第二十二章　声音与音乐

在 Microsoft Windows 中，声音、音乐与视讯的综合运用是一个重要的进步。对多媒体的支援起源於 1991 年所谓的 Microsoft Windows 多媒体延伸功能（Multimedia Extensions to Microsoft Windows）。1992 年，Windows 3.1 的发布使得对多媒体的支援成为另一类 API。最近几年，CD-ROM 驱动器和音效卡——在 90 年代初期还很少见——已成为新 PC 的标准配备。现在，几乎所有的人们都深信：多媒体在很大程度上有益於 Windows 的视觉化图形，从而使电脑摆脱了其只是处理数字和文字的机器的传统角色。

WINDOWS 和多媒体

从某种意义上来说，多媒体就是透过与装置无关的函式呼叫来获得对各种硬体的存取。让我们首先看一下硬体，然後再看看 Windows 多媒体 API 的结构。

多媒体硬体

或许最常用的多媒体硬体就是波形声音设备，也就是平常所说的音效卡。波形声音设备将麦克风的输入或其他声音输入转换为数位取样，并将其储存到记忆体或者储存到以.WAV 为副档名的磁碟档案中。波形声音设备还将波形转换回类比声音，以便通过 PC 扩音器来播放。

音效卡通常还包含 MIDI 设备。MIDI 是符合工业标准的乐器数位化介面（Musical Instrument Digital Interface）。这类硬体播放音符以回应短的二进位命令讯息。MIDI 硬体通常还可以通过电缆连结到如音乐键盘等的 MIDI 输入设备上。通常，外部的 MIDI 合成器也能够添加到音效卡上。

现在，大多数 PC 上的 CD-ROM 驱动器都具备播放普通音乐 CD 的能力。这就是平常所说的「CD 声音」。来自波形声音设备、MIDI 设备以及 CD 声音设备的输出，一般在使用者的控制下用「音量控制」程式混合在一起。

另外几种普遍的多媒体「设备」不需要额外的硬体。Windows 视讯设备（也称作 AVI 视讯设备）播放副档名为.AVI（audio-video interleave：声音视频插格）的电影或动画档案。「ActiveMovie 控制项」可以播放其他型态的电影，包括 QuickTime 和 MPEG。PC 上的显示卡需要特定的硬体来协助播放这些电影。

还有个别 PC 使用者使用某种 Pioneer 雷射影碟机或者 Sony VISCA 系列录放影机。这些设备都有序列埠介面，因此可由 PC 软体来控制。某些显示卡具有一种称为「视窗影像（video in a window）」的功能，此功能允许一个外部的

视讯信号与其他应用程式一起出现在 Windows 的萤幕上。这也可认为是一种多媒体设备。

API 概述

在 Windows 中，API 支援的多媒体功能主要分成两个集合。它们通常称为「低阶」和「高阶」介面。

低阶介面是一系列函式，这些函式以简短的说明性字首开头，而且在/Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Functions（与高阶函式一起）中列出。

低阶的波形声音输入输出函式的字首是 waveIn 和 waveOut。我们将在本章看到这些函式。另外，本章还讨论用 midiOut 函式来控制 MIDI 输出设备。这些 API 还包括 midiIn 和 midiStream 函式。

本章还使用字首为 time 的函式，这些函式允许设定一个高解析度的计时器常式，其计时器的时间间隔速率最低能够到 1 毫秒。此程式主要用於播放 MIDI 音乐。其他几组函式包括声音压缩、视讯压缩以及动画和视讯序列，可惜的是本章不包括这些函式。

您还会注意到多媒体函式列表中七个带有字首 mci 的函式，它们允许存取媒体控制介面（MCI：Media Control Interface）。这是一个高阶的开放介面，用於控制多媒体 PC 中所有的多媒体硬体。MCI 包括所有多媒体硬体都共有的许多命令，因为多媒体的许多方面都以磁带答录机这类设备播放/记录方式为模型。您为输入或输出而「打开」一台设备，进而可以「录音」（对於输入）或者「播放」（对於输出），并且结束後可以「关闭」设备。

MCI 本身分为两种形式。一种形式下，可以向 MCI 发送讯息，这类似於 Windows 讯息。这些讯息包括位元编码标记和 C 资料结构。另一种形式下，可以向 MCI 发送文字字串。这个程式主要用於描述命令语言，此语言具有灵活的字串处理函式，但支援呼叫 Windows API 的函式不多。字串命令版的 MCI 还有利於交互研究和学习 MCI，我们马上就举一个例子。MCI 中的设备名称包括 CD 声音（cdaudio）、波形音响（waveaudio）、MIDI 编曲器（sequencer）、影碟机（videodisc）、vcr、overlay（视窗中的类比视频）、dat（digital audio tape：数位式录频磁带）以及数位视频（digitalvideo）。MCI 设备分为「简单型」和「混合型」。简单型设备（如 CD 声音）不使用档案。混合型设备（如波形音响）则使用档案。使用波形音响时，这些档案的副档名是.WAV。

存取多媒体硬体的另一种方法包括 DirectX API，它超出了本书的范围。

另外两个高阶多媒体函式也值得一提：MessageBeep 和 PlaySound，它们在

第三章有示范。MessageBeep 播放「控制台」的「声音」中指定的声音。PlaySound 可播放磁碟上、记忆体中或者作为资源载入的.WAV 档案。本章的後面还会用到 PlaySound 函式。

用 TESTMCI 研究 MCI

在 Windows 多媒体的早期，软体开发套件含有一个名为 MCITEST 的 C 程式，它允许程式写作者交谈式输入 MCI 命令并学习这些命令的工作方式。这个程式，至少是 C 语言版，显然已经消失了。因此，我又重新建立了它，即程式 22-1 所示的 TESTMCI 程式。虽然我不认为目前程式码与旧的程式码有什么区别，但现在的使用者介面还是依据以前的 MCITEST 程式，并且没有使用现在的程式码。

程式 22-1 TESTMCI

```
TESTMCI.C
/*-----------------------------------------------------------------------
-
      TESTMCI.C -- MCI Command String Tester
                                                (c) Charles Petzold, 1998
-----------------------------------------------------------------------
*/

#include <windows.h>
#include "resource.h"

#define ID_TIMER    1
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("TestMci") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                          PSTR szCmdLine, int iCmdShow)
{
      if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                szAppName, MB_ICONERROR) ;
      }
      return 0 ;
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static HWND hwndEdit ;
      int        iCharBeg, iCharEnd, iLineBeg, iLineEnd, iChar, iLine, iLength ;
      MCIERROR               error ;
      RECT                   rect ;
```

```
TCHAR                  szCommand [1024], szReturn [1024],
                            szError [1024], szBuffer [32] ;

switch (message)
{
case  WM_INITDIALOG:
                            // Center the window on screen

           GetWindowRect (hwnd, &rect) ;
           SetWindowPos (hwnd, NULL,
           (GetSystemMetrics (SM_CXSCREEN) - rect.right + rect.left) / 2,
           (GetSystemMetrics (SM_CYSCREEN) - rect.bottom + rect.top) / 2,
                0, 0, SWP_NOZORDER | SWP_NOSIZE) ;

                hwndEdit = GetDlgItem (hwnd, IDC_MAIN_EDIT) ;
                SetFocus (hwndEdit) ;
                return FALSE ;

case  WM_COMMAND:
                switch (LOWORD (wParam))
                {
           case  IDOK:
           // Find the line numbers corresponding to the selection

        SendMessage (hwndEdit, EM_GETSEL, (WPARAM) &iCharBeg,
                     (LPARAM) &iCharEnd) ;

       iLineBeg = SendMessage (hwndEdit, EM_LINEFROMCHAR, iCharBeg, 0) ;
       iLineEnd = SendMessage (hwndEdit, EM_LINEFROMCHAR, iCharEnd, 0) ;

           // Loop through all the lines

           for (iLine = iLineBeg ; iLine <= iLineEnd ; iLine++)
                {
           // Get the line and terminate it; ignore if blank

           * (WORD *) szCommand = sizeof (szCommand) / sizeof (TCHAR) ;

                iLength = SendMessage  (hwndEdit, EM_GETLINE, iLine,
                                  (LPARAM) szCommand) ;
             szCommand [iLength] = '\0' ;

                                             if (iLength == 0)
                                             continue ;

     // Send the MCI command

     error =     mciSendString (szCommand, szReturn,
```

```
          sizeof (szReturn) / sizeof (TCHAR), hwnd) ;

                    // Set the Return String field

          SetDlgItemText (hwnd, IDC_RETURN_STRING, szReturn) ;

                   // Set the Error String field (even if no error)

         mciGetErrorString (error, szError, sizeof (szError) / sizeof (TCHAR)) ;

                SetDlgItemText (hwnd, IDC_ERROR_STRING, szError) ;
                     }
                     // Send the caret to the end of the last selected line

                iChar  = SendMessage (hwndEdit, EM_LINEINDEX,  iLineEnd, 0) ;
                iChar += SendMessage (hwndEdit, EM_LINELENGTH, iCharEnd, 0) ;
                     SendMessage (hwndEdit, EM_SETSEL, iChar, iChar) ;

                     // Insert a carriage return/line feed combination

                     SendMessage (hwndEdit, EM_REPLACESEL, FALSE,
                         (LPARAM) TEXT ("\r\n")) ;
                                SetFocus (hwndEdit) ;
                                return TRUE ;

                     case  IDCANCEL:
                                      EndDialog (hwnd, 0) ;
                                      return TRUE ;
                     case  IDC_MAIN_EDIT:
                                      if (HIWORD (wParam) == EN_ERRSPACE)
                                      {
              MessageBox (hwnd, TEXT ("Error control out of space."),
                          szAppName, MB_OK | MB_ICONINFORMATION) ;
             return TRUE ;
                }
                break ;
                     }
                     break ;

    case  MM_MCINOTIFY:
                EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_MESSAGE), TRUE) ;

                     wsprintf (szBuffer, TEXT ("Device ID = %i"), lParam) ;
                     SetDlgItemText (hwnd, IDC_NOTIFY_ID, szBuffer) ;
                     EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ID), TRUE) ;

                     EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUCCESSFUL),
                           wParam & MCI_NOTIFY_SUCCESSFUL) ;
```

```
                    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUPERSEDED),
                          wParam & MCI_NOTIFY_SUPERSEDED) ;

                    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ABORTED),
                          wParam & MCI_NOTIFY_ABORTED) ;

                    EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_FAILURE),
                          wParam & MCI_NOTIFY_FAILURE) ;

                    SetTimer (hwnd, ID_TIMER, 5000, NULL) ;
               return TRUE ;

     case  WM_TIMER:
          KillTimer (hwnd, ID_TIMER) ;

          EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_MESSAGE), FALSE) ;
          EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ID), FALSE) ;
          EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUCCESSFUL), FALSE) ;
          EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_SUPERSEDED), FALSE) ;
          EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_ABORTED), FALSE) ;
          EnableWindow (GetDlgItem (hwnd, IDC_NOTIFY_FAILURE), FALSE) ;
          return TRUE ;

     case  WM_SYSCOMMAND:
                    switch (LOWORD (wParam))
                    {
                    case  SC_CLOSE:
                                   EndDialog (hwnd, 0) ;
                                   return TRUE ;
                    }
                    break ;
    }
     return FALSE ;
}
```

TESTMCI.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Dialog
TESTMCI     DIALOG DISCARDABLE  0, 0, 270, 276
STYLE            WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION     "MCI Tester"
FONT 8,     "MS Sans Serif"
BEGIN
```

```
      EDITTEXT                IDC_MAIN_EDIT,8,8,254,100,ES_MULTILINE          |
ES_AUTOHSCROLL |
                                        WS_VSCROLL
      LTEXT                         "Return String:",IDC_STATIC,8,114,60,8
      EDITTEXT                IDC_RETURN_STRING,8,126,120,50,ES_MULTILINE |
                                        ES_AUTOVSCROLL | ES_READONLY | WS_GROUP
| NOT WS_TABSTOP
      LTEXT                         "Error String:",IDC_STATIC,142,114,60,8
      EDITTEXT                IDC_ERROR_STRING,142,126,120,50,ES_MULTILINE |
                                        ES_AUTOVSCROLL  |  ES_READONLY  |  NOT
WS_TABSTOP
      GROUPBOX                "MM_MCINOTIFY Message",IDC_STATIC,9,186,254,58
      LTEXT                         "",IDC_NOTIFY_ID,26,198,100,8
      LTEXT
      "MCI_NOTIFY_SUCCESSFUL",IDC_NOTIFY_SUCCESSFUL,26,212,100,
                                        8,WS_DISABLED
      LTEXT
      "MCI_NOTIFY_SUPERSEDED",IDC_NOTIFY_SUPERSEDED,26,226,100,
                                        8,WS_DISABLED
      LTEXT
      "MCI_NOTIFY_ABORTED",IDC_NOTIFY_ABORTED,144,212,100,8,
                                        WS_DISABLED
      LTEXT
      "MCI_NOTIFY_FAILURE",IDC_NOTIFY_FAILURE,144,226,100,8,
                                        WS_DISABLED
      DEFPUSHBUTTON     "OK",IDOK,57,255,50,14
      PUSHBUTTON              "Close",IDCANCEL,162,255,50,14
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by TestMci.rc

#define IDC_MAIN_EDIT                                           1000
#define IDC_NOTIFY_MESSAGE                                      1005
#define IDC_NOTIFY_ID                                           1006
#define IDC_NOTIFY_SUCCESSFUL                                  1007
#define IDC_NOTIFY_SUPERSEDED                                   1008
#define IDC_NOTIFY_ABORTED                                      1009
#define IDC_NOTIFY_FAILURE                                      1010
#define IDC_SIGNAL_MESSAGE                                      1011
#define IDC_SIGNAL_ID                                           1012
#define IDC_SIGNAL_PARAM                                        1013
#define IDC_RETURN_STRING                                       1014
#define IDC_ERROR_STRING                                        1015
#define IDC_DEVICES                                             1016
#define IDC_STATIC                                              -1
```

与本章的大多数程式一样，TESTMCI 使用非模态对话方块作为它的主视窗。

与本章所有的程式一样，TESTMCI 要求 WINMM.LIB 引用程式库在 Microsoft Visual C++「Projects Settings」对话方块的「Links」页列出。

此程式用到了两个最重要的多媒体函式：mciSendString 和 mciGetErrorText。在 TESTMCI 的主编辑视窗输入一些内容然後按下 Enter 键(或「OK」按钮）後，程式将输入的字串作为第一个参数传递给 mciSendString 命令：

```
error =    mciSendString (szCommand, szReturn,
                          sizeof (szReturn) / sizeof (TCHAR), hwnd) ;
```

如果在编辑视窗选择了不止一行，则程式将按顺序将它们发送给 mciSendString 函式。第二个参数是字串位址，此字串取得从函式传回的资讯。程式将此资讯显示在视窗的「Return String」区域。从 mciSendString 传回的错误代码传递给 mciGetErrorString 函式，以获得文字错误说明；此说明显示在 TESTMCI 视窗的「Error String」区域。

MCITEXT 和 CD 声音

通过控制 CD-ROM 驱动器和播放声音 CD，您会对 MCI 命令字串留下很好的印象。因为这些命令字串一般都非常简单，并且更重要的是您可以听到一些音乐，所以这是好的起点。您可以在/Platform SDK/Graphics and Multimedia Services/Multimedia Reference/Multimedia Command Strings 中获得 MCI 命令字串的参考，以方便本练习。

请确认 CD-ROM 驱动器的声音输出已连结到扩音器或耳机，然後放入一张声音 CD，如 Bruce Springsteen 的「Born to Run」。Windows 98 中，「CD 播放程式」将启动并开始播放此唱片。如果是这样的话，终止「CD 播放程式」，然後可以叫出 TESTMCI 并且键入命令：

```
open cdaudio
```

然後按 Enter 键。其中 open 是 MCI 命令，cdaudio 是 MCI 认定的 CD-ROM 驱动器的设备名称（假定您的系统中只有一个 CD-ROM 驱动器。要获得多个 CD-ROM 驱动器名称需使用 sysinfo 命令）。

TESTMCI 中的「Return String」区域显示 mciSendString 函式中系统传回给程式的字串。如果执行了 open 命令，则此值是 1。TESTMCI 在「Error String」区域中显示 mciGetErrorString 依据 mciSendString 传回值所传回的资讯。如果 mciSendString 没有传回错误代码，则「Error String」区域显示文字"The specified command was carried out"。

假定执行了 open 命令，现在就可以输入：

