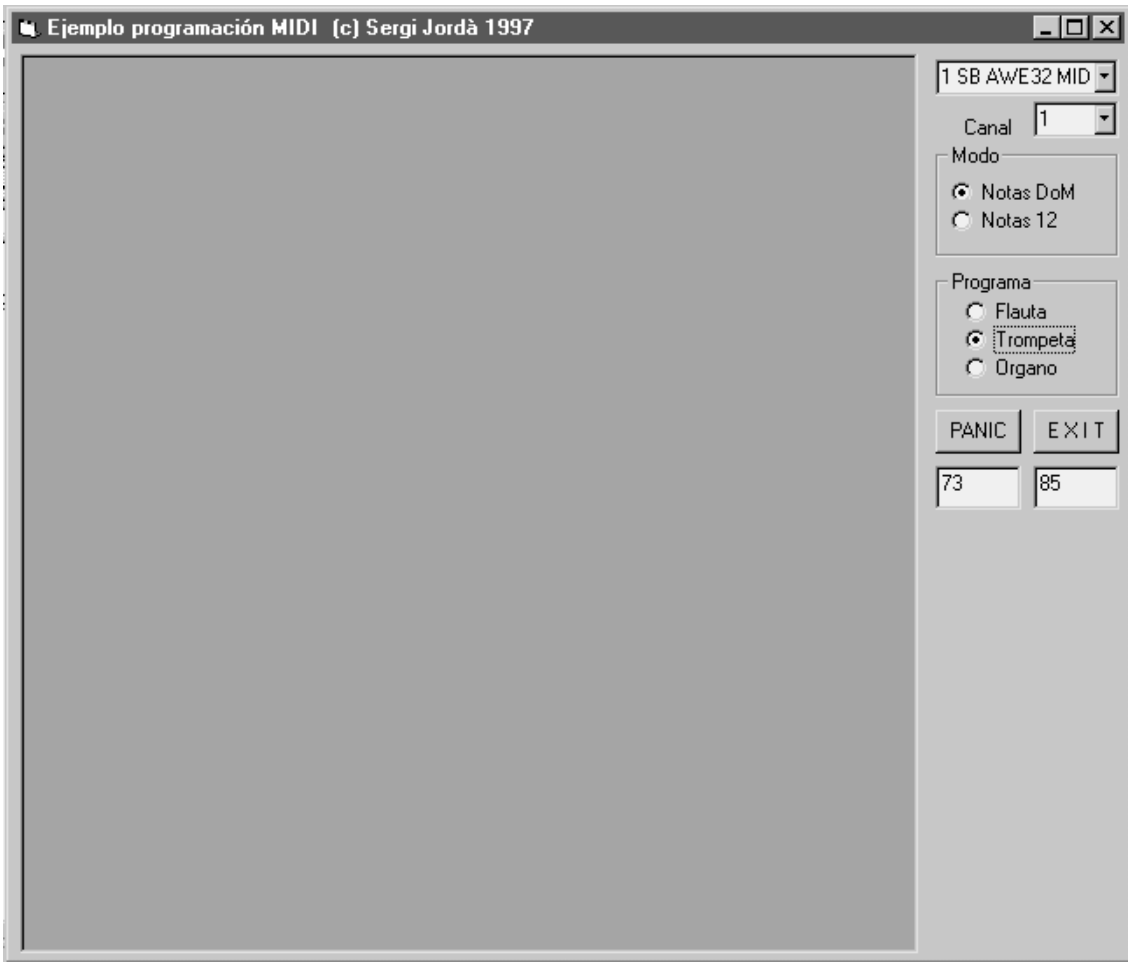

Figura 17.3. Aspecto del programa ejemplo

A. Lista desplegable (control *ComboBox*) con todos los dispositivos disponibles
B. Lista desplegable (control *ComboBox*) con los 16 canales MIDI
C. Selección de tonalidad (2 controles *OptionButton*) a elegir entre todas las notas o Do Mayor (teclas blancas del piano)
D. Selección de instrumento a elegir entre tres posibles (3 controles *OptionButton*)
E. Salida del programa *(control CommandButton*)
F. Apagar todas las notas (control *CommandButton*)
G. Visualización de la posición del ratón (2 controles *TextBox*)
H. "Pizarra" o zona sensible al movimiento del ratón (control *PictureBox*)