```
play cdaudio
```

CD 将开始播放唱片上的第一首乐曲「Thunder Road」。输入下面的命令可以暂停播放：

```
pause cdaudio
```

或者

```
stop cdaudio
```

对於 CD 声音设备来说，这些叙述的功能相同。您可用下面的叙述重新播放：

```
play cdaudio
```

迄今为止，我们使用的全部字串都由命令和设备名称组成。其中有些命令带有选项。例如，键入：

```
status cdaudio position
```

根据收听时间的长短，「Return String」区域将显示类似下面的一些字元：

01:15:25

这是些什么？很显然不是小时、分钟和秒，因为 CD 没有那么长。要找出时间格式，请键入：

```
status cdaudio time format
```

现在「Return String」区域显示下面的字串：

msf

这代表「分-秒-格」。CD 声音中，每秒有 75 格。时间格式的讯格部分可在 0 到 74 之间的范围内变化。

状态命令有一连串的选项。使用下面的命令，您可以确定 msf 格式的 CD 全部长度：

```
status cdaudio length
```

对於「Born to Run」，「Return String」区域将显示：

39:28:19

这指的是 39 分 28 秒 19 格。

现在试一下

status cdaudio number of tracks

「Return String」区域将显示：

8

我们从 CD 封面上知道「Born to Run」CD 上第五首乐曲是主题曲。MCI 命令中的乐曲从 1 开始编号。要想知道乐曲「Born to Run」的长度，可以键入下面的命令：

```
status cdaudio length track 5
```

「Return String」区域将显示：

04:30:22

我们还可确定此乐曲从盘上的哪个位置开始：

status cdaudio position track 5

「Return String」区域将显示：

17:36:35

根据这条资讯，我们可以直接跳到乐曲标题：

play cdaudio from 17:36:35 to 22:06:57

此命令只播放一首乐曲，然後停止。最後的值是由 4:30:22（乐曲长度）加 17:36:35 得到的。或者，也可以用下面的命令确定：

status cdaudio position track 6

或者，也可以将时间格式设定为乐曲-分-秒-格：

set cdaudio time format tmsf

然後

play cdaudio from 5:0:0:0 to 6:0:0:0

或者，更简单地

play cdaudio from 5 to 6

如果时间的尾部是 0，那么您可去掉它们。还可以用毫秒设定时间格式。

每个 MCI 命令字串都可以在字串的後面包括选项 wait 和 notify（但不是同时使用）。例如，假设您只想播放「Born to Run」的前 10 秒，而且播放後，您还想让程式完成其他工作。您可按下面的方法进行（假定您已经将时间格式设定为 tmsf）：

play cdaudio from 5:0:0 to 5:0:10 wait

这种情况下，直到函式执行结束，也就是说，直到播放完「Born to Run」的前 10 秒，mciSendString 函式才传回。

现在很明显，一般来说，在单执行绪的应用程式中这不是一件好事。如果不小心键入：

play cdaudio wait

直到整个唱片播放完以後，mciSendString 函式才将控制权传回给程式。如果必须使用 wait 选项（在只要执行 MCI 描述档案而不管其他事情的时候，这么做很方便，与我将展示的一样），首先使用 break 命令。此命令可设定一个虚拟键码，此码将中断 mciSendString 命令并将控制权传回给程式。例如，要设定 Escape 键来实作此目的，可用：

break cdaudio on 27

这里，27 是十进位的 VK_ESCAPE 值。

比 wait 选项更好的是 notify 选项：

play cdaudio from 5:0:0 to 5:0:10 notify

这种情况下，mciSendString 函式立即传回，但如果该操作在 MCI 命令的尾部定义，则 mciSendString 函式的最後一个参数所指定代号的视窗会收到 MM_MCINOTIFY 讯息。TESTMCI 程式在 MM_MCINOTIFY 框中显示此讯息的结果。为避免与其他可能键入的命令混淆,TESTMCI 程式在 5 秒後停止显示 MM_MCINOTIFY 讯息的结果。

您可以同时使用 wait 和 notify 关键字，但没有理由这么做。不使用这两个关键字，内定的操作就既不是 wait，也不是您通常所希望的 notify。

用这些命令结束播放时，可键入下面的命令来停止 CD：

stop cdaudio

如果在关闭之前没有停止 CD-ROM 设备，那么甚至在关闭设备之後还会继续播放 CD。

另外，您还可以试试您的硬体允许或者不允许的一些命令：

eject cdaudio

最後按下面的方法关闭设备：

close cdaudio

虽然 TESTMCI 自己不能储存或载入文字档案，但可以在编辑控制项和剪贴簿之间复制文字：先从 TESTMCI 选择一些内容，将其复制到剪贴簿（用 Ctrl-C），再将这些文字从剪贴簿复制到「记事本」，然後储存。相反的操作，可以将一系列的 MCI 命令载入到 TESTMCI。如果选择了一系列命令然後按下「OK」按钮（或者 Enter 键），则 TESTMCI 将每次执行一条命令。这就允许您编写 MCI 的「描述档案」，即 MCI 命令的简单列表。

例如，假设您想听歌曲「Jungleland」（唱片中的最後一首）、「Thunder Road」和「Born to Run」，并要按此顺序听，可以编写如下的描述命令：

```
open cdaudio
set cdaudio time format tmsf
break cdaudio on 27
play  cdaudio from 8 wait
play  cdaudio from 1 to 2 wait
play  cdaudio from 5 to 6 wait
stop  cdaudio
eject       cdaudio
close       cdaudio
```

不用 wait 关键字，就不能正常工作，因为 mciSendString 命令会立即传回，然後执行下一条命令。

此时，如何编写模拟 CD 播放程式的简单应用程式，就应该相当清楚了。程式可以确定乐曲数量、每个乐曲的长度并能显示允许使用者从任意位置开始播放（不过，请记住：mciSendString 总是传回文字字串资讯，因此您需要编写解

析处理程式来将这些字串转换成数字)。可以肯定,这样的程式还要使用 Windows 计时器，以产生大约 1 秒的时间间隔。在 WM_TIMER 讯息处理期间，程式将呼叫：

```
status cdaudio mode
```

来查看 CD 是暂停还是在播放。

```
status cdaudio position
```

命令允许程式更新显示以给使用者显示目前的位置。但可能还存在更令人感兴趣的事：如果程式知道音乐音调部分的节拍位置，那么就可以使萤幕上的图形与 CD 同步。这对於音乐指令或者建立自己的图形音乐视讯程式极为有用。

波形声音

波形声音是最常用的 Windows 多媒体特性。波形声音设备可以通过麦克风捕捉声音，并将其转换为数值，然後把它们储存到记忆体或者磁碟上的波形档案中，波形档案的副档名是.WAV。这样，声音就可以播放了。

声音与波形

在接触波形声音 API 之前，具备一些预备知识很重要，这些知识包括物理学、听觉以及声音进出电脑的程序。

声音就是振动。当声音改变了鼓膜上空气的压力时，我们就感觉到了声音。麦克风可以感应这些振动，并且将它们转换为电流。同样，电流再经过放大器和扩音器，就又变成了声音。传统上，声音以类比方式储存（例如录音磁带和唱片），这些振动储存在磁气脉冲或者轮廓凹槽中。当声音转换为电流时，就可以用随时间振动的波形来表示。振动最自然的形式可以用正弦波表示，它的一个周期如图 5-5 所示。

正弦波有两个参数——振幅（也就是一个周期中的最大振幅）和频率。我们已知振幅就是音量，频率就是音调。一般来说人耳可感受的正弦波的范围是从 20Hz（每秒周期）的低频声音到 20,000Hz 的高频声，但随著年龄的增长，对高频声音的感受能力会逐年退化。

人感受频率的能力与频率是对数关系而不是线性关系。也就是说，我们感受 20Hz 到 40Hz 的频率变化与感受 40Hz 到 80Hz 的频率变化是一样的。在音乐中，这种加倍的频率定义为八度音阶。因此，人耳可感觉到大约 10 个八度音阶的声音。钢琴的范围是从 27.5 Hz 到 4186 Hz 之间，略小於 7 个八度音阶。

虽然正弦波代表了振动的大多数自然形式，但纯正弦波很少在现实生活中单独出现，而且，纯正弦波并不动听。大多数声音都很复杂。

任何周期的波形（即，一个回圈波形）可以分解成多个正弦波，这些正弦

波的频率都是整倍数。这就是所谓的 Fourier 级数，它以法国数学家和物理学家 Jean Baptiste Joseph Fourier（1768-1830）的名字命名。周期的频率是基础。级数中其他正弦波的频率是基础频率的 2 倍、3 倍、4 倍（等等）。这些频率的声音称为泛音。基础频率也称作一级谐波。第一泛音是二级谐波，以此类推。

正弦波谐波的相对强度给每个周期的波形唯一的声音。这就是「音质」，它使得喇叭吹出喇叭声，钢琴弹出钢琴声。

人们一度认为电子合成乐器仅仅需要将声音分解成谐波并且与多个正弦波重组即可。不过，事实证明现实世界中的声音并不是这么简单。代表现实世界中声音的波形都没有严格的周期。乐器之间谐波的相对强度是不同的，并且谐波也随著每个音符的演奏时间改变。特别是乐器演奏音符的开始位置——我们称作起奏（attack）——相当复杂，但这个位置又对我们感受音质至关重要。

由於近年来数位储存能力的提高，我们可以将声音直接以数位形式储存而不用复杂的重组。

脉冲编码调制（Pulse Code Modulation）

电脑处理的是数值，因此要使声音进入电脑，就必须设计一种能将声音与数位信号相互转换的机制。

不压缩资料就完成此功能的最常用方法称作「脉冲编码调制」（PCM：pulse code modulation）。PCM 可用在光碟、数位式录音磁带以及 Windows 中。脉冲编码调制其实只是一种概念上很简单的处理步骤的奇怪代名词而已。

利用脉冲编码调制，波形可以按固定的周期频率取样，其频率通常是每秒几万次。对於每个样本都测量其波形的振幅。完成将振幅转换成数位信号工作的硬体是类比数位转换器（ADC：analog-to-digital converter）。类似地，通过数位类比转换器（DAC：digital-to-analog converter）可将数位信号转换回波形电子信号。但这样转换得到的波形与输入的并不完全相同。合成的波形具有由高频组成的尖锐边缘。因此，播放硬体通常在数位类比转换器後还包括一个低通滤波器。此滤波器滤掉高频，并使合成後的波形更平滑。在输入端，低通滤波器位於 ADC 前面。

脉冲编码调制有两个参数：取样频率，即每秒内测量波形振幅的次数；样本大小，即用於储存振幅级的位元数。与您想像的一样：取样频率越高，样本大小越大，原始声音的复制品才更好。不过，存在一个提高取样频率和样本大小的极点，超过这个极点也就超过了人类分辨声音的极限。另外，如果取样频率和样本大小过低，将导致不能精确地复制音乐以及其他声音。

取样频率

取样频率决定声音可被数位化和储存的最大频率。尤其是，取样频率必须是样本声音最高频率的两倍。这就是「Nyquist 频率（Nyquist Frequency）」，以 30 年代研究取样程序的工程师 Harry Nyquist 的名字命名。

以过低的取样频率对正弦波取样时，合成的波形比最初的波形频率更低。这就是所说的失真信号。为避免失真信号的发生，在输入端使用低通滤波器以阻止频率大於半个取样频率的所有波形。在输出端，数位类比转换器产生的粗糙的波形边缘实际上是由频率大於半个取样频率的波形组成的泛音。因此，位於输出端的低通滤波器也阻止频率大於半个取样频率的所有波形。

声音 CD 中使用的取样频率是每秒 44,100 个样本，或者称为 44.1kHz。这个特有的数值是这样产生的：

人耳可听到最高 20kHz 的声音，因此要拦截人能听到的整个声音范围，就需要 40kHz 的取样频率。然而，由於低通滤波器具有频率下滑效应，所以取样频率应该再高出大约百分之十才行。现在，取样频率就达到了 44kHz。这时，我们要与视讯同时记录数位声音，於是取样频率就应该是美国、欧洲电视显示格速率的整数倍，这两种视讯格速率分别是 30Hz 和 25Hz。这就使取样频率升高到了 44.1kHz。

取样频率为 44.1kHz 的光碟会产生大量的资料，这对於一些应用程式来说实在是太多了，例如对於录制声音而不是录制音乐时就是这样。把取样频率减半到 22.05 kHz，可由一个 10 kHz 的泛音来简化复制声音的上半部分。再将其减半到 11.025 kHz 就向我们提供了 5 kHz 频率范围。44.1 kHz、22.05 kHz 和 11.025 kHz 的取样频率，以及 8 kHz 都是波形声音设备普遍支援的标准。

因为钢琴的最高频率为 4186 Hz，所以您可能会认为给钢琴录音时，11.025 kHz 的取样频率就足够了。但 4186 Hz 只是钢琴最高的基础频率而已，滤掉大於 5000Hz 的所有正弦波将减少可被复制的泛音，而这样将不能精确地捕捉和复制钢琴的声音。

样本大小

脉冲编码调制的第二个参数是按位元计算的样本大小。样本大小决定了可供录制和播放的最低音与最高音之间的区别。这就是通常所说的动态范围。

声音强度是波形振幅的平方（即每个正弦波一个周期中最大振幅的合成）。与频率一样，人对声音强度的感受也呈对数变化。

两个声音在强度上的区别是以贝尔（以电话发明人 Alexander Graham Bell

的名字命名）和分贝（dB）为单位进行测量的。1 贝尔在声音强度上呈 10 倍增加。1dB 就是以相同的乘法步骤成为 1 贝尔的十分之一。由此，1dB 可增加声音强度的 1.26 倍（10 的 10 次方根），或者增加波形振幅的 1.12 倍（10 的 20 次方根）。1 分贝是耳朵可感觉出的声强的最小变化。从开始能听到的声音极限到让人感到疼痛的声音极限之间的声强差大约是 100 dB。

可用下面的公式来计算两个声音间的动态范围，单位是分贝：

$$dB = 20 \cdot \log\left(\frac{A_1}{A_2}\right)$$

其中 A1 和 A2 是两个声音的振幅。因为只可能有一个振幅，所以样本大小是 1 位元，动态范围是 0。

如果样本大小是 8 位元，则最大振幅与最小振幅之间的比例就是 256。这样，动态范围就是：

$$dB = 20 \cdot \log(256)$$

或者 48 分贝。48 的动态范围大约相当於非常安静的房屋与电动割草机之间的差别。将样本大小加倍到 16 位元产生的动态范围是：

$$dB = 20 \cdot \log(65536)$$

或者 96 分贝。这非常接近听觉极限和疼痛极限，而且人们认为这就是复制音乐的理想值。

Windows 同时支援 8 位元和 16 位元的样本大小。储存 8 位元的样本时，样本以无正负号位元组处理，静音将储存为一个值为 0x80 的字串。16 位元的样本以带正负号整数处理，这时静音将储存为一个值为 0 的字串。

要计算未压缩声音所需的储存空间，可用以秒为单位的声音持续时间乘以取样频率。如果用 16 位元样本而不是 8 位元样本，则将其加倍，如果是录制立体声则再加倍。例如，1 小时的 CD 声音（或者是在每个立体声样本占 2 位元组、每秒 44,100 个样本的速度下进行 3 600 秒）需要 635MB，这快要接近一张 CD-ROM 的储存量了。

在软体中产生正弦波

对於第一个关於波形声音的练习，我们不打算将声音储存到档案中或播放录制的声音。我们将使用低阶的波形声音 API（即，字首是 waveOut 的函式）来建立一个称作 SINEWAVE 的声音正弦波生成器。此程式以 1 Hz 的增量来生成从

20Hz（人可感觉的最低值）到 5,000Hz（与人感觉的最高值相差两个八度音阶）的正弦波。

我们知道，标准 C 执行时期程式库包括了一个 sin 函式，该函式传回一个弧度角的正弦值（2 鸹《鹊褥 360 度）。sin 函式传回值的范围是从-1 到 1（早在第五章，我们就在 SINEWAVE 程式中使用过这个函式）。因此，应该很容易使用 sin 函式生成输出到波形声音硬体的正弦波资料。基本上是用代表波形（这时是正弦波）的资料来填充缓冲区，并将此缓冲区传递给 API。（这比前面所讲的稍微有些复杂，但我将详细介绍）。波形声音硬体播放完缓冲区中的资料後，应将第二个缓冲区中的资料传递给它，并且以此类推。

第一次考虑这个问题（而且对 PCM 也一无所知）时，您大概会认为将一个周期的正弦波分成若干固定数量的样本——例如 360 个——才合理。对於 20 Hz 的正弦波，每秒输出 7,200 个样本。对於 200 Hz 的正弦波，每秒则要输出 72,000 个样本。这有可能实作，但实际上却不能这么做。对於 5,000 Hz 的正弦波，就需要每秒输出 1,800,000 个样本，这的确会增大 DAC 的负担！更重要的是，对於更高的频率，这种作法会比实际需要的精确度还高。

就脉冲编码调制而言，取样频率是个常数。假定取样频率是 SINEWAVE 程式中使用的 11,025Hz。如果要生成一个 2,756.25Hz（确切地说是四分之一的取样频率）的正弦波，则正弦波的每个周期就有 4 个样本。对於 25Hz 的正弦波，每个周期就有 441 个样本。通常，每周期的样本数等於取样频率除以要得到的正弦波频率。一旦知道了每周期的样本数，用 2π 弧度除以此数，然後用 sin 函式来获得每周期的样本。然後再反复对一个周期进行取样，从而建立一个连续的波形。

问题是每周期的样本数可能带有小数，因此在使用时这种方法并不是很好。每个周期的尾部都会有间断。

使它正常工作的关键是保留一个静态的「相位角」变数。此角初始化为 0。第一个样本是 0 度正弦。随後，相位角增加一个值，该值等於 2π 乘以频率再除以取样频率。用此相位角作为第二个样本，并且按此方法继续。一旦相位角超过 2π 弧度，则减去 2π 弧度，而不要把相位角再初始化为 0。

例如，假定要用 11,025Hz 的取样频率来生成 1,000Hz 的正弦波。即每周期有大约 11 个样本。为便於理解，此处相位角按度数给出——大约前一个半周期的相位角是：0、32.65、65.31、97.96、130.61、163.27、195.92、228.57、261.22、293.88、326.53、359.18、31.84、64.49、97.14、129.80、162.45、195.10，以此类推。存入缓冲区的波形资料是这些角度的正弦值，并已缩放到每样本的位元数。为後来的缓冲区建立资料时，可继续增加最後的相位角，而

不要将它初始化为 0。

如程式 22-2 所示，FillBuffer 函式完成这项工作——与 SINEWAVE 程式的其余部分一起完成。

程式 22-2 SINEWAVE