17.10. Definición de constantes

Este proyecto Visual Basic constará de dos ficheros, *ejemplo.frm* (el principal de tipo Form) y *constants.bas*. En este módulo auxiliar incluiremos únicamente las declaraciones de funciones y estructuras C del API de Windows, utilizadas en el programa, por lo que el

fichero contendrá únicamente la sección **(General) (Declarations**), que mostramos a continuación.

```
'Declaraciones funciones MIDI (Versión 32 bits)
Declare Function midiOutClose Lib "winmm.dll" (ByVal hMidiOut As Long)
As Long
Declare Function midiOutOpen Lib "winmm.dll" (lphMidiOut As Long,
ByVal uDeviceID As Long, ByVal dwCallback As Long, ByVal dwInstance As
Long, ByVal dwFlags As Long) As Long
Declare Function midiOutShortMsg Lib "winmm.dll" (ByVal hMidiOut As
Long, ByVal dwMsg As Long) As Long
Declare Function mciExecute Lib "winmm.dll" (ByVal lpstrCommand As
String) As Long
Declare Function midiOutGetNumDevs Lib "winmm" () As Integer
Declare    Function    midiOutGetDevCaps    Lib    "winmm.dll"    Alias
"midiOutGetDevCapsA" (ByVal uDeviceID As Long, lpCaps As MIDIOUTCAPS,
ByVal uSize As Long) As Long

Const MAXPNAMELEN = 32  ' max product name length (including NULL)

Type MIDIOUTCAPS
    wMid As Integer                 '  manufacturer ID
    wPid As Integer                 '  product ID
    vDriverVersion As Integer       '  version of the driver
    szPname As String * MAXPNAMELEN '  product name string)
    wTechnology As Integer          '  type of device
    wVoices As Integer              '  # of voices
    wNotes As Integer               '  max # of notes
    wChannelMask As Integer         '  channels used
    dwSupport As Long               '  functionality supported
End Type
```

A continuación incluimos el código correspondiente a la declaración de variables globales y constantes en la sección **(General) (Declarations**) del fichero principal (ejemplo.frm).

```
'código correspondiente a (General)(Declarations)
Option Explicit

'variables globales
'==================
Dim nota As Integer     'nota MIDI activa
Dim canal As Integer    'canal MIDI seleccionado
Dim modo As Integer     'modo seleccionado (7 ó 12 notas)
Dim hMidiOut As Long    'handle al puerto MIDI de salida abierto

'constantes
'==========
'modos de funcionamiento
Const Notas7 = 0    'modo 7 notas (Do Mayor)
Const Notas12 = 1   'modo 12 notas “libre"
'cambios de programa: los valores de estas constantes
'vienen dados por el General MIDI (véase tabla 7.1)
Const Flauta = 73
Const Trompeta = 56
Const Organo = 16
'tipos de mensajes MIDI (en hexadecimal)
Const NOTE = &H90
```

```
Const PC = &HC0
Const CC = &HB0
Const PB = &HE0
'otras constantes
Const Tamano = 4    'multiplica tamaño pizarra
Const BOTIZQ = 1    'pulsado botón izquierdo ratón
Const BOTDER = 2    'pulsado botón derecho ratón
Const NOBOTN = 0    'ningún botón pulsado
```

17.11. Inicio de la aplicación

Al iniciarse, el programa:

1. Ajusta el tamaño de la zona sensible.
2. Consulta todos los dispositivos de salida disponibles y crea una lista desplegable, para que el usuario pueda elegir uno de ellos.

Por ello, llamamos a dos rutinas en la carga de la ventana principal (**Form Load**).

```
Private Sub Form_Load()
    PreparaPizarra
    PreparaMIDI
End Sub
```

PreparaPizarra() ajusta el tamaño del *PictureBox* que utilizaremos para capturar la posición del ratón. Al asignarle un tamaño múltiplo de 127, se simplifica el cambio de escala.

```
Public Sub PreparaPizarra()
    Pizarra.Left = 5
    Pizarra.Top = 5
    Pizarra.Height = Pizarra.Top + (127 * Tamano)
    Pizarra.Width = Pizarra.Left + (127 * Tamano)
End Sub
```