```
SINEWAVE.C
/*------------------------------------------------------------------------
      SINEWAVE.C --          Multimedia Windows Sine Wave Generator
                                                  (c) Charles Petzold, 1998
------------------------------------------------------------------------*/

#include <windows.h>
#include <math.h>
#include "resource.h"

#define           SAMPLE_RATE            11025
#define           FREQ_MIN                  20
#define           FREQ_MAX               5000
#define           FREQ_INIT              440
#define           OUT_BUFFER_SIZE      4096
#define           PI                3.14159

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("SineWave") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                  PSTR    szCmdLine,    int
iCmdShow)
{
      if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                  szAppName, MB_ICONERROR) ;
      }
      return 0 ;
}

VOID FillBuffer (PBYTE pBuffer, int iFreq)
{
      static double          fAngle ;
      int                              i ;

      for (i = 0 ; i < OUT_BUFFER_SIZE ; i++)
      {
                  pBuffer [i] = (BYTE) (127 + 127 * sin (fAngle)) ;
                  fAngle += 2 * PI * iFreq / SAMPLE_RATE ;
            if (  fAngle > 2 * PI)
                                    fAngle -= 2 * PI ;
    }
```

```
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static      BOOL                                    bShutOff, bClosing ;
      static      HWAVEOUT                          hWaveOut ;
      static      HWND                                    hwndScroll ;
      static      int                                     iFreq = FREQ_INIT ;
      static      PBYTE                                   pBuffer1, pBuffer2 ;
      static      PWAVEHDR                          pWaveHdr1, pWaveHdr2 ;
      static      WAVEFORMATEX          waveformat ;
      int                                                 iDummy ;

      switch (message)
      {
      case  WM_INITDIALOG:
            hwndScroll =      GetDlgItem (hwnd, IDC_SCROLL) ;
            SetScrollRange    (hwndScroll, SB_CTL, FREQ_MIN, FREQ_MAX, FALSE) ;
            SetScrollPos      (hwndScroll, SB_CTL, FREQ_INIT, TRUE) ;
            SetDlgItemInt     (hwnd, IDC_TEXT, FREQ_INIT, FALSE) ;

            return TRUE ;

      case  WM_HSCROLL:
                        switch (LOWORD (wParam))
                        {
                        case  SB_LINELEFT:          iFreq -=  1 ;  break ;
                        case  SB_LINERIGHT:         iFreq +=  1 ;  break ;
                        case  SB_PAGELEFT:          iFreq /=  2 ;  break ;
                        case  SB_PAGERIGHT:    iFreq *=  2 ;  break ;

                        case  SB_THUMBTRACK:
                                          iFreq = HIWORD (wParam) ;
                                          break ;

                        case  SB_TOP:
                        GetScrollRange (hwndScroll, SB_CTL, &iFreq, &iDummy) ;
                                          break ;

                        case  SB_BOTTOM:
                        GetScrollRange (hwndScroll, SB_CTL, &iDummy, &iFreq) ;
                                          break ;
                        }

                        iFreq = max (FREQ_MIN, min (FREQ_MAX, iFreq)) ;
                        SetScrollPos (hwndScroll, SB_CTL, iFreq, TRUE) ;
                        SetDlgItemInt (hwnd, IDC_TEXT, iFreq, FALSE) ;
```

```
                    return TRUE ;

     case  WM_COMMAND:
                    switch (LOWORD (wParam))
                    {
                    case  IDC_ONOFF:
                              // If turning on waveform, hWaveOut is NULL

                                   if (hWaveOut == NULL)
                                   {
                 // Allocate memory for 2 headers and 2 buffers

                pWaveHdr1   = malloc (sizeof (WAVEHDR)) ;
                pWaveHdr2   = malloc (sizeof (WAVEHDR)) ;
                pBuffer1         = malloc (OUT_BUFFER_SIZE) ;
                pBuffer2         = malloc (OUT_BUFFER_SIZE) ;

        if (!pWaveHdr1 || !pWaveHdr2 || !pBuffer1 || !pBuffer2)
               {
        if (!pWaveHdr1) free (pWaveHdr1) ;
        if (!pWaveHdr2) free (pWaveHdr2) ;
        if (!pBuffer1)  free (pBuffer1) ;
        if (!pBuffer2)  free (pBuffer2) ;

        MessageBeep (MB_ICONEXCLAMATION) ;
        MessageBox (hwnd, TEXT ("Error allocating memory!"),
                  szAppName, MB_ICONEXCLAMATION | MB_OK) ;
        return TRUE ;
       }

                    // Variable to indicate Off button pressed

                                        bShutOff = FALSE ;

                    // Open waveform audio for output

   waveformat.wFormatTag                    = WAVE_FORMAT_PCM ;
   waveformat.nChannels                     = 1 ;
   waveformat.nSamplesPerSec              = SAMPLE_RATE ;
   waveformat.nAvgBytesPerSec           = SAMPLE_RATE ;
   waveformat.nBlockAlign                   = 1 ;
   waveformat.wBitsPerSample              = 8 ;
   waveformat.cbSize                        = 0 ;

   if (waveOutOpen (&hWaveOut, WAVE_MAPPER, &waveformat,
DWORD) hwnd, 0, CALLBACK_WINDOW)!= MMSYSERR_NOERROR)
                                   {
        free (pWaveHdr1) ;
```

```
        free (pWaveHdr2) ;
        free (pBuffer1) ;
        free (pBuffer2) ;

        hWaveOut = NULL ;
        MessageBeep (MB_ICONEXCLAMATION) ;
        MessageBox (hwnd, TEXT ("Error opening waveform audio device!"),
                    szAppName, MB_ICONEXCLAMATION | MB_OK) ;
      return TRUE ;
      }

      // Set up headers and prepare them

      pWaveHdr1->lpData                                    = pBuffer1 ;
      pWaveHdr1->dwBufferLength                            = OUT_BUFFER_SIZE ;
      pWaveHdr1->dwBytesRecorded                                 = 0 ;
      pWaveHdr1->dwUser                                    = 0 ;
      pWaveHdr1->dwFlags                                   = 0 ;
      pWaveHdr1->dwLoops                                   = 1 ;
      pWaveHdr1->lpNext                                    = NULL ;
      pWaveHdr1->reserved                                  = 0 ;

      waveOutPrepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

      pWaveHdr2->lpData                                    = pBuffer2 ;
      pWaveHdr2->dwBufferLength                            = OUT_BUFFER_SIZE ;
      pWaveHdr2->dwBytesRecorded                                 = 0 ;
      pWaveHdr2->dwUser                                    = 0 ;
      pWaveHdr2->dwFlags                                   = 0 ;
      pWaveHdr2->dwLoops                                   = 1 ;
      pWaveHdr2->lpNext                                    = NULL ;
      pWaveHdr2->reserved                                  = 0 ;

     waveOutPrepareHeader (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;
                                        }
     // If turning off waveform, reset waveform audio
                else
                                        {
                                              bShutOff = TRUE ;
                                              waveOutReset (hWaveOut) ;
                                        }
                                        return TRUE ;
                      }
                      break ;

                                  // Message generated from waveOutOpen call

    case  MM_WOM_OPEN:
```

```
                    SetDlgItemText (hwnd, IDC_ONOFF, TEXT ("Turn Off")) ;

                                        // Send two buffers to waveform
output device

                    FillBuffer (pBuffer1, iFreq) ;
                    waveOutWrite (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

                    FillBuffer (pBuffer2, iFreq) ;
                    waveOutWrite (hWaveOut, pWaveHdr2, sizeof (WAVEHDR)) ;
                    return TRUE ;

                                        // Message generated when a buffer
is finished

     case  MM_WOM_DONE:
                    if (bShutOff)
                    {
                              waveOutClose (hWaveOut) ;
                              return TRUE ;
                    }

                              // Fill and send out a new buffer

                    FillBuffer (((PWAVEHDR) lParam)->lpData, iFreq) ;
                    waveOutWrite  (hWaveOut,  (PWAVEHDR)  lParam,  sizeof
(WAVEHDR)) ;
                    return TRUE ;

     case  MM_WOM_CLOSE:
                    waveOutUnprepareHeader  (hWaveOut,  pWaveHdr1,  sizeof
(WAVEHDR)) ;
                    waveOutUnprepareHeader  (hWaveOut,  pWaveHdr2,  sizeof
(WAVEHDR)) ;

                    free (pWaveHdr1) ;
                    free (pWaveHdr2) ;
                    free (pBuffer1) ;
                    free (pBuffer2) ;

                    hWaveOut = NULL ;
                    SetDlgItemText (hwnd, IDC_ONOFF, TEXT ("Turn On")) ;

                    if (bClosing)
                                   EndDialog (hwnd, 0) ;

                    return TRUE ;
```

```
      case  WM_SYSCOMMAND:
                        switch (wParam)
                        {
                        case SC_CLOSE:
                                          if (hWaveOut != NULL)
                                          {
                                                bShutOff = TRUE ;
                                                bClosing = TRUE ;

                                                waveOutReset (hWaveOut) ;
                                          }
                                          else
                                                EndDialog (hwnd, 0) ;

                                          return TRUE ;
                        }
                        break ;
            }
            return FALSE ;
}
```

SINEWAVE.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Dialog
SINEWAVE          DIALOG DISCARDABLE  100, 100, 200, 50
STYLE                   WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION           "Sine Wave Generator"
FONT 8,           "MS Sans Serif"
BEGIN
      SCROLLBAR                   IDC_SCROLL,8,8,150,12
      RTEXT                                 "440",IDC_TEXT,160,10,20,8
      LTEXT                                 "Hz",IDC_STATIC,182,10,12,8
      PUSHBUTTON                  "Turn On",IDC_ONOFF,80,28,40,14
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by SineWave.rc

#define IDC_STATIC                                                      -1
#define IDC_SCROLL                                                1000
#define IDC_TEXT                                                  1001
#define IDC_ONOFF                                                 1002
```

注意，FillBuffer 常式中用到的 OUT_BUFFER_SIZE、SAMPLE_RATE 和 PI 识

别字在程式的顶部定义。FillBuffer 的 iFreq 参数是需要的频率，单位是 Hz。还要注意，sin 函式的结果调整到了 0 到 254 的范围之间。对於每个样本，sin 函式的 fAngle 参数都增加一个值，该值的大小是 2π 弧度乘以需要的频率再除以取样频率。

SINEWAVE 的视窗包含三个控制项：一个用於选择频率的水平卷动列，一个用於显示目前所选频率的静态文字区域，以及一个标记为「Turn On」的按钮。按下此按钮後，您将从连结音效卡的扩音器中听到正弦波的声音，同时按钮上的文字将变成「Turn Off」。用键盘或者滑鼠移动卷动列可以改变频率。要关闭声音，可以再次按下按钮。

SINEWAVE 程式码初始化卷动列，以便频率在 WM_INITDIALOG 讯息处理期间最低是 20Hz，最高是 5000Hz。初始化时，卷动列设定为 440 Hz。用音乐术语来说就是中音上面的A,它在管弦乐队演奏时用来调音。DlgProc 在接收 WM_HSCROLL 讯息处理期间改变静态变数 iFreq。注意，Page Left 和 Page Right 将导致 DlgProc 增加或者减少一个八度音阶。

当 DlgProc 从按钮收到一个 WM_COMMAND 讯息时，它首先配置 4 个记忆体块——2 个用於 WAVEHDR 结构，我们马上讨论。另两个用於缓冲区储存波形资料，我们将这两个缓冲区称为 pBuffer1 和 pBuffer2。

通过呼叫 waveOutOpen 函式，SINEWAVE 打开波形声音设备以便输出，waveOutOpen 函式使用下面的参数：

```
waveOutOpen (&hWaveOut, wDeviceID, &waveformat, dwCallBack,
                              dwCallBackData, dwFlags) ;
```

将第一个参数设定为指向 HWAVEOUT（handle to waveform audio output：波形声音输出代号）型态的变数。从函式传回时，此变数将设定为一个代号，後面的波形输出呼叫中将使用该代号。

waveOutOpen 的第二个参数是设备 ID。它允许函式可以在安装多个音效卡的机器上使用。参数的范围在 0 到系统所安装的波形输出设备数之间。呼叫 waveOutGetNumDevs 可以获得波形输出设备数，而呼叫 waveOutGetDevCaps 可以找出每个波形输出设备。如果想消除设备问号，那么您可以用常数 WAVE_MAPPER（定义为-1）来选择设备，该设备在「控制台」的「多媒体」中「音效」页面标签里的「喜欢使用的装置」中指定。另外，如果首选设备不能满足您的需要，而其他设备可以，那么系统将选择其他设备。

第三个参数是指向 WAVEFORMATEX 结构的指标（後面将详细介绍）。第四个参数是视窗代号或指向动态连结程式库中 callback 函式的指标，用来表示接收波形输出讯息的视窗或者 callback 函式。使用 callback 函式时，可在第五个参数中指定程式定义的资料。dwFlags 参数可设为 CALLBACK_WINDOW 或

CALLBACK_FUNCTION，以表示第四个参数的型态。您也可用 WAVE_FORMAT_QUERY 标记来检查能否打开设备（实际上并不打开它）。还有其他几个标记可用。

waveOutOpen 的第三个参数定义为指向 WAVEFORMATEX 型态结构的指标，此结构在 MMSYSTEM.H 中定义如下：

```
typedef struct waveformat_tag
{
      WORD  wFormatTag ;           // waveform format = WAVE_FORMAT_PCM
      WORD  nChannels ;                  // number of channels = 1 or 2
      DWORD       nSamplesPerSec ;             // sample rate
      DWORD       nAvgBytesPerSec ;          // bytes per second
      WORD  nBlockAlign ;                // block alignment
      WORD  wBitsPerSample ;             // bits per samples = 8 or 16
      WORD  cbSize ;                     // 0 for PCM
}
WAVEFORMATEX, * PWAVEFORMATEX ;
```

您可用此结构指定取样频率（nSamplesPerSec）和取样精确度（nBitsPerSample），以及选择单声道或立体声（nChannels）。结构中有些资讯看起来是多余的，但该结构也可用於非 PCM 的取样方式。在非 PCM 取样方式下，此结构的最後一个栏位设定为非 0 值，并带有其他资讯。

对於 PCM 取样方式，nBlockAlign 栏位设定为 nChannels 乘以 wBitsPerSample 再除以 8 所得到的数值，它表示每次取样的总位元组数。nAvgBytesPerSec 栏位设定为 nSamplesPerSec 和 nBlockAlign 的乘积。

SINEWAVE 初始化 WAVEFORMATEX 结构的栏位，并呼叫 waveOutOpen 函式：

```
waveOutOpen (    &hWaveOut, WAVE_MAPPER, &waveformat,
             (DWORD) hwnd, 0, CALLBACK_WINDOW)
```

如果呼叫成功，则 waveOutOpen 函式传回 MMSYSERR_NOERROR（定义为 0），否则传回非 0 的错误代码。如果 waveOutOpen 的传回值非 0，则 SINEWAVE 清除视窗，并显示一个标识错误的讯息方块。

现在设备打开了，SINEWAVE 继续初始化两个 WAVEHDR 结构的栏位，这两个结构用於在 API 中传递缓冲。WAVEHDR 定义如下：

```
typedef struct wavehdr_tag
{
      LPSTR lpData;                                          // pointer to data
buffer
         DWORD dwBufferLength;                                     // length of
data buffer
      DWORD dwBytesRecorded;                                 // used for recorded
      DWORD dwUser;                                          // for program use
      DWORD dwFlags;                                         // flags
      DWORD dwLoops;                                         //    number    of
repetitions
```

```
      struct wavehdr_tag FAR *lpNext;                // reserved
      DWORD reserved;                                     // reserved
}
WAVEHDR, *PWAVEHDR ;
```

SINEWAVE 将 lpData 栏位设定为包含资料的缓冲区位址，dwBufferLength 栏位设定为此缓冲区的大小，dwLoops 栏位设定为 1，其他栏位都设定为 0 或 NULL。如果要重复回圈播放声音，可设定 dwFlags 和 dwLoops 栏位。

SINEWAVE 下一步为两个资讯表头呼叫 waveOutPrepareHeader 函式，以防止结构和缓冲区与磁碟发生资料交换。

到此为止，所有的这些准备都是回应单击开启声音的按钮。但在程式的讯息伫列里已经有一个讯息在等待回应。因为我们已经在函式 waveOutOpen 中指定要用一个视窗讯息处理程式来接收波形输出讯息，所以 waveOutOpen 函式向程式的讯息伫列发送了 MM_WOM_OPEN 讯息，wParam 讯息参数设定为波形输出代号。要处理 MM_WOM_OPEN 讯息，SINEWAVE 呼叫 FillBuffer 函式两次，并用正弦波形资料填充 pBuffer 缓冲区。然後 SINEWAVE 把两个 WAVEHDR 结构传送给 waveOutWrite，此函式将资料传送到波形输出硬体，才真正开始播放声音。

当波形硬体播放完 waveOutWrite 函式传送来的资料後，就向视窗发送 MM_WOM_DONE 讯息，其中 wParam 参数是波形输出代号，lParam 是指向 WAVEHDR 结构的指标。SINEWAVE 在处理此讯息时，将计算缓冲区的新资料，并呼叫 waveOutWrite 来重新提交缓冲区。

编写 SINEWAVE 程式时也可以只用一个 WAVEHDR 结构和一个缓冲区。不过，这样在播放完资料後将会有很短暂的停顿，以等待程式处理 MM_WOM_DONE 讯息来提交新的缓冲区。SINEWAVE 使用的「双缓冲」技术避免了声音的不连续。

当使用者单击「Turn Off」按钮关闭声音时，DlgProc 接收到另一个 WM_COMMAND 讯息。对此讯息，DlgProc 把 bShutOff 变数设定为 TRUE，并呼叫 waveOutReset 函式。此函式停止处理声音并发送一条 MM_WOM_DONE 讯息。bShutOff 为 TRUE 时，SINEWAVE 透过呼叫 waveOutClose 来处理 MM_WOM_DONE，从而产生一条 MM_WOM_CLOSE 讯息。处理 MM_WOM_CLOSE 通常包括清除程序。SINEWAVE 为两个 WAVEHDR 结构而呼叫 waveOutUnprepareHeader、释放所有的记忆体块并把按钮上的文字改回「Turn On」。

如果硬体继续播放缓冲区的声音资料，那么它自己呼叫 waveOutClose 就没有作用。您必须先呼叫 waveOutReset 来停止播放并产生 MM_WOM_DONE 讯息。当 wParam 是 SC_CLOSE 时，DlgProc 也处理 WM_SYSCOMMAND 讯息，这是因为使用者从系统功能表中选择了「Close」。如果波形声音继续播放，DlgProc 则呼叫 waveOutReset。无论如何，最後总要呼叫 EndDialog 来结束程式。

数位录音机

Windows 提供了一个称为「录音程式」来录制和播放数位声音。程式 22-3 所示的程式（RECORD1）不如「录音程式」完善，因为它不含有任何档案 I/O，也不允许声音编辑。然而，这个程式显示了使用低阶波形声音 API 来录制和重播声音的基本方法。

程式 22-3　RECORD1

```
RECORD1.C
/*------------------------------------------------------------------------
-
      RECORD1.C -- Waveform Audio Recorder
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------
*/

#include <windows.h>
#include "resource.h"

#define INP_BUFFER_SIZE 16384
BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record1") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
      {
                  MessageBox (     NULL, TEXT ("This program requires Windows
NT!"),
                                                          szAppName,
MB_ICONERROR) ;
      }
      return 0 ;
}

void ReverseMemory (BYTE * pBuffer, int iLength)
{
      BYTE  b ;
      int   i ;

      for (i = 0 ; i < iLength / 2 ; i++)
      {
                  b = pBuffer [i] ;
                  pBuffer [i] = pBuffer [iLength - i - 1] ;
                  pBuffer [iLength - i - 1] = b ;
      }
```

```
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static      BOOL                        bRecording, bPlaying, bReverse,
bPaused,
                                                            bEnding,
bTerminating ;
      static      DWORD                       dwDataLength, dwRepetitions = 1 ;
      static      HWAVEIN               hWaveIn ;
      static      HWAVEOUT              hWaveOut ;
      static      PBYTE                       pBuffer1, pBuffer2, pSaveBuffer,
pNewBuffer ;
      static      PWAVEHDR              pWaveHdr1, pWaveHdr2 ;
      static            TCHAR                 szOpenError[]  =  TEXT  ("Error
opening waveform audio!");
      static            TCHAR                 szMemError  []  =  TEXT  ("Error
allocating memory!") ;
      static      WAVEFORMATEX waveform ;

      switch (message)
     {
      case  WM_INITDIALOG:
                                    // Allocate memory for wave header

                        pWaveHdr1 = malloc (sizeof (WAVEHDR)) ;
                        pWaveHdr2 = malloc (sizeof (WAVEHDR)) ;

                                    // Allocate memory for save buffer

                        pSaveBuffer = malloc (1) ;
                        return TRUE ;

      case  WM_COMMAND:
                        switch (LOWORD (wParam))
                        {
                        case  IDC_RECORD_BEG:
                                    // Allocate buffer memory

                                    pBuffer1 = malloc (INP_BUFFER_SIZE) ;
                                    pBuffer2 = malloc (INP_BUFFER_SIZE) ;

                                    if (!pBuffer1 || !pBuffer2)
                                    {
                                          if (pBuffer1) free (pBuffer1) ;
                                          if (pBuffer2) free (pBuffer2) ;
```

```
                                    MessageBeep (MB_ICONEXCLAMATION) ;
                                    MessageBox     (hwnd,     szMemError,
szAppName,
                                MB_ICONEXCLAMATION | MB_OK) ;
                                    return TRUE ;
                                }

                                    // Open waveform audio for input

                    waveform.wFormatTag         = WAVE_FORMAT_PCM ;
                    waveform.nChannels          = 1 ;
                    waveform.nSamplesPerSec           = 11025 ;
                    waveform.nAvgBytesPerSec    = 11025 ;
                    waveform.nBlockAlign              = 1 ;
                    waveform.wBitsPerSample           = 8 ;
                    waveform.cbSize                   = 0 ;

               if (waveInOpen (&hWaveIn, WAVE_MAPPER, &waveform,
                        (DWORD) hwnd, 0, CALLBACK_WINDOW))
                                {
                                    free (pBuffer1) ;
                                    free (pBuffer2) ;
                                    MessageBeep (MB_ICONEXCLAMATION) ;
                                MessageBox (hwnd, szOpenError, szAppName,
                                MB_ICONEXCLAMATION | MB_OK) ;
                                }
                                    // Set up headers and prepare them

                                pWaveHdr1->lpData          = pBuffer1 ;
                            pWaveHdr1->dwBufferLength    = INP_BUFFER_SIZE ;
                                pWaveHdr1->dwBytesRecorded   = 0 ;
                                pWaveHdr1->dwUser            = 0 ;
                                pWaveHdr1->dwFlags           = 0 ;
                                pWaveHdr1->dwLoops           = 1 ;
                                pWaveHdr1->lpNext            = NULL ;
                                pWaveHdr1->reserved          = 0 ;
        waveInPrepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;

                                pWaveHdr2->lpData        = pBuffer2 ;
                            pWaveHdr2->dwBufferLength  = INP_BUFFER_SIZE ;
                                pWaveHdr2->dwBytesRecorded        = 0 ;
                                pWaveHdr2->dwUser                 = 0 ;
                                pWaveHdr2->dwFlags                = 0 ;
                                pWaveHdr2->dwLoops                = 1 ;
                                pWaveHdr2->lpNext           = NULL ;
                                pWaveHdr2->reserved               = 0 ;

                                waveInPrepareHeader  (hWaveIn,  pWaveHdr2,
```

```
sizeof (WAVEHDR)) ;
                                return TRUE ;

                case  IDC_RECORD_END:
                                      // Reset input to return last buffer

                                bEnding = TRUE ;
                                waveInReset (hWaveIn) ;
                                return TRUE ;

                case  IDC_PLAY_BEG:
                                      // Open waveform audio for output

                      waveform.wFormatTag         = WAVE_FORMAT_PCM ;
                                waveform.nChannels          = 1 ;
                                waveform.nSamplesPerSec           = 11025 ;
                                waveform.nAvgBytesPerSec    = 11025 ;
                                waveform.nBlockAlign              = 1 ;
                                waveform.wBitsPerSample           = 8 ;
                                waveform.cbSize                   = 0 ;

                      if (waveOutOpen (&hWaveOut, WAVE_MAPPER, &waveform,
                         (DWORD) hwnd, 0, CALLBACK_WINDOW))
                                {
                      MessageBeep (MB_ICONEXCLAMATION) ;
                      MessageBox (hwnd,szOpenError, szAppName,
                      MB_ICONEXCLAMATION | MB_OK) ;
                                }
                                return TRUE ;

                case  IDC_PLAY_PAUSE:
                                            // Pause or restart output

                                if (!bPaused)
                                {
           waveOutPause (hWaveOut) ;
           SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
                           bPaused = TRUE ;
                                }
                                else
                                {
           waveOutRestart (hWaveOut) ;
           SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
                           bPaused = FALSE ;
                                }
                                      return TRUE ;

                      case  IDC_PLAY_END:
```

```
                    // Reset output for close preparation

                                   bEnding = TRUE ;
                                   waveOutReset (hWaveOut) ;
                                   return TRUE ;

                    case  IDC_PLAY_REV:
                                          // Reverse save buffer and play

                                   bReverse = TRUE ;
                                ReverseMemory (pSaveBuffer, dwDataLength) ;

                                SendMessage (hwnd, WM_COMMAND, IDC_PLAY_BEG, 0) ;
                                   return TRUE ;

                    case  IDC_PLAY_REP:
                                   // Set infinite repetitions and play

                                   dwRepetitions = -1 ;
                                SendMessage (hwnd, WM_COMMAND, IDC_PLAY_BEG, 0) ;
                                   return TRUE ;

                    case  IDC_PLAY_SPEED:
                                   // Open waveform audio for fast output

                                waveform.wFormatTag    = WAVE_FORMAT_PCM ;
                                   waveform.nChannels     = 1 ;
                                   waveform.nSamplesPerSec =22050 ;
                                   waveform.nAvgBytesPerSec= 22050 ;
                                   waveform.nBlockAlign    = 1 ;
                                   waveform.wBitsPerSample = 8 ;
                                   waveform.cbSize        = 0 ;
             if (waveOutOpen (&hWaveOut,  0, &waveform, (DWORD) hwnd, 0,
            CALLBACK_WINDOW))
                                   {
            essageBeep (MB_ICONEXCLAMATION) ;
 MessageBox (hwnd, szOpenError, szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                                }
                    return TRUE ;
                    }
                    break ;

    case  MM_WIM_OPEN:
                                          // Shrink down the save buffer

                    pSaveBuffer = realloc (pSaveBuffer, 1) ;

                                          // Enable and disable buttons
```

```
                EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG),    FALSE) ;
                EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END),    TRUE)  ;
                EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG),      FALSE) ;
                EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE),    FALSE) ;
                EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),      FALSE) ;
                EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV),      FALSE) ;
                EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP),      FALSE) ;
                EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED),    FALSE) ;
                SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

                                        // Add the buffers

                     waveInAddBuffer (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
                     waveInAddBuffer (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

                                        // Begin sampling

                     bRecording = TRUE ;
                     bEnding = FALSE ;
                     dwDataLength = 0 ;
                     waveInStart (hWaveIn) ;
                     return TRUE ;

     case  MM_WIM_DATA:

                                        // Reallocate save buffer memory

                     pNewBuffer = realloc ( pSaveBuffer, dwDataLength +
((PWAVEHDR) lParam)->dwBytesRecorded) ;

                     if (pNewBuffer == NULL)
                     {
                                        waveInClose      (hWaveIn) ;
                                        MessageBeep
     (MB_ICONEXCLAMATION) ;
                                   MessageBox  (hwnd, szMemError, szAppName,
                          MB_ICONEXCLAMATION | MB_OK) ;
                                        return TRUE ;
                     }

           pSaveBuffer = pNewBuffer ;
           CopyMemory (pSaveBuffer + dwDataLength, ((PWAVEHDR) lParam)->lpData,
                ((PWAVEHDR) lParam)->dwBytesRecorded) ;

                     dwDataLength += ((PWAVEHDR) lParam)->dwBytesRecorded ;

                     if (bEnding)
```

```
                    {
                              waveInClose (hWaveIn) ;
                              return TRUE ;
                    }

                         // Send out a new buffer

          waveInAddBuffer (hWaveIn, (PWAVEHDR) lParam, sizeof (WAVEHDR)) ;
          return TRUE ;

     case  MM_WIM_CLOSE:
                              // Free the buffer memory

               waveInUnprepareHeader (hWaveIn, pWaveHdr1, sizeof (WAVEHDR)) ;
               waveInUnprepareHeader (hWaveIn, pWaveHdr2, sizeof (WAVEHDR)) ;

                    free (pBuffer1) ;
                    free (pBuffer2) ;

                              // Enable and disable buttons

                    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
                    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), FALSE) ;
                    SetFocus (GetDlgItem (hwnd, IDC_RECORD_BEG)) ;

                    if (dwDataLength > 0)
                    {
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG),      TRUE) ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE),    FALSE) ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),      FALSE) ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REP),      TRUE)  ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV),      TRUE)  ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED),    TRUE)  ;
               SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;
                    }
                    bRecording = FALSE ;

                    if (bTerminating)
                         SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

                    return TRUE ;

     case  MM_WOM_OPEN:
                              // Enable and disable buttons

          EnableWindow     (GetDlgItem (hwnd, IDC_RECORD_BEG),     FALSE) ;
          EnableWindow     (GetDlgItem (hwnd, IDC_RECORD_END),     FALSE) ;
          EnableWindow     (GetDlgItem (hwnd, IDC_PLAY_BEG),       FALSE) ;
```

```
          EnableWindow     (GetDlgItem (hwnd, IDC_PLAY_PAUSE),TRUE);
          EnableWindow     (GetDlgItem (hwnd, IDC_PLAY_END),       TRUE);
          EnableWindow     (GetDlgItem (hwnd, IDC_PLAY_REP),       FALSE) ;
          EnableWindow     (GetDlgItem (hwnd, IDC_PLAY_REV),       FALSE) ;
          EnableWindow     (GetDlgItem (hwnd, IDC_PLAY_SPEED),     FALSE) ;
          SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

                                // Set up header

                    pWaveHdr1->lpData            = pSaveBuffer ;
                    pWaveHdr1->dwBufferLength    = dwDataLength ;
                    pWaveHdr1->dwBytesRecorded   = 0 ;
                    pWaveHdr1->dwUser            = 0 ;
                    pWaveHdr1->dwFlags     = WHDR_BEGINLOOP | WHDR_ENDLOOP ;
                    pWaveHdr1->dwLoops           = dwRepetitions ;
                    pWaveHdr1->lpNext            = NULL ;
                    pWaveHdr1->reserved          = 0 ;

                                // Prepare and write

               waveOutPrepareHeader (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;
                    waveOutWrite (hWaveOut, pWaveHdr1, sizeof (WAVEHDR)) ;

                    bEnding = FALSE ;
                    bPlaying = TRUE ;
                    return TRUE ;

     case  MM_WOM_DONE:
                    waveOutUnprepareHeader  (hWaveOut,  pWaveHdr1,  sizeof
(WAVEHDR)) ;
                    waveOutClose (hWaveOut) ;
                    return TRUE ;

     case  MM_WOM_CLOSE:
                                // Enable and disable buttons

               EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG),    TRUE)  ;
               EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END),    TRUE)  ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG),      TRUE)  ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE),    FALSE) ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),      FALSE) ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_REV),      TRUE) ;
               EnableWindow(GetDlgItem (hwnd, IDC_PLAY_REP),       TRUE);
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_SPEED),    TRUE) ;
               SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

                    SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
                    bPaused = FALSE ;
```

```
                        dwRepetitions = 1 ;
                        bPlaying = FALSE ;

                        if (bReverse)
                        {
                                    ReverseMemory (pSaveBuffer, dwDataLength) ;
                                    bReverse = FALSE ;
                        }

                        if (bTerminating)
                              SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;

                        return TRUE ;

      case  WM_SYSCOMMAND:
                        switch (LOWORD (wParam))
                        {
                        case  SC_CLOSE:
                                          if (bRecording)
                                          {
                                                bTerminating = TRUE ;
                                                bEnding = TRUE ;
                                                waveInReset (hWaveIn) ;
                                                return TRUE ;
                                          }
                                          if (bPlaying)
                                          {
                                                bTerminating = TRUE ;
                                                bEnding = TRUE ;
                                                waveOutReset (hWaveOut) ;
                                                return TRUE ;
                                          }

                                          free (pWaveHdr1) ;
                                          free (pWaveHdr2) ;
                                          free (pSaveBuffer) ;
                                          EndDialog (hwnd, 0) ;
                                          return TRUE ;
                              }
                              break ;
      }
      return FALSE ;
}
```