*PreparaMIDI()*comprueba el número de dispositivos MIDI de salida disponibles, consulta las propiedades de cada uno, y guarda sus nombres en una lista de tipo *ComboBox*.

```
Public Sub PreparaMIDI()
    Dim PrestacionesMO As MIDIOUTCAPS
    Dim i As Integer
    Dim n As Integer

    CombDisp.ListIndex = -1
    CombChan.ListIndex = 0
    n = midiOutGetNumDevs()

    For i = 0 To n - 1
        midiOutGetDevCaps i, PrestacionesMO, 50
        CombDisp.AddItem CStr(i + 1) + Strip(PrestacionesMO.szPname)
    Next i

    CombDisp.ListIndex = 0
```

```
End Sub
```

Strip() es una pequeña rutina que se utiliza para adaptar las cadenas C a las que necesita la lista.

```
Public Function Strip(cadena As String) As String
    Dim i As Integer
    'apaño para convertir las cadenas, de forma que sean
    'visualizables en el combo
    i = InStr(3, cadena, Chr$(0))
    Strip = Mid$(cadena, 3, 16)
End Function
```

Tras esta inicialización el programa comienza ya a funcionar.

17.12. Funcionamiento y prestaciones del programa

Una vez iniciada la ejecución, el usuario podrá en cualquier momento:

1. **Seleccionar un nuevo puerto de salida MIDI**, mediante la lista desplegable. Esta acción cerrará el puerto anterior, abrirá el nuevo puerto y almacenará el nuevo valor del *handle* en una variable global.
2. **Seleccionar un nuevo canal MIDI**, mediante la lista desplegable con opciones del 1 al 16. Esta acción tan sólo modificará la variable global *canal*, que se utiliza en todas las funciones de envío de mensajes MIDI. (estas dos selecciones se representan en la figura 17.4).
3. **Seleccionar la tonalidad**. Las dos posibles elecciones son Do Mayor o todas las doce notas (no hay tonalidad). Hemos incluido esta posibilidad para mostrar un muy sencillo ejemplo de "instrumento inteligente", ya que en modo Do Mayor, el programa deberá hacer correcciones permanentes para que las notas emitidas al desplazar y hacer clic sobre el ratón, sean las correctas.
4. **Aplicar un nuevo programa MIDI**. Aunque una forma "seria" de cambiar de programa podría mostrar una lista desplegable o una nueva ventana con los nombres de todos los 128 instrumentos General MIDI, hemos simplificado la opción dejando elegir entre tan solo tres programas, mediante botones de opción. La implementación a nivel MIDI es la misma en ambos casos y consiste simplemente en enviar un mensaje de cambio de programa con el número correspondiente a la opción elegida.
5. **Cancelar todas las notas**. En alguna ocasión, puede suceder que alguna nota se quede colgada porque se perdiera un mensaje de *Note Off*. Para esta emergencia, todos los secuenciadores incluyen un botón denominado frecuentemente *Panic*, que manda un mensaje de *All Notes Off*.
6. **Activar notas y controles con el ratón**. En este caso, la explicación es un poco más larga. Al hacer un clic sobre la pizarra, se activa una nota MIDI. La altura de esta nota, depende de la posición horizontal del ratón (izquierda-grave, derecha-aguda), mientras que su velocidad depende de la posición vertical (arriba-fuerte). Al soltar el botón del ratón la nota se desactiva. Si la tonalidad elegida es Do Mayor, al hacer clic con el botón *derecho*, en lugar de una sola nota, se dispara el acorde correspondiente a esta nota. Además, mientras la nota se mantiene sonando (es decir mientras el ratón está

apretado), el desplazamiento del ratón envía controles de volumen (arriba-fuerte) y de *pitch bend* (derecha-agudo), de forma continua.

Esta aplicación no es un alarde de musicalidad, pero permite estudiar bastantes puntos de la programación MIDI, así como aspectos muy triviales de la composición algorítmica. A continuación, comentaremos con mayor detalle cada uno de los bloques junto con su correspondiente código.

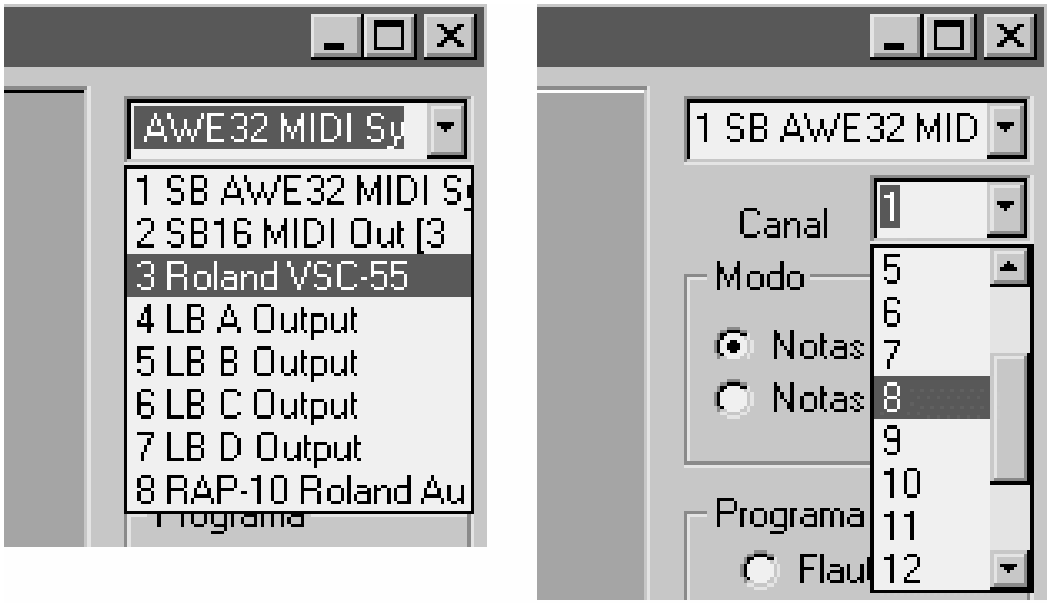

Figura 17.4. Listas de selección de dispositivo y de canal

17.13. Estudio del código

17.13.1. Selección de un nuevo puerto de salida MIDI

Esta rutina se activa automáticamente cuando hacemos clic sobre un elemento de la lista de dispositivos. Se obtiene el valor ordinal seleccionado y se manda a la rutina *MidiOpen()*, que se encarga de cerrar el puerto ya abierto, y abrir el nuevo. *MidiOpen()* hace una llamada a la función del API *midiOutOpen()* (véase 17.6). Al terminar la llamada, la variable global *hMidiOut*, posee ya un nuevo valor.

```
'código asociado al clic sobre la lista de dispositivos
Private Sub CombDisp_Click()
    If CombDisp.ListIndex >= 0 Then
        MidiOpen (CombDisp.ListIndex)
    End If
End Sub
Sub MidiOpen(i As Integer)
    'cierra el puerto que estuviera abierto
    midiOutClose hMidiOut
    'abre un nuevo puerto
    midiOutOpen hMidiOut, i, Ejemplo.hWnd, 0&, &H10000
End Sub
```

17.13.2. Selección del canal MIDI

Este código requiere pocas explicaciones. La variable *canal* es global, y la utilizan todas las funciones encargadas de enviar mensajes MIDI al puerto de salida.

```
Private Sub CombChan_Click()
    canal = CombChan.ListIndex
End Sub
```

17.13.3. Selección de tonalidad

La selección de una tonalidad no reviste ninguna complicación en si, ya que tan solo se debe guardar una variable global, con el valor de la opción elegida. Lo que, como veremos más adelante, no va a resultar tan sencillo será controlar la altura de las notas en función de esta tonalidad.

```
Private Sub Opcion_Click(Index As Integer)
    modo = Index
End Sub
```

17.13.4. Cambio de programa

Tampoco esta rutina presenta complicación alguna, aunque supone la primera llamada a una de nuestras funciones de mensajes MIDI (véase 17.12). las tres constantes, están definidas en la sección **(General) (Declaratives)** y sus valores corresponden con los del estándar General MIDI (tabla 7.1).

```
Private Sub OpPrograma_Click(Index As Integer)
    Dim programa As Integer
    Select Case Index
        Case 0: programa = Flauta
        Case 1: programa = Trompeta
        Case 2: programa = Organo
    End Select
    PChange canal, programa
End Sub
```