RECORD.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"
```

```
//////////////////////////////////////////////////////////////////////////
/
// Dialog
RECORD            DIALOG DISCARDABLE  100, 100, 152, 74
STYLE                   WS_MINIMIZEBOX | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION           "Waveform Audio Recorder"
FONT 8,           "MS Sans Serif"
BEGIN
      PUSHBUTTON             "Record",IDC_RECORD_BEG,28,8,40,14
      PUSHBUTTON             "End",IDC_RECORD_END,76,8,40,14,WS_DISABLED
      PUSHBUTTON             "Play",IDC_PLAY_BEG,8,30,40,14,WS_DISABLED
      PUSHBUTTON             "Pause",IDC_PLAY_PAUSE,56,30,40,14,WS_DISABLED
      PUSHBUTTON             "End",IDC_PLAY_END,104,30,40,14,WS_DISABLED
      PUSHBUTTON             "Reverse",IDC_PLAY_REV,8,52,40,14,WS_DISABLED
      PUSHBUTTON             "Repeat",IDC_PLAY_REP,56,52,40,14,WS_DISABLED
      PUSHBUTTON
      "Speedup",IDC_PLAY_SPEED,104,52,40,14,WS_DISABLED
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Record.rc

#define           IDC_RECORD_BEG                                  1000
#define           IDC_RECORD_END                                  1001
#define           IDC_PLAY_BEG                                    1002
#define           IDC_PLAY_PAUSE                                  1003
#define           IDC_PLAY_END                                    1004
#define           IDC_PLAY_REV                                    1005
#define           IDC_PLAY_REP                                    1006
#define           IDC_PLAY_SPEED                                  1007
```

RECORD.RC 和 RESOURCE.H 档案也在 RECORD2 和 RECORD3 程式中使用。

RECORD1 视窗有 8 个按钮。第一次执行 RECORD1 时，只有「Record」按钮有效。按下「Record」後，就开始录音，这时「Record」按钮无效，而「End」按钮有效。按下「End」可停止录音。这时，「Play」、「 Reverse」、「 Repeat」和「Speedup」也都有效，选择任一个按钮都可重放声音：「Play」表示正常播放；「Reverse」表示反向播放；「Repeat」表示无限的重复播放（好像回圈录音带）；「Speedup」以正常速度的两倍来播放。要停止播放，您可以选择「End」按钮，而按下「Pause」按钮可停止播放。按下後，「Pause」按钮将变为「Resume」按钮，用於继续播放声音。如果要录制另一段声音，新录制的声音将替换记忆体里现有的声音。

任何时候，有效按钮都是可以执行有效操作的按钮。这需要在 RECORD1 原始码中包括对 EnableWindow 的多次呼叫，但是程式并不检查具体的按钮操作是否有效。显然，这使得程式操作更为直观。

RECORD1 用了许多快捷方式来简化程式码。首先，如果安装了多个波形声音硬体设备，则 RECORD1 只使用内定设备。其次，程式按标准的 11.025 kHz 的取样频率和 8 位元的取样精确度来录音和放音，而不管设备能否提供更高的取样频率和取样精确度。唯一的例外是加速功能，加速时 RECORD1 按 22.050kHz 的取样频率播放声音，这样不仅播放速度提高了一倍，而且频率也提高了一个音阶。

录制声音既包括为输入而打开波形声音硬体，还包括将缓冲区传递给 API，以便接收声音资料。

RECORD1 设有几个记忆体块。其中三个很小，至少在初始化时很小，并且在 DlgProc 的 WM_INITDIALOG 讯息处理期间进行配置。程式配置两个 WAVEHDR 结构，分别由指标 pWaveHdr1 和 pWaveHdr2 指向。这两个结构用於将缓冲区传递给波形 API。pSaveBuffer 指标指向储存整个录音的缓冲区，最初配置时只有一个位元组。然後，随著录音的进行，该缓冲区不断增大，以适应所有的声音资料（如果录音时间过长，则 RECORD1 能够在录制程序中及时发现记忆体溢出，并允许您重放成功储存的声音）。由於这个缓冲区用来储存堆积的声音资料，所以我将其称为「储存缓冲区（save buffer）」。指标 pBuffer1 和 pBuffer2 指向的另外两个记忆体块，大小是 16K，它们在记录接收的声音资料时配置。录音结束後释放这些记忆体块。

8 个按钮中的每一个都向 REPORT1 视窗的对话程序 DlgProc 产生 WM_COMMAND 讯息。最初只有「Record」按钮有效。按下此按钮将产生 WM_COMMAND 讯息，其中 wParam 参数等於 IDC_RECORD_BEG。为处理这个讯息，RECORD1 配置两个 16K 的缓冲区来接收声音资料，初始化 WAVEFORMATEX 结构的栏位，并将此结构传递给 waveInOpen 函式，然後设定两个 WAVEHDR 结构。

waveInOpen 函式产生一条 MM_WIM_OPEN 讯息。在此讯息处理期间，RECORD1 把储存缓冲区的大小缩减到 1 个位元组，以准备接收资料（当然，第一次录音时，储存缓冲区的大小就是 1 个位元组，但以後录制时，就可能大多了）。在 MM_WIM_OPEN 讯息处理期间，RECORD1 也将适当的按钮设定为有效和无效。然後，程式用 waveInAddBuffer 把两个 WAVEHDR 结构和缓冲区传送给 API。这时会设定某些标记，然後呼叫 waveInStart 开始录音。

采用 11.025kHz 的取样频率和 8 位元的取样精确度时，16K 的缓冲区可储存大约 1.5 秒的声音。这时，RECORD1 接收 MM_WIM_DATA 讯息。在回应此讯息处理期间，程式将根据变数 dwDataLength 和 WAVEHDR 结构中的栏位 dwBytesRecorded 对缓冲区重新配置。如果配置失败，RECORD1 呼叫 waveInClose 来停止录音。

如果重新配置成功，则 RECORD1 把 16K 缓冲区里的资料复制到储存缓冲区，

然後再次呼叫 waveInAddBuffer。此程序将持续到 RECORD1 用完储存缓冲区的记忆体，或使用者按下「End」按钮为止。

「End」按钮产生 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_END。处理这个讯息很简单，RECORD1 把 bEnding 标记设定为 TRUE 并呼叫 waveInReset。waveInReset 函式使录音停止，并产生 MM_WIM_DATA 讯息，该讯息含有部分填充的缓冲区。除了呼叫 waveInClose 来关闭波形输入设备外，RECORD1 对这个讯息正常回应。

waveInClose 产生 MM_WIM_CLOSE 讯息。RECORD1 回应此讯息时，释放 16K 输入缓冲区，并使相应的按钮有效或无效。尤其是，当储存缓冲区里存有资料（除非第一次配置就失败，否则一般都含有资料）时，播放按钮将有效。

录音以後，储存缓冲区里将含有这些声音资料。当使用者选择「Play」按钮时，DlgProc 就接收一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_PLAY_BEG。回应时，程式将初始化 WAVEFORMATEX 结构的栏位，并呼叫 waveOutOpen。

waveOutOpen 呼叫再次产生 MM_WOM_OPEN 讯息，在此讯息处理期间，RECORD1 把相应的按钮设为有效或无效（只允许使用「Pause」和「End」），用储存缓冲区来初始化 WAVEHDR 结构的栏位，呼叫 waveOutPrepareHeader 来准备要播放的声音，然後呼叫 waveOutWrite 开始播放。

一般情况下，直到播放完储存缓冲区里的所有资料才停止。这时产生 MM_WOM_DONE 讯息。如果还有缓冲区要播放，则程式会在这时将它们传递给 API。由於 RECORD1 只播放一个大缓冲区，因此程式不再简单地准备标题，而是呼叫 waveOutClose。waveOutClose 函式产生 MM_WOM_CLOSE 讯息。在此讯息处理期间，RECORD1 使相应的按钮有效或无效，并允许声音再次播放或者录制新声音。

程式中还有一个「End」按钮，利用此按钮，使用者可以在播放完储存缓冲区之前的任何时刻停止播放。「End」按钮产生一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_PLAY_END，回应时，程式呼叫 waveOutReset，此函式产生一条正常处理的 MM_WOM_DONE 讯息。

RECORD1 的视窗中还包括一个「Pause」按钮。处理此按钮很简单：第一次按时下，RECORD1 呼叫 waveOutPause 来暂停播放，并将按钮上的文字改为「Resume」。按下「Resume」按钮时，通过呼叫 waveOutRestart 来继续播放。

为了使程式更有趣，视窗中还包括另外三个按钮：「Reverse」、「Repeat」和「Speedup」。这些按钮都产生 WM_COMMAND 讯息，其中 wParam 的值分别等於 IDC_PLAY_REV、IDC_PLAY_REP 和 IDC_PLAY_SPEED。

倒放声音就是把储存缓冲区里的资料按位元组顺序反向，然後再正常播放。RECORD1 中有一个称为 ReverseMemory 的小函式使位元组反向。在 WM_COMMAND

讯息处理期间，程式在播放块之前呼叫此函式，并在 MM_WOM_CLOSE 讯息的後期再次呼叫此函式，以便将其恢复到正常状态。

「Repeat」按钮将往复不停地播放声音。由於 API 支援重复播放声音，所以这并不复杂。只要将 WAVEHDR 结构的 dwLoops 栏位设为重复次数，将 dwFlags 栏位设为 WHDR_BEGINLOOP 和 WHDR_ENDLOOP，分别表示回圈时缓冲区的开始部分和结束部分。因为 RECORD1 只使用一个缓冲区来播放声音，所以这两个标记组合到了 dwFlags 栏位。

要实作两倍速播放也很容易。在准备为输出而打开波形声音期间，初始化 WAVEFORMATEX 结构的栏位时，只需将 nSamplesPerSec 和 nAvgBytesPerSec 栏位设定为 22050，而不是 11025。

另一种 MCI 介面

您可能已经发现，RECORD1 很复杂。特别是在处理波形声音函式呼叫和它们产生的讯息间的交互时，更复杂。处理可能出现的记忆体不足的情况也是如此。但这也许正是它称为低阶介面的原因。我在本章的前面提到过，Windows 也提供高阶媒体控制介面（Media Control Interface）。

对波形声音来说，低阶介面与 MCI 之间的主要区别在於 MCI 用波形档案记录声音资料，并通过读取档案来播放声音。由於在播放声音之前要读取档案、处理档案然後再写入档案，所以让 RECODE1 来实作「特殊效果」很困难。这是典型的折衷选择问题：功能齐全或是使用方便？低阶介面很灵活，但 MCI（其中的大部分）更方便。

MCI 有两种不同但又相关的实作形式。一种形式用讯息和资料结构将命令发送给多媒体设备，然後再从那里接收资讯。另一种形式使用 ASCII 文字字串。建立文字命令的介面最初是为了让多媒体设备接受简单的描述命令语言的控制。但它也提供非常容易的交谈式控制，请参见本章前面，TESTMCI 程式的展示。

RECORD2 程式，如程式 22-4 所示，使用 MCI 形式的讯息和资料结构来实作另一个数位声音录音机和播放器。虽然它使用的对话方块模板与 RECORD1 一样，但并没有实作三个特殊效果的按钮。

程式 22-4　RECORD2

```
RECORD2.C
/*------------------------------------------------------------------------
-
      RECORD2.C -- Waveform Audio Recorder
                                              (c) Charles Petzold, 1998
 ------------------------------------------------------------------------*/
```

```
#include <windows.h>
#include "..\\record1\\resource.h"

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record2") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                    szAppName, MB_ICONERROR) ;
      }
      return 0 ;
}

void ShowError (HWND hwnd, DWORD dwError)
{
      TCHAR szErrorStr [1024] ;
      mciGetErrorString (dwError, szErrorStr, sizeof (szErrorStr) / sizeof
(TCHAR)) ;
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szErrorStr, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static      BOOL    bRecording, bPlaying, bPaused ;
      static      TCHAR   szFileName[] = TEXT ("record2.wav") ;
      static      WORD    wDeviceID ;
      DWORD                   dwError ;
      MCI_GENERIC_PARMS           mciGeneric ;
      MCI_OPEN_PARMS              mciOpen ;
      MCI_PLAY_PARMS              mciPlay ;
      MCI_RECORD_PARMS            mciRecord ;
      MCI_SAVE_PARMS              mciSave ;

      switch (message)
     {
      case  WM_COMMAND:
                        switch (wParam)
                        {
                        case  IDC_RECORD_BEG:
                                                // Delete existing waveform file

                                          DeleteFile (szFileName) ;
```

```
                                    // Open waveform audio

                    mciOpen.dwCallback          = 0 ;
                    mciOpen.wDeviceID           = 0 ;
                    mciOpen.lpstrDeviceType=TEXT ("waveaudio") ;
                    mciOpen.lpstrElementName    = TEXT ("") ;
                    mciOpen.lpstrAlias                = NULL ;
                    dwError = mciSendCommand (0, MCI_OPEN,
                  MCI_WAIT | MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                  (DWORD) (LPMCI_OPEN_PARMS) &mciOpen) ;
                                if (dwError != 0)
                                {
                                      ShowError (hwnd, dwError) ;
                                      return TRUE ;
                                }
                                      // Save the Device ID

                                wDeviceID = mciOpen.wDeviceID ;

                                      // Begin recording

                          mciRecord.dwCallback   = (DWORD) hwnd ;
                          mciRecord.dwFrom = 0 ;
                          mciRecord.dwTo              = 0 ;

                    mciSendCommand (wDeviceID, MCI_RECORD, MCI_NOTIFY,
                     (DWORD) (LPMCI_RECORD_PARMS) &mciRecord) ;

                    // Enable and disable buttons

              EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
              EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
              EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG),   FALSE);
              EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
              EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),   FALSE);
              SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

                                bRecording = TRUE ;
                                return TRUE ;

              case  IDC_RECORD_END:
                                                  // Stop recording

              mciGeneric.dwCallback = 0 ;

              mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT,
        (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;
```

```
                                                    // Save the file

                    mciSave.dwCallback = 0 ;
                    mciSave.lpfilename = szFileName ;

                    mciSendCommand   (wDeviceID,   MCI_SAVE,   MCI_WAIT   |
MCI_SAVE_FILE,
                (DWORD) (LPMCI_SAVE_PARMS) &mciSave) ;