17.13.5. Anular todas las notas

Como indicábamos en el anterior apartado, todo programa MIDI que se precie debería, por pequeño que fuera, incluir esta opción de emergencia. Para ello hemos incluido un botón de comando que envía un mensaje de *allnotesoff*. En caso de que su tarjeta no respondiera correctamente a este mensaje, mostramos también la codificación alternativa, lenta pero segura, que consiste en mandar mediante dos bucles, mensajes de *Note Off* a todas las notas de todos los canales.

```
Private Sub ComPanic_Click()
    AllNotesOff
End Sub
```

```
Public Sub AllNotesOff()
    'desactiva todas las notas
    CChange 0, 123, 127
End Sub

Public Sub AllNotesOffAlternativo()
    'desactiva todas las notas alternativa lenta
    Dim c As Integer
    Dim n As Integer
    For c = 0 To 15
        For n = 0 To 127
            NoteOff c, n
        Next n
    Next c
End Sub
```

17.13.6. Envío de mensajes MIDI

Este bloque constituye el corazón de la aplicación y requiere una explicación mucho más detallada. Los eventos del ratón que capta el control *Pizarra* (de tipo *PictureBox*) son *MouseDown*, *MouseUp* y *MouseMove*. En cada uno de ellos, extrae las coordenadas *x* e *y* del ratón, y las divide por la constante *Tamano*, para obtener valores comprendidos entre 0 y 127, que son los que nos interesan para el MIDI.

Mouse Down - Note ON

Como indicábamos, al disparar notas, la posición *x* determinará la altura de la nota y la posición *y* su velocidad MIDI. Dado que parece razonable que las velocidades superiores correspondan a la parte superior de la ventana, y que sin embargo Visual Basic coloca el origen de coordenadas en el extremo superior, realizamos un sencillo cambio de coordenadas, mediante la operación *velocity = 127 - y*.

Si la "tonalidad" seleccionada fuese de 12 notas (es decir, no hay tonalidad), envía un mensaje de *NoteOn* directamente, pasándole el canal, la nota y la velocidad.

Si la tonalidad fuese en cambio la de Do Mayor, antes de enviar la nota, realiza una corrección mediante la función *CorrecNota()* que recibe la nota original y devuelve la nota corregida.

En este caso y si el botón apretado fuese el derecho, no enviará una sola nota, sino que tocará el acorde correspondiente. Esta prestación se incluye como propina y se deja a la curiosidad del lector, ya que en ella intervienen algunos conceptos muy básicos de armonía que no discutiremos aquí.

Recordemos que la variable *nota* es global. Esto nos permite guardar el valor de la nota activa, para después anularla con un mensaje de *NoteOff*.

```
Private Sub Pizarra_MouseDown(Button As Integer, Shift As Integer, x
As Single, Y As Single)
    Dim velocity As Integer
    velocity = 127 - Int(Y / Tamano) 'cambio de coordenadas
    If modo = Notas12 Then
        nota = Int(x / Tamano)
        NoteOn canal, nota, velocity
    ElseIf modo = Notas7 Then
```

```
        nota = CorrecNota(Int(x / Tamano))
        If Button = BOTDER Then
            Acorde canal, nota, velocity
        Else
            NoteOn canal, nota, velocity
        End If
    End If
End Sub
```

La función *CorrecNota()* corrige la nota recibida como argumento y la adapta a la nota de la escala de Do Mayor inmediatamente anterior. Para ello, calcula el resto de dividir por 12 la nota original. Esto nos permite saber el nombre de la nota (0:do, 1:do#, 2:re, 3:re#….), independientemente de su octava, y es un truco que se utiliza bastante en música algorítmica. Si la nota no perteneciera a la escala de Do Mayor, simplemente se le resta uno.

```
Public Function CorrecNota(nota As Integer) As Integer
    Dim grado As Integer
    grado = nota Mod 12
    If grado = 1 Or grado = 3 Or grado = 6 Or grado = 8 Or grado = 10
Then
        CorrecNota = nota - 1
    Else
        CorrecNota = nota
    End If
End Function
```