                                                //Closethewaveformdevice

                    mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT,
                (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

                                                // Enable and disable buttons

          EnableWindow (GetDlgItem      (hwnd, IDC_RECORD_BEG),      TRUE);
          EnableWindow (GetDlgItem      (hwnd, IDC_RECORD_END),      FALSE);
          EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_BEG),        TRUE);
          EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_PAUSE),      FALSE);
          EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_END),      FALSE);
          SetFocus            (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

          bRecording = FALSE ;
          return TRUE ;

                    case  IDC_PLAY_BEG:
                                                // Open waveform audio

                              mciOpen.dwCallback                = 0 ;
                              mciOpen.wDeviceID                 = 0 ;
                              mciOpen.lpstrDeviceType           = NULL ;
                              mciOpen.lpstrElementName     = szFileName ;
                              mciOpen.lpstrAlias                = NULL ;

                              dwError = mciSendCommand (   0, MCI_OPEN,
                         MCI_WAIT | MCI_OPEN_ELEMENT,
                         (DWORD) (LPMCI_OPEN_PARMS) &mciOpen) ;

                                   if (dwError != 0)
                                   {
                                         ShowError (hwnd, dwError) ;
                                         return TRUE ;
                                   }
                                                // Save the Device ID

                                   wDeviceID = mciOpen.wDeviceID ;
```

```
                                        // Begin playing

                        mciPlay.dwCallback     = (DWORD) hwnd ;
                        mciPlay.dwFrom              = 0 ;
                        mciPlay.dwTo                = 0 ;

                    mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY,
                     (DWORD) (LPMCI_PLAY_PARMS) &mciPlay) ;

                                      // Enable and disable buttons

  EnableWindow     (GetDlgItem      (hwnd, IDC_RECORD_BEG), FALSE);
 EnableWindow (GetDlgItem      (hwnd, IDC_RECORD_END), FALSE);
 EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_BEG),  FALSE);
 EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_PAUSE), TRUE) ;
 EnableWindow (GetDlgItem      (hwnd, IDC_PLAY_END),   TRUE) ;
 SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

 bPlaying = TRUE ;
 return TRUE ;

             case  IDC_PLAY_PAUSE:
                             if (!bPaused)
                                        // Pause the play
                        {
        mciGeneric.dwCallback = 0 ;

        mciSendCommand (wDeviceID, MCI_PAUSE, MCI_WAIT,
           (DWORD) (LPMCI_GENERIC_PARMS) & mciGeneric);

SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
        Paused = TRUE ;
                        }
                        else
                                        // Begin playing again
                        {
             mciPlay.dwCallback          = (DWORD) hwnd ;
             mciPlay.dwFrom                        = 0 ;
             mciPlay.dwTo                          = 0 ;

             mciSendCommand (wDeviceID, MCI_PLAY, MCI_NOTIFY,
              (DWORD) (LPMCI_PLAY_PARMS) &mciPlay) ;

SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
             bPaused = FALSE ;
                        }
```

```
                                return TRUE ;

                case  IDC_PLAY_END:
                                                // Stop and close

                    mciGeneric.dwCallback = 0 ;

                    mciSendCommand (wDeviceID, MCI_STOP, MCI_WAIT,
                (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

                    mciSendCommand (wDeviceID, MCI_CLOSE, MCI_WAIT,
                (DWORD) (LPMCI_GENERIC_PARMS) &mciGeneric) ;

                 // Enable and disable buttons

                    EnableWindow (GetDlgItem(hwnd,IDC_RECORD_BEG),      TRUE) ;
                    EnableWindow (GetDlgItem(hwnd, IDC_RECORD_END),     FALSE);
                    EnableWindow(GetDlgItem(hwnd, IDC_PLAY_BEG),  TRUE) ;
                    EnableWindow (GetDlgItem(hwnd, IDC_PLAY_PAUSE),     FALSE);
                    EnableWindow (GetDlgItem     (hwnd, IDC_PLAY_END),  FALSE);
                    SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

                    bPlaying = FALSE ;
                    bPaused  = FALSE ;
                    return TRUE ;
                          }
                          break ;

    case  MM_MCINOTIFY:
                          switch (wParam)
                          {
                          case  MCI_NOTIFY_SUCCESSFUL:
                                                if (bPlaying)
                    SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0) ;

                                                if (bRecording)
                    SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0);

                                                return TRUE ;
                          }
                          break ;

    case  WM_SYSCOMMAND:
                          switch (wParam)
                          {
                          case  SC_CLOSE:
                                                if (bRecording)
                          SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0L) ;
```

```
                                        if (bPlaying)
                            SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0L) ;

                                        EndDialog (hwnd, 0) ;
                                        return TRUE ;
                            }
                            break ;
        }
          return FALSE ;
}
```

RECORD2 只使用两个 MCI 函式呼叫，其中最重要的呼叫如下所示：

```
error = mciSendCommand (wDeviceID, message, dwFlags, dwParam)
```

第一个参数是设备的识别数字（ID），您可以按代号来使用 ID。打开设备时就可以获得 ID，并在随後的 mciSendCommand 呼叫中使用。第二个参数是字首为 MCI 的常数。这些称为 MCI 命令讯息，RECORD2 展示了其中的七个：MCI_OPEN、MCI_RECORD、MCI_STOP、MCI_SAVE、MCI_PLAY、MCI_PAUSE 和 MCI_CLOSE。

dwFlags 参数通常由 0 或者多个位元旗标常数（由 C 的位元 OR 运算子合成）组成。这些通常用来表示不同的选项。一些选项是某个命令讯息所特有的，而另一些对所有的讯息都是通用的。dwParam 参数通常是指向一个资料结构的长指标，该结构表示选项以及由设备获得的资讯。许多 MCI 讯息都与资料结构有关，而且这些资料结构对於讯息来说都是唯一的。

如果 mciSendCommand 函式呼叫成功，则传回 0 值，否则传回错误代码。要向使用者报告此错误，可用下面的函式获得描述错误的文字字串：

```
mciGetErrorString (error, szBuffer, dwLength)
```

此函式在程式 TESTMCI 中也用到过。

按下「Record」按钮後，RECORD2 的视窗讯息处理程式就收到一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_BEG。RECORD2 从打开设备开始，包括设定 MCI_OPEN_PARMS 结构的栏位，并用 MCI_OPEN 命令讯息呼叫 mciSendCommand。录音时，lpstrDeviceType 栏位设定为字串「waveaudio」以说明设备型态，lpstrElementName 栏位设定为长度为 0 的字串。MCI 驱动程式使用内定的取样频率和取样精确度，但是您可以用 MCI_SET 命令进行修改。录音程序中，声音资料先储存在硬碟上的暂存档案中，最後再转化成标准的波形档案。本章的後面将介绍波形档案的格式。播放录制的声音时，MCI 使用波形档案中定义的取样频率和取样精确度。

如果 RECORD2 不能打开设备，则用 mciGetErrorString 和 MessageBox 提示错误资讯。否则从 mciSendCommand 呼叫传回，MCI_OPEN_PARMS 结构的 wDeviceID 栏位包含有设备 ID，以供後面的呼叫使用。

要开始录音，RECORD2 就呼叫 mciSendCommand，以 MCI_RECORD 命令讯息和

MCI_WAVE_RECORD_PARMS 资料结构为参数。当然，您也可以将此结构（并使用表示这些栏位已设定的位元旗标）的 dwFromz 和 dwTo 栏位进行设定，以便将声音插入现有的波形档案，其档案名在 MCI_OPEN_PARMS 结构的 lpstrElementName 栏位指定。内定状态下，任何新的声音都插入在现有档案的开始位置。

RECORD2 将 MCI_WAVE_RECORD_PARMS 结构的 dwCallback 栏位设定为程式的视窗代号，并在 mciSendCommand 呼叫中包含 MCI_NOTIFY 标记。这导致录音结束後向视窗讯息处理程式发送一条通知讯息。我将简要讨论一下这条通知讯息。

录音结束後，按下前一个「End」按钮来停止录音，这时产生一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_END。回应时，视窗讯息处理程式将呼叫 mciSendCommand 三次：MCI_STOP 命令讯息用於停止录音；MCI_SAVE 命令讯息用於把暂存档案中的声音资料传递到 MCI_SAVE_PARMS 结构中指定的档案（「record2.wav」）；MCI_CLOSE 命令讯息用於删除所有的暂存档案、释放已经建立的记忆体块并关闭设备。

播放时，MCI_OPEN_PARMS 结构的 lpstrElementName 栏位设定为档案名「record2.wav」。 mciSendCommand 第三个参数中所包含的 MCI_OPEN_ELEMENT 标记表示 lpstrElementName 栏位是一个有效的档案名。通过档案的副档名称.WAV，MCI 知道使用者要打开一个波形声音设备。如果存在多个波形硬体，则打开第一个（设定 MCI_OPEN_PARMS 结构的 lpstrDeviceType 栏位，也可以打开其他波形设备）。

播放将包括带有 MCI_PLAY 命令讯息和 MCI_PLAY_PARMS 结构的 mciSendCommand 呼叫。虽然波形档案的任意部分都可以播放，但 RECORD2 只播放整个档案。

RECORD2 还包括一个「Pause」按钮来暂停播放音效档案。这个按钮产生一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_PLAY_PAUSE。回应时，程式将呼叫 mciSendCommand，并以 MCI_PAUSE 命令讯息和 MCI_GENERIC_PARMS 结构作为参数。MCI_GENERIC_PARMS 结构用於这样一些讯息：它们除了需要用於通知的可选视窗代号外，不需要任何资讯。如果播放已经暂停，则通过再次使用 MCI_PLAY 命令讯息呼叫 mciSendCommand 继续播放。

按下第二个「End」按钮也可以停止播放。这时产生 wParam 等於 IDC_PLAY_END 的 WM_COMMAND 讯息。回应时，视窗讯息处理程式将呼叫 mciSendCommand 两次：第一次使用 MCI_STOP 命令讯息；第二次使用 MCI_CLOSE 命令讯息。

现在有一个问题：虽然可以通过按下「End」按钮来手工终止播放，但您可能需要播放整个档案。程式如何知道档案播放完的时间呢？这是 MCI 通知讯息的任务。

当带有 MCI_RECORD 和 MCI_PLAY 讯息来呼叫 mciSendCommand 时，RECORD2 将包括 MCI_NOTIFY 标记，并将资料结构的 dwCallback 栏位设定为程式视窗代号。这样就产生一个通知讯息，称为 MM_MCINOTIFY，并在某些环境下传递给视窗讯息处理程式。讯息参数 wParam 是一个状态代码，而 lParam 是设备 ID。

带有 MCI_STOP 或者 MCI_PAUSE 命令讯息来呼叫 mciSendCommand 时，您将接收到一个 MM_MCINOTIFY 讯息，其中 wParam 等於 MCI_NOTIFY_ABORTED。当您按下「Pause」按钮或者两个「End」按钮中的一个时，就会出现这种情况。由於对这些按钮已进行过适当的处理，所以 RECORD2 可以忽略这种情况。播放时，您会在音效档案结束後接收到 MM_MCINOTIFY 讯息，其中 wParam 等於 MCI_NOTIFY_SUCCESSFUL。这种情况下，视窗讯息处理程式给自己发送一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_PLAY_END，来模拟使用者按下「End」按钮。然後视窗讯息处理程式作出正常回应：停止播放，关闭设备。

录音时，如果用於储存暂存档案的硬碟空间不够，您就会接收一个 MM_MCINOTIFY 讯息，其中 wParam 等於 MCI_NOTIFY_SUCCESSFUL（虽然现在还不能说它很完美，但其功能已经很齐全了）。回应时，视窗讯息处理程式给自己发送一个 WM_COMMAND 讯息，其中 wParam 等於 IDC_RECORD_END，然後与正常情况下一样：停止录音、储存档案并关闭设备。

MCI 命令字串的方法

Windows 的多媒体介面曾经包含函式 mciExecute，其语法如下：

```
bSuccess = mciExecute (szCommand) ;
```

其中唯一的参数是 MCI 命令字串。函式传回布林值——如果呼叫成功，则传回非 0 值，否则传回 0。在功能上，mciExecute 函式相同於呼叫後三个参数为 NULL 或 0 的 mciSendString（TESTMCI 中使用的依据字串的 MCI 函式），然後在发生错误时呼叫 mciGetErrorString 和 MessageBox。

虽然 mciExecute 不再是 API 的一部分，但我还是在 RECORD3 版的数位录音机中使用了这个函式。和 RECORD2 一样，RECORD3 程式也使用 RECORD1 中的资源描述档 RECORD.RC 和 RESOURCE.H，如程式 22-5 所示。

程式 22-5　RECORD3

```
RECORD3.C
/*------------------------------------------------------------------------
-
      RECORD3.C -- Waveform Audio Recorder
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------
*/
```

```
#include <windows.h>
#include "..\\record1\\resource.h"

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("Record3") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      if (-1 == DialogBox (hInstance, TEXT ("Record"), NULL, DlgProc))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                    szAppName, MB_ICONERROR) ;
      }
      return 0 ;
}

BOOL mciExecute (LPCTSTR szCommand)
{
      MCIERROR error ;
      TCHAR   szErrorStr [1024] ;

      if (error = mciSendString (szCommand, NULL, 0, NULL))
      {
                  mciGetErrorString (error, szErrorStr, sizeof (szErrorStr) /
sizeof (TCHAR)) ;
                  MessageBeep (MB_ICONEXCLAMATION) ;
                  MessageBox (     NULL, szErrorStr, TEXT ("MCI Error"),
                                                          MB_OK              |
MB_ICONEXCLAMATION) ;
      }
      return error == 0 ;
}

BOOL CALLBACK DlgProc (     HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static BOOL bRecording, bPlaying, bPaused ;
      switch (message)
      {
      case  WM_COMMAND:
                        switch (wParam)
                        {
                        case  IDC_RECORD_BEG:
                                                // Delete existing waveform file

                                      DeleteFile (TEXT ("record3.wav")) ;
```

```
                                    // Open waveform audio and record

               if (!mciExecute (TEXT ("open new type waveaudio alias mysound")))
                                    return TRUE ;

                                mciExecute (TEXT ("record mysound")) ;

                                    // Enable and disable buttons

                    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), FALSE);
                    EnableWindow (GetDlgItem (hwnd, IDC_RECORD_END), TRUE) ;
                    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG),   FALSE);
                    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
                    EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),   FALSE);
                    SetFocus (GetDlgItem (hwnd, IDC_RECORD_END)) ;

                                bRecording = TRUE ;
                                return TRUE ;

                    case  IDC_RECORD_END:
                                    // Stop, save, and close recording

                         mciExecute (TEXT ("stop mysound")) ;
                         mciExecute (TEXT ("save mysound record3.wav")) ;
                         mciExecute (TEXT ("close mysound")) ;

                                    // Enable and disable buttons

               EnableWindow(GetDlgItem(hwnd,IDC_RECORD_BEG), TRUE) ;
               EnableWindow(GetDlgItem (hwnd, IDC_RECORD_END),     FALSE);
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_BEG),      TRUE) ;
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_PAUSE),    FALSE);
               EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),      FALSE);
               SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

                                bRecording = FALSE ;
                                return TRUE ;

                    case  IDC_PLAY_BEG:
                                    // Open waveform audio and play

               if (!mciExecute (TEXT ("open record3.wav alias mysound")))
                                    return TRUE ;

                                mciExecute (TEXT ("play mysound")) ;

                                    // Enable and disable buttons
```

```
EnableWindow (GetDlgItem     (hwnd, IDC_RECORD_BEG),      FALSE);
 EnableWindow (GetDlgItem    (hwnd, IDC_RECORD_END),      FALSE);
 EnableWindow (GetDlgItem    (hwnd, IDC_PLAY_BEG),        FALSE);
 EnableWindow (GetDlgItem    (hwnd, IDC_PLAY_PAUSE),      TRUE) ;
 EnableWindow (GetDlgItem    (hwnd, IDC_PLAY_END),        TRUE) ;
 SetFocus (GetDlgItem (hwnd, IDC_PLAY_END)) ;

     bPlaying = TRUE ;
  return TRUE ;

            case  IDC_PLAY_PAUSE:
                        if (!bPaused)
                                    // Pause the play
                        {
                        mciExecute (TEXT ("pause mysound")) ;
            SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Resume")) ;
                        bPaused = TRUE ;
                  }
                  else
                                    // Begin playing again
                  {
                        mciExecute (TEXT ("play mysound")) ;
            SetDlgItemText (hwnd, IDC_PLAY_PAUSE, TEXT ("Pause")) ;
                        bPaused = FALSE ;
                  }

                  return TRUE ;

            case  IDC_PLAY_END:
                              // Stop and close

                  mciExecute (TEXT ("stop mysound")) ;
                  mciExecute (TEXT ("close mysound")) ;