La función *Acorde()* se deja a la curiosidad del lector. Cuando la velocidad no es nula, calcula un acorde triada y guarda como variables estáticas (que mantienen su valor) las dos notas adicionales. Cuando la velocidad es nula, envía tres mensaje de *NoteOff*, correspondientes a la nota original y a las dos adicionales.

```
Public Sub Acorde(ch As Integer, nota As Integer, vel As Integer)
    'esta función calcula la tercera y la quinta de la nota
    'recibida (se supone que ésta pertenece a la escala de Do Mayor)
    'para crear el acorde correspondiente

    'son static, para guardar el valor cuando le toca hacer
    'note off (es decir cuando velocity llega con valor nulo)
    Static tercera As Integer
    Static quinta As Integer

    If vel = 0 Then
        NoteOff ch, nota
        NoteOff ch, tercera
        NoteOff ch, quinta
    Else
        Select Case (nota Mod 12)
            Case 0: tercera = 4
                    quinta = 7
            Case 2: tercera = 5
                    quinta = 9
            Case 4: tercera = 7
                    quinta = 11
            Case 5: tercera = 9
                    quinta = 12
            Case 7: tercera = 11
```

```
                quinta = 14
            Case 9: tercera = 12
                quinta = 16
            Case 11: tercera = 14
                quinta = 17
        End Select
        tercera = nota + tercera
        quinta = nota + quinta
        While tercera > 127
            tercera = tercera - 12
        Wend
        While quinta > 127
            quinta = quinta - 12
        Wend
        NoteOn ch, nota, vel
        NoteOn ch, tercera, vel
        NoteOn ch, quinta, vel
    End If
End Sub
```

Mouse Up - Note OFF

Al soltar el ratón se desactiva la nota, guardada como variable global, mediante un mensaje de *NoteOff*. Si hubiésemos disparado un acorde, no bastará con callar una nota, ya que habrá que anular también todas las del acorde. En este caso, se realiza una llamada a la función *Acorde()* pasándole una velocidad nula.

```
Private Sub Pizarra_MouseUp(Button As Integer, Shift As Integer, x As
Single, Y As Single)
    If modo = Notas12 Then
        NoteOff canal, nota
   ElseIf modo = Notas7 Then
        If Button = BOTDER Then
            Acorde canal, nota, 0
        Else
            NoteOff canal, nota
        End If
    End If
End Sub
```

Mouse Move - Envío de controles

Esta rutina normaliza las coordenadas del ratón, de forma similar a la rutina *MouseDown*. Si el ratón estuviera pulsado, manda mensaje de volumen y de *pitch bend*. En Visual Basic, cuando se hace clic sobre un control, éste sigue poseyendo al ratón, aun cuando se salga de su zona. Este implica que las coordenadas pueden dispararse a valores negativos o por encima de 127, por lo que debemos realizar una corrección.

En cualquier caso, aunque no hubiera ningún botón pulsado, muestra por pantalla la posición actual del ratón en dos controles de texto.

```
Private Sub Pizarra_MouseMove(Button As Integer, Shift As Integer, X
As Single, Y As Single)
    Dim xi As Integer
    Dim yi As Integer
    xi = Int(X / Tamano)
    yi = 127 - Int(Y / Tamano)  'invertimos sentido de ordenadas
```

```
    If Button <> NOBOTN Then
        'cuando el ratón está pulsado, el control sigue controlando el
        'movimiento, aunque el ratón salga de su área. Por ello,
        ' se deben acotar correctamente los valores
        If xi >= 0 And xi <= 127 Then PBend canal, xi * 255
        If yi >= 0 And yi <= 127 Then CChange canal, 7, yi
    End If
    Text1(0).Text = xi ‘muestra las coordenas del ratón
    Text1(1).Text = yi
End Sub
```

17.13.7. Codificación de las funciones de mensaje MIDI

Incluimos a continuación el código completo de las cinco funciones de mensaje utilizadas en el ejemplo, y que ya fueron comentadas en 17.8. Como se verá, son todas muy similares. Recordemos que *hMidiOut* es una variable global. No habría ningún problema en tener varios dispositivos MIDI abiertos, en cuyo caso sería más conveniente pasar el dispositivo al que queremos enviar el mensaje, como un parámetro adicional, en cada una de estas funciones. Por último, si programa en C, recuerde que el uso de uniones o bien de operadores de desplazamiento (>> y <<) sería bastante más eficaz que las multiplicaciones que realizamos aquí, por limitaciones del lenguaje.

```
Sub NoteOn(ch As Integer, nota As Integer, vel As Integer)
    Dim msg As Long
    msg = (vel * 65536) Or (nota * 256) Or NOTE Or ch
    midiOutShortMsg hMidiOut, msg
End Sub
```