                              // Enable and disable buttons
            EnableWindow (GetDlgItem (hwnd, IDC_RECORD_BEG), TRUE) ;
            EnableWindow(GetDlgItem (hwnd, IDC_RECORD_END), FALSE);
            EnableWindow(GetDlgItem (hwnd, IDC_PLAY_BEG),   TRUE) ;
            EnableWindow(GetDlgItem (hwnd, IDC_PLAY_PAUSE), FALSE);
            EnableWindow (GetDlgItem (hwnd, IDC_PLAY_END),   FALSE);
            SetFocus (GetDlgItem (hwnd, IDC_PLAY_BEG)) ;

                  bPlaying = FALSE ;
                  bPaused  = FALSE ;
                  return TRUE ;
            }
            break ;
```

```
      case  WM_SYSCOMMAND:
                        switch (wParam)
                        {
                        case  SC_CLOSE:
                                        if (bRecording)
                        SendMessage (hwnd, WM_COMMAND, IDC_RECORD_END, 0L);

                                        if (bPlaying)
                        SendMessage (hwnd, WM_COMMAND, IDC_PLAY_END, 0L) ;

                                        EndDialog (hwnd, 0) ;
                                        return TRUE ;
                        }
                        break ;
      }
      return FALSE ;
}
```

在研究讯息导向和文字导向的 MCI 介面时，您会发现它们非常相近。很容易就可以猜测出 MCI 将命令字串转换为相应的命令讯息和资料结构。RECORD3 可以使用像 RECORD2 一样使用 MM_MCINOTIFY 讯息，但是它没有选择 mciExecute 函式的好处，它的缺点是程式不知道什么时候播放完波形档案。因此，这些按钮不能自动改变状态。您必须人工按下「End」按钮，以便让程式知道它已经准备再次录音或播放。

注意 MCI 的 open 命令中 alias 关键字的用法。它允许所有後来的 MCI 命令使用别名来引用设备。

波形声音档案格式

如果在十六进位转储程式下研究未压缩的.WAV 档案（即 PCM），您会发现它们具有表 22-1 所示的格式。

表 22-1 .WAV 档案格式

偏移量	位元组	资料
0000	4	「RIFF」
0004	4	波形块的大小（档案大小减 8）
0008	4	「WAVE」
000C	4	「fmt 」
0010	4	格式块的大小（16 位元组）
0014	2	wf.wFormatTag = WAVE_FORMAT_PCM = 1
0016	2	wf.nChannels

0018	4	wf.nSamplesPerSec
001C	4	wf.nAvgBytesPerSec
0020	2	wf.nBlockAlign
0022	2	wf.wBitsPerSample
0024	4	「data」
0028	4	波形资料的大小
002C		波形资料

这是一种扩充自 RIFF（Resource Interchange File Format：资源交换档案格式）的格式。RIFF 是用於多媒体资料档案的万用格式，它是一种标记档案格式。在这种格式下，档案由资料「块」组成，而这些资料块则由前面 4 个字元的 ASCII 名称和 4 位元组（32 位元）的资料块大小来确认。资料块大小值不包括名称和大小所需要的 8 位元组。

波形声音档案以文字字串「RIFF」开始，用来标识这是一个 RIFF 档案。字串後面是一个 32 位元的资料块大小，表示档案其余部分的大小，或者是小於 8 位元组的档案大小。

资料块以文字字串「WAVE」开始，用来标识这是一个波形声音块，後面是文字字串「fmt」——注意用空白使之成为 4 字元的字串——用来标识包含波形声音资料格式的子资料块。「fmt」字串的後面是格式资讯大小，这里是 16 位元组。格式资讯是 WAVEFORMATEX 结构的前 16 个位元组，或者，像最初定义时一样，是包含 WAVEFORMAT 结构的 PCMWAVEFORMAT 结构。

nChannels 栏位的值是 1 或 2,分别对应於单声道和立体声。nSamplesPerSec 栏位是每秒的样本数；标准值是每秒 11,025、22,050 和 44 100 个样本。nAvgBytesPerSec 栏位是取样速率，单位是每秒样本数乘以通道数，再乘以以位元为单位的每个样本的大小，然後除以 8 并往上取整数。标准样本大小是 8 位元和 16 位元。nBlockAlign 栏位是通道数乘以以位元为单位的样本大小，然後除以 8 并往上取整数。最後，该格式以 wBitsPerSample 栏位结束，该栏位是通道数乘以以位元为单位的样本大小。

格式资讯的後面是文字字串「data」，然後是 32 位元的资料大小，最後是波形资料本身。这些资料是按相同格式进行简单连结的样本，这与低阶波形声音设备上所使用的格式相同。如果样本大小是 8 位元，或者更少，那么每个样本有 1 位元组用於单声道，或者有 2 位元组用於立体声。如果样本大小在 9 到 16 位元之间，则每个样本就有 2 位元组用於单声道，或者 4 位元组用於立体声。对於立体声波形资料，每个样本都由左值及其後面的右值组成。

对於 8 位元或不到 8 位元的样本大小，样本位元组被解释为无正负号值。

例如，对於 8 位元的样本大小，静音等於 0x80 位元组的字串。对於 9 位元或更多的样本大小，样本被解释为有正负号值，这时静音的字串等於值 0。

用於读取标记档案的一个重要规则是忽略不准备处理的资料块。尽管波形声音档案需要「fmt」和「data」子资料块（按照此顺序），但它还包含其他子资料块。尤其是，波形声音档案可能包含一个标记为「INFO」的子资料块，和提供波形声音档案资讯的子资料块的子资料块。

叠加合成实验

许多年来——至少从毕达哥拉斯的年代起——人们就已经试图分析音调。起初好像非常简单，但随後就变得复杂了。抱歉，我将重复一些已经说过的有关声音的问题。

音调，除了一些撞击声以外，都有特殊的音调或频率。这个频率可以在人类能够感受到的频谱范围内，也就是从 20Hz 到 20,000Hz 以内。例如，钢琴的频率范围在 27.5Hz 到 4186Hz 之间。音调的另一个特徵是音量或响度。这与产生音调的波形的所有振幅相对应。响度的变化用分贝度量。迄今为止，一切都很好。

然後有一件难办的事称做「音质」。非常简单，音质就是声音的性质，利用它，我们可以区分按相同音调相同音量演奏的钢琴、小提琴和喇叭。

法国数学家 Fourier 发现一些周期性的波形——不论多么复杂——它们都可以表示为许多频率是基础频率整数倍的正弦波形。这个基础频率，也称作第一个谐波，是波形周期的频率。第一个泛音，也称作二级谐波，是基本频率的两倍；第二个泛音，或者三级谐波的频率是基本频率的三倍，依次类推。谐波振幅的相互关系形成了波形的形状。

例如，方波可以表示为许多的正弦波，其中偶数谐波（即 2、4、6 等等）的振幅都是 0，而奇数谐波（即 1、3、5 等等）的振幅都按 1、1/3、1/5 比例依次类推。在锯齿波中，所有的泛音都出现，而振幅都按 1、1/2、1/3、1/4 比例依此类推。

对於德国科学家 Hermann Helmholtz（1821-1894），这是了解音质的关键。在他的名著《On the Sensations of Tone》（1885 年，1954 年由 Dover Press 再版）中，Helmholtz 假定耳朵和大脑将复杂的声音分解为正弦波，而这些正弦波相关的强度就是我们所感受的音质。不幸的是，事情还没有这么简单。

随著 1968 年 Wendy Carlos 的唱片《Switched on Bach》的发布，电子音乐合成器引起了公众的广泛注意。那时使用的合成器（例如 Moog）是类比合成器。这些合成器使用类比电路来产生各种声音波形，例如方波、三角波形和锯

齿波形。要使这些波形听起来更像真实的乐器，它们取决於单个音符的变化程序。波形的所有振幅以「包络（envelope）」形成。当音符开始时，振幅由 0 开始增加，通常增加非常快。这就是所谓的起奏。然後当音符持续时，振幅保持为常数，这时称为持续。音符结束时，振幅降为 0，这时称为释放。

波形通过滤波器，滤波器将削弱一些谐波，并将简单波形转换得更复杂、更有乐感。这些滤波器的切断频率由包络控制，以便声音的谐波内容在音符的程序中改变。

因为这些合成器以丰富的波形格式调和开始，而且一些谐波通过滤波器进行了削弱，这种形式的合成称为「负合成」。

即使在负合成期间，许多人也还会在电子音乐中发现叠加合成是下一个大问题。

在叠加合成中，您可以从许多整数倍正弦波生成器开始，选择整数倍以便於每个正弦波都对应一个谐波。每个谐波的振幅都由一个包络单独控制。使用类比电路的叠加合成不实用，因为对单个音符就需要 8 和 24 之间数目的正弦波生成器，而与这些正弦波生成器相关的频率必须精确的互相对齐。类比波形生成器稳定性很差，而且容易发生频率漂移。

不过，由数位合成器（可以数位化地使用对照表产生波形）和电脑产生的波形，频率漂移并不是个问题，因而叠加合成也就切实可行了。因此总的来说：在录制真实的乐曲时，可以用 Fourier 分解法将其分解成多个谐波。然後就可以确定每个谐波的相对强度，再用多个正弦波数位化地产生声音。

如果开始实验时用 Fourier 分析法分析实际的音调，并从多个正弦波来产生这些音调，那么人们将发现音质并不像 Helmholtz 所认为的那样简单。

最大的问题是真实音调的谐波之间并没有精确的整数关系。事实上，「谐波」一词对於实际的音调来说并不十分适当。各种正弦波组成都不和谐，或者更准确地说是「泛音」。

人们发现，实际音调泛音之间的不和谐在创造「真实的」声音时很重要。静态和谐会产生「电流」声。每个泛音都在单个音符上改变振幅和频率。泛音中，相对频率和振幅的关系对於不同的泛音以及来自相同乐器的不同强度是不同的。实际音调中最复杂的部分发生在音符的起奏部分，这时比较不和谐。人们发现音符的这个复杂的起奏位置对於人类感受音质很重要。

简而言之，实际乐器的声音比任何想像的都更复杂。分析音调的观点，以及後面用於控制泛音的振幅和频率的相对简单的包络观点显然都不实用。

实际乐曲的一些分析法发表於早期（1977 到 1978 年间）的《Computer Music Journal》（当时由 People's Computer Company 发行，现在由 MIT Press 发行）

由 James A. Moorer、John Grey 和 John Strawn Some 编写了第三部分丛书《Lexicon of Analyzed Tones》，该书显示了在小提琴、双簧管、单簧管和喇叭上演奏一个音符（小於半秒种）的泛音的振幅和频率图形。所用的音符是中音 C 上的降 E。小提琴用 20 个泛音，双簧管和单簧管用 21 个，而喇叭用 12 个。实际上，《Computer Music Journal》的 Volume II、Number 2（1978 年 9 月）包含了用线段来近似双簧管、单簧管和喇叭的不同频率和振幅的包络。

因此，利用 Windows 上支援的声音波形功能，下面的程序很简单：将这些数字键入程式、为每个泛音都产生多个正弦波样本、添加这些样本并将其发送给波形声音音效卡，因此把 20 年前原始录制的声音重新制造出来也很容易。ADDSYNTH（「叠加合成」）如程式 22-6 所示。

程式 22-6　ADDSYNTH

ADDSYNTH.C

```
/*------------------------------------------------------------------------
      ADDSYNTH.C -- Additive Synthesis Sound Generation
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>
#include <math.h>
#include "addsynth.h"
#include "resource.h"

#define          ID_TIMER         1
#define          SAMPLE_RATE      22050
#define          MAX_PARTIALS     21
#define          PI                     3.14159

BOOL CALLBACK DlgProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName [] = TEXT ("AddSynth") ;
// Sine wave generator
// -------------------

double SineGenerator (double dFreq, double * pdAngle)
{
      double dAmp ;
      dAmp = sin (* pdAngle) ;
      * pdAngle += 2 * PI * dFreq / SAMPLE_RATE ;

      if (* pdAngle >= 2 * PI)
                        * pdAngle -= 2 * PI ;

      return dAmp ;
}
```

```
// Fill a buffer with composite waveform
// -------------------------------------

VOID FillBuffer (INS ins, PBYTE pBuffer, int iNumSamples)
{
      static double          dAngle [MAX_PARTIALS] ;
      double                 dAmp, dFrq, dComp, dFrac ;
      int                    i, iPrt, iMsecTime, iCompMaxAmp, iMaxAmp, iSmp ;
                             // Calculate the composite maximum amplitude

      iCompMaxAmp = 0 ;
      for (iPrt = 0 ; iPrt < ins.iNumPartials ; iPrt++)
      {
                        iMaxAmp = 0 ;
                        for (i = 0 ; i < ins.pprt[iPrt].iNumAmp ; i++)
                  iMaxAmp=max(iMaxAmp, ins.pprt[iPrt].pEnvAmp[i].iValue) ;
                        iCompMaxAmp += iMaxAmp ;
      }

                             // Loop through each sample
      for (iSmp = 0 ; iSmp < iNumSamples ; iSmp++)
      {
                        dComp = 0 ;
                        iMsecTime = (int) (1000 * iSmp / SAMPLE_RATE) ;

                                  // Loop through each partial
                        for (iPrt = 0 ; iPrt < ins.iNumPartials ; iPrt++)
                        {
                                  dAmp = 0 ;
                                  dFrq = 0 ;

                                  for (i = 0 ; i < ins.pprt[iPrt].iNumAmp - 1 ; i++)
                                  {
           if (iMsecTime >= ins.pprt[iPrt].pEnvAmp[i  ].iTime &&
           iMsecTime <= ins.pprt[iPrt].pEnvAmp[i+1].iTime)
                        {
           dFrac = (double) (iMsecTime -
           ins.pprt[iPrt].pEnvAmp[i  ].iTime) /
           (ins.pprt[iPrt].pEnvAmp[i+1].iTime -
           ins.pprt[iPrt].pEnvAmp[i  ].iTime) ;

           dAmp = dFrac  * ins.pprt[iPrt].pEnvAmp[i+1].iValue +
           (1-dFrac) * ins.pprt[iPrt].pEnvAmp[i  ].iValue ;
           break ;
           }
           }
```

```
         for (i = 0 ; i < ins.pprt[iPrt].iNumFrq - 1 ; i++)
         {
        if (iMsecTime >= ins.pprt[iPrt].pEnvFrq[i  ].iTime &&
           iMsecTime <= ins.pprt[iPrt].pEnvFrq[i+1].iTime)
         {
           dFrac = (double) (iMsecTime -ins.pprt[iPrt].pEnvFrq[i  ].iTime) /

      (ins.pprt[iPrt].pEnvFrq[i+1].iTime -

      ins.pprt[iPrt].pEnvFrq[i  ].iTime) ;
            dFrq = dFrac  * ins.pprt[iPrt].pEnvFrq[i+1].iValue + (1-dFrac) *
ins.pprt[iPrt].pEnvFrq[i  ].iValue ;
          break ;
            }
            }
           dComp += dAmp * SineGenerator (dFrq, dAngle + iPrt) ;
                 }
           pBuffer[iSmp] = (BYTE) (127 + 127 * dComp / iCompMaxAmp) ;
      }
}

// Make a waveform file
// --------------------------------------------------------------------------

BOOL MakeWaveFile (INS ins, TCHAR * szFileName)
{
      DWORD                              dwWritten ;
      HANDLE                             hFile ;
      int                              iChunkSize, iPcmSize, iNumSamples ;
      PBYTE                              pBuffer ;
      WAVEFORMATEX                      waveform ;

      hFile = CreateFile (szFileName, GENERIC_WRITE, 0, NULL,
             CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL) ;
      if (hFile == NULL)
                        return FALSE ;
      iNumSamples = ((long) ins.iMsecTime * SAMPLE_RATE / 1000 + 1) / 2 * 2 ;
      iPcmSize    = sizeof (PCMWAVEFORMAT) ;
      iChunkSize  = 12 + iPcmSize + 8 + iNumSamples ;

      if (NULL == (pBuffer = malloc (iNumSamples)))
      {
                 CloseHandle (hFile) ;
                 return FALSE ;
      }