Observe que en lugar de enviar propiamente mensajes MIDI de *Note Off*, mandamos en realidad mensajes de *Note On* con velocidad cero (véase 8.5.2).

```
Public Sub NoteOff(ch As Integer, nota As Integer)
    Dim msg As Long
    msg = (nota * 256) Or NOTE Or ch
    midiOutShortMsg hMidiOut, msg
End Sub

Public Sub CChange(ch As Integer, control As Integer, val As Integer)
    Dim msg As Long
    msg = (val * 65536) Or (control * 256) Or CC Or ch
    midiOutShortMsg hMidiOut, msg
End Sub

Sub PChange(ch As Integer, Prog As Integer)
    Dim msg As Long
    msg = Prog * 256 Or PC Or ch
    midiOutShortMsg hMidiOut, msg
End Sub

Sub PBend(ch As Integer, valor As Long)
    Dim msg As Long
    msg = (valor * 256) Or PB Or ch
    midiOutShortMsg hMidiOut, msg
End Sub
```

17.13.8. Fin de la aplicación

Al pulsar el botón de comando *ComEnd*, descargamos la ventana principal, cerramos el puerto MIDI y ¡terminamos el programa!

```
Private Sub ComEnd_Click()
    Unload Ejemplo
    End
End Sub

Private Sub Form_Unload(Cancel As Integer)
      midiOutClose hMidiOut
End Sub
```

17.14. Como compartir puertos

La mayoría de programas MIDI comerciales son muy egoístas con los puertos MIDI que abren. Trate de tener dos secuenciadores simultáneos y verá los mensajes de error que se producen al intentar abrir la segunda aplicación. Una posible solución para la multitarea MIDI radica en el uso de los puertos virtuales descritos en 15.5.3 y 15.5.4, pero la programación a bajo nivel aporta una solución adicional.

Las direcciones de los puertos (tanto los de entrada como los de salida) se mantienen constantes en una máquina dada. Esto significa que puede conocer las direcciones de estos puertos en su ordenador y guardarlas como constantes en un programa.

Supongamos que inicia la ejecución de un secuenciador comercial. Este abre todos los puertos y toma posesión de ellos. A continuación ejecuta usted su programa y, sin abrir ningún puerto (no podría puesto que ya están abiertos) utiliza la dirección que ya conoce, para mandar mensajes con la función *midiOutShortMsg()*.

Otra posibilidad más flexible consiste en cerrar momentáneamente desde su programa el puerto con la dirección conocida y volverlo a abrir inmediatamente después. El secuenciador ni se entera, y los dos programas comparten el puerto tranquilamente. Esta segunda forma podría aplicarse en cualquier ordenador; bastaría con que la primera vez que se ejecuta el programa los puertos estuvieran disponibles y guardara las direcciones obtenidas en algún fichero (un .ini por ejemplo).

17.15. Recapitulación

Ha sido largo, pero esperamos que haya valido la pena. Como ocurre siempre con los primeros ejemplos, hemos conseguido realizar un programa bastante "inútil", pero aun así, muy demostrativo de lo que puede suponer y aportar la programación MIDI. Muchos puntos no han podido ser tratados por razones de espacio y de complejidad, pero lo visto cubre bastantes de las posibles necesidades de un programador MIDI que comienza. Ahora ya sabe como estudiar los dispositivos instalados, analizar sus propiedades y enviar mensajes MIDI básicos, lo cual puede ser muy útil para un sinfín de aplicaciones multimedia.

17.16. Uso de librerías de programación MIDI

El siguiente paso consistiría en escribir un pequeño secuenciador, pero esto implicaría complicaciones mucho mayores, inabarcables desde estas páginas. De hecho, si decide escribir una aplicación MIDI "seria", le convendría utilizar algunas de las librerías de programación MIDI que se encuentran disponibles para Windows. Estas utilidades suelen ser de dominio público y pueden encontrarse en Internet. Algunas permiten el uso de Visual Basic, aunque la mayoría aconsejan el uso del lenguaje C, o más específicamente, de Visual C++ de Microsoft. Todas ellas incorporan rutinas que simplifican muchas de las tareas comunes a toda aplicación MIDI convencional (secuenciación, sincronía, lectura y escritura de ficheros MIDI estándar, etc.). Dos muy recomendables son los *paquetes MIDIShare Development Tools*, que incluye librerías en C y en Java para Windows 95, y el *MIDI Programmers Toolkit*, especialmente enfocado a Visual C++ 4.x. En el apéndice C se incluyen direcciones de Internet donde conseguir éstas y otras aplicaciones relacionadas.