      FillBuffer (ins, pBuffer, iNumSamples) ;
      waveform.wFormatTag                       = WAVE_FORMAT_PCM ;
```

```
      waveform.nChannels                    = 1 ;
      waveform.nSamplesPerSec           = SAMPLE_RATE ;
      waveform.nAvgBytesPerSec          = SAMPLE_RATE ;
      waveform.nBlockAlign                  = 1 ;
      waveform.wBitsPerSample  = 8 ;
      waveform.cbSize         = 0 ;

      WriteFile   (hFile, "RIFF", 4, &dwWritten, NULL) ;
      WriteFile   (hFile, &iChunkSize, 4, &dwWritten, NULL) ;
      WriteFile   (hFile, "WAVEfmt ", 8, &dwWritten, NULL) ;
      WriteFile   (hFile, &iPcmSize, 4, &dwWritten, NULL) ;
      WriteFile    (hFile, &waveform, sizeof (WAVEFORMATEX) - 2, &dwWritten,
NULL) ;
      WriteFile   (hFile, "data", 4, &dwWritten, NULL) ;
      WriteFile   (hFile, &iNumSamples, 4, &dwWritten, NULL) ;
      WriteFile (hFile, pBuffer,              iNumSamples,        &dwWritten,
NULL) ;

      CloseHandle (hFile) ;
      free (pBuffer) ;

      if ((int) dwWritten != iNumSamples)
      {
                  DeleteFile (szFileName) ;
                  return FALSE ;
      }
      return TRUE ;
}

void TestAndCreateFile (     HWND hwnd, INS ins, TCHAR * szFileName,
                                                      int idButton)
{
      TCHAR szMessage [64] ;
      if (-1 != GetFileAttributes (szFileName))
                        EnableWindow (GetDlgItem (hwnd, idButton), TRUE) ;
      else
      {
                  if (MakeWaveFile (ins, szFileName))
                              EnableWindow (GetDlgItem (hwnd, idButton), TRUE) ;
                  else
                  {
                  wsprintf (szMessage, TEXT ("Could not create %x."), szFileName) ;
                              MessageBeep (MB_ICONEXCLAMATION) ;
                              MessageBox (hwnd,      szMessage, szAppName,
                         MB_OK | MB_ICONEXCLAMATION) ;
            }
      }
}
```

```
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
      if (-1 == DialogBox (hInstance, szAppName, NULL, DlgProc))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                    szAppName, MB_ICONERROR) ;
      }
      return 0 ;
}

BOOL CALLBACK DlgProc (      HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static TCHAR * szTrum = TEXT ("Trumpet.wav") ;
      static TCHAR * szOboe = TEXT ("Oboe.wav") ;
      static TCHAR * szClar = TEXT ("Clarinet.wav") ;

      switch (message)
      {
      case  WM_INITDIALOG:
                        SetTimer (hwnd, ID_TIMER, 1, NULL) ;
                        return TRUE ;

      case  WM_TIMER:
                        KillTimer (hwnd, ID_TIMER) ;
                        SetCursor (LoadCursor (NULL, IDC_WAIT)) ;
                        ShowCursor (TRUE) ;

                        TestAndCreateFile (hwnd, insTrum, szTrum, IDC_TRUMPET) ;
                        TestAndCreateFile (hwnd, insOboe, szOboe, IDC_OBOE) ;
                        TestAndCreateFile (hwnd, insClar, szClar, IDC_CLARINET) ;

                        SetDlgItemText (hwnd, IDC_TEXT, TEXT (" ")) ;
                        SetFocus (GetDlgItem (hwnd, IDC_TRUMPET)) ;

                        ShowCursor (FALSE) ;
                        SetCursor (LoadCursor (NULL, IDC_ARROW)) ;
                        return TRUE ;

      case  WM_COMMAND:
                        switch (LOWORD (wParam))
                        {
                        case  IDC_TRUMPET:
                              PlaySound (szTrum, NULL, SND_FILENAME | SND_SYNC) ;
                                          return TRUE ;
```

```
                        case  IDC_OBOE:
                              PlaySound (szOboe, NULL, SND_FILENAME | SND_SYNC) ;
                                    return TRUE ;

                        case  IDC_CLARINET:
                              PlaySound (szClar, NULL, SND_FILENAME |SND_SYNC) ;
                                    return TRUE ;
                        }
                        break ;

      case  WM_SYSCOMMAND:
                        switch (LOWORD (wParam))
                        {
                        case SC_CLOSE:
                                    EndDialog (hwnd, 0) ;
                                    return TRUE ;
                        }
                        break ;
      }
      return FALSE ;
}
```

ADDSYNTH.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////////
/
// Dialog
ADDSYNTH          DIALOG DISCARDABLE  100, 100, 176, 49
STYLE                   WS_MINIMIZEBOX | WS_CAPTION | WS_SYSMENU
CAPTION           "Additive Synthesis"
FONT 8,                       "MS Sans Serif"
BEGIN
      PUSHBUTTON                  "Trumpet",IDC_TRUMPET,8,8,48,16
      PUSHBUTTON                  "Oboe",IDC_OBOE,64,8,48,16
      PUSHBUTTON                  "Clarinet",IDC_CLARINET,120,8,48,16
      LTEXT                       "Preparing Data...",IDC_TEXT,8,32,100,8
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by AddSynth.rc

#define           IDC_TRUMPET                               1000
#define           IDC_OBOE                                  1001
#define           IDC_CLARINET                              1002
#define           IDC_TEXT                                  1003
```

这里没有给出附加档案 ADDSYNTH.H，因为它包含几百行令人讨厌的叙述，您将在本书附上的光碟上找到它。在 ADDSYNTH.H 的开始位置，我定义了三个结构，用於储存包络资料。每个振幅和频率分别储存到型态 ENV 的结构阵列中。这些数字对由时间（毫秒）和振幅值（按任意度量单位）或频率（以周期/秒为单位）组成。这些阵列的长度可变，其变化范围从 6 到 14。假定振幅和频率值之间直接相关。

每种乐器都包括一个泛音集（喇叭用 12 个，双簧管和单簧管分别使用 21 个），这些泛音集储存在型态 PRT 的结构阵列中。PRT 结构储存振幅和频率包络的点数，以及指向 ENV 阵列的指标。INS 结构包括音调的总时间（以毫秒为单位）、泛音数以及指向储存泛音的 PRT 阵列的指标。

ADDSYNTH 有三个标记为「Trumpet」、「Oboe」和「Clarinet」的按钮。PC 的速度还没有快到足以即时计算所有的叠加合成，因此第一次执行 ADDSYNTH 时，这些按钮将失效，直到程式计算完样本并建立了 TRUMPET.WAV、OBOE.WAV 和 CLARINET.WAV 音效档案後，按钮才启动，而且可以使用 PlaySound 函式播放这三种声音。下次执行时，程式将检查波形档案是否存在，而不需重新建立。

ADDSYNTH 中的 FillBuffer 函式完成了大多数工作。FillBuffer 从计算合成最大振幅的总数开始。为此，它在乐器的泛音中回圈，以找出每个泛音的最大振幅，然後将所有的最大振幅加起来。此值後来用於将样本缩放到 8 位元的样本大小。

然後 FillBuffer 计算每个样本的值。每个样本都对应於一段以毫秒为单位的时间，该时间取决於取样频率（实际上，在 22.05 kHz 的取样频率下，每 22 个样本对应於相同的毫秒时间值）。然後，FillBuffer 在泛音中回圈。对於频率和振幅，它找出与毫秒时间值对应的包络线段，并执行线性插补。

频率值与相位角值一起传递给 SineGenerator 函式。本章前面讨论过，产生数位化的正弦波形需要保持相位角值，并依据频率值增加。从 SineGenerator 函式传回时，正弦值将乘以泛音的振幅并累加。样本的所有泛音都加在起来之後，样本就缩放到位元组大小。

起床号波形声音

WAKEUP，如程式 22-7 所示，是原始码档案看起来不是很完整的程式之一。程式视窗看起来像对话方块，但是没有资源描述档（我们已经知道如何编写），并且程式使用一个波形档案，但在光碟上却没有这样的档案。不过，程式非常有趣：它播放的声音很大，并且非常令人讨厌。WAKEUP 是我的闹钟，能够唤醒我继续工作。

程式 22-7 WAKEUP

```
WAKEUP.C
/*-------------------------------------------------------------------------
-
      WAKEUP.C --       Alarm Clock Program
                                                (c) Charles Petzold, 1998
-------------------------------------------------------------------------*
/

#include <windows.h>
#include <commctrl.h>

            // ID values for 3 child windows
#define           ID_TIMEPICK             0
#define           ID_CHECKBOX       1
#define           ID_PUSHBTN        2

            // Timer ID

#define           ID_TIMER                1

            // Number of 100-nanosecond increments (ie FILETIME ticks) in an hour
#define FTTICKSPERHOUR (60 * 60 * (LONGLONG) 10000000)
            // Defines and structure for waveform "file"
#define           SAMPRATE          11025
#define           NUMSAMPS          (3 * SAMPRATE)
#define           HALFSAMPS   (NUMSAMPS / 2)

typedef struct
{
      char        chRiff[4] ;
      DWORD             dwRiffSize ;
      char        chWave[4] ;
      char        chFmt [4] ;
      DWORD             dwFmtSize ;
      PCMWAVEFORMAT pwf ;
      char        chData[4] ;
      DWORD             dwDataSize ;
      BYTE        byData[0] ;
}
WAVEFORM ;
      // The window proc and the subclass proc
LRESULT     CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
LRESULT     CALLBACK SubProc (HWND, UINT, WPARAM, LPARAM) ;

      // Original window procedure addresses for the subclassed windows

WNDPROC SubbedProc [3] ;
```

```
     // The current child window with the input focus

HWND hwndFocus ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInst,
                                                    PSTR    szCmdLine,    int
iCmdShow)
{
     static TCHAR     szAppName [] = TEXT ("WakeUp") ;
     HWND                              hwnd ;
     MSG                         msg ;
     WNDCLASS                    wndclass ;

     wndclass.style                    = 0 ;
     wndclass.lpfnWndProc              = WndProc ;
     wndclass.cbClsExtra               = 0 ;
     wndclass.cbWndExtra               = 0 ;
     wndclass.hInstance                = hInstance ;
     wndclass.hIcon                    = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor                  = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground       = (HBRUSH) (1 + COLOR_BTNFACE) ;
     wndclass.lpszMenuName        = NULL ;
     wndclass.lpszClassName       = szAppName ;

     if (!RegisterClass (&wndclass))
     {
          MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                    szAppName, MB_ICONERROR) ;
               return 0 ;
     }

     hwnd = CreateWindow (  szAppName, szAppName,
                            WS_OVERLAPPED | WS_CAPTION |
                       WS_SYSMENU | WS_MINIMIZEBOX,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            CW_USEDEFAULT, CW_USEDEFAULT,
                            NULL, NULL, hInstance, NULL) ;

     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg, NULL, 0, 0))
    {
                    TranslateMessage (&msg) ;
                    DispatchMessage (&msg) ;
     }
     return msg.wParam ;
```

```
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static HWND                  hwndDTP, hwndCheck, hwndPush ;
      static WAVEFORM              waveform = { "RIFF", NUMSAMPS + 0x24,
"WAVE", "fmt ",
                                        sizeof (PCMWAVEFORMAT), 1, 1, SAMPRATE,
                                 SAMPRATE, 1, 8, "data", NUMSAMPS } ;
      static WAVEFORM                    * pwaveform ;
      FILETIME                     ft ;
      HINSTANCE                    hInstance ;
      INITCOMMONCONTROLSEX         icex ;
      int                          i, cxChar, cyChar ;
      LARGE_INTEGER                li ;
      SYSTEMTIME                   st ;

      switch (message)
      {
      case  WM_CREATE:
                                        // Some initialization stuff

                        hInstance    =    (HINSTANCE)    GetWindowLong    (hwnd,
GWL_HINSTANCE) ;

                        icex.dwSize = sizeof (icex) ;
                        icex.dwICC  = ICC_DATE_CLASSES ;
                        InitCommonControlsEx (&icex) ;

                        // Create the waveform file with alternating square waves

                        pwaveform = malloc (sizeof (WAVEFORM) + NUMSAMPS) ;
                        * pwaveform = waveform ;

                        for (i = 0 ; i < HALFSAMPS ; i++)
                                        if (i % 600 < 300)
                              if (i % 16 < 8)
                  pwaveform->byData[i] = 25 ;
                  else
                  pwaveform->byData[i] = 230 ;
                        else
                  if (i % 8 < 4)
                  pwaveform->byData[i] = 25 ;
                  else
                  pwaveform->byData[i] = 230 ;
                        // Get character size and set a fixed window size.
                        cxChar = LOWORD (GetDialogBaseUnits ()) ;
```

```
                    cyChar = HIWORD (GetDialogBaseUnits ()) ;

                SetWindowPos (   hwnd, NULL, 0, 0, 42 * cxChar, 10 * cyChar /
3 + 2 *
GetSystemMetrics (SM_CYBORDER) +GetSystemMetrics (SM_CYCAPTION)
,SWP_NOMOVE | SWP_NOZORDER | SWP_NOACTIVATE) ;

                                    // Create the three child windows

          hwndDTP = CreateWindow (DATETIMEPICK_CLASS, TEXT (""),
                 WS_BORDER | WS_CHILD | WS_VISIBLE | DTS_TIMEFORMAT,
                 2 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
                 hwnd, (HMENU) ID_TIMEPICK, hInstance, NULL) ;
          hwndCheck = CreateWindow (TEXT ("Button"), TEXT ("Set Alarm"),
                  WS_CHILD | WS_VISIBLE | BS_AUTOCHECKBOX,
                  16 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
                  hwnd, (HMENU) ID_CHECKBOX, hInstance, NULL) ;

          hwndPush = CreateWindow (TEXT ("Button"), TEXT ("Turn Off"),
                  WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON | WS_DISABLED,
                  28 * cxChar, cyChar, 12 * cxChar, 4 * cyChar / 3,
                  hwnd, (HMENU) ID_PUSHBTN, hInstance, NULL) ;

                    hwndFocus = hwndDTP ;

                                    // Subclass the three child windows

                    SubbedProc [ID_TIMEPICK] = (WNDPROC)
               SetWindowLong (hwndDTP, GWL_WNDPROC, (LONG) SubProc) ;
                    SubbedProc [ID_CHECKBOX] = (WNDPROC)
               SetWindowLong (hwndCheck, GWL_WNDPROC, (LONG) SubProc);
                    SubbedProc [ID_PUSHBTN] = (WNDPROC)
               SetWindowLong (hwndPush, GWL_WNDPROC, (LONG) SubProc) ;

                // Set the date and time picker control to the current time
                           // plus 9 hours, rounded down to next lowest hour

                    GetLocalTime (&st) ;
                    SystemTimeToFileTime (&st, &ft) ;
                    li = * (LARGE_INTEGER *) &ft ;
                    li.QuadPart += 9 * FTTICKSPERHOUR ;
                    ft = * (FILETIME *) &li ;
                    FileTimeToSystemTime (&ft, &st) ;
                    st.wMinute = st.wSecond = st.wMilliseconds = 0 ;
               SendMessage (hwndDTP, DTM_SETSYSTEMTIME, 0, (LPARAM) &st) ;
                    return 0 ;

     case  WM_SETFOCUS:
```

```
                    SetFocus (hwndFocus) ;
                    return 0 ;

     case  WM_COMMAND:
                    switch (LOWORD (wParam))     // control ID
                    {
                    case  ID_CHECKBOX:

                    // When the user checks the "Set Alarm" button, get the
                    // time in the date and time control and subtract from
                                        // it the current PC time.

                         if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
                              {
                SendMessage (hwndDTP, DTM_GETSYSTEMTIME, 0, (LPARAM) &st) ;
                                              SystemTimeToFileTime (&st, &ft) ;
                                        li = * (LARGE_INTEGER *) &ft ;

                                        GetLocalTime (&st) ;
                                        SystemTimeToFileTime (&st, &ft) ;
                         li.QuadPart -= ((LARGE_INTEGER *) &ft)->QuadPart ;

                         // Make sure the time is between 0 and 24 hours!
                         // These little adjustments let us completely ignore
                         // the date part of the SYSTEMTIME structures.

                              while (     li.QuadPart < 0)
                                   li.QuadPart += 24 * FTTICKSPERHOUR ;

                                   li.QuadPart %= 24 * FTTICKSPERHOUR ;

                         // Set a one-shot timer! (See you in the morning.)

                SetTimer (hwnd, ID_TIMER, (int) (li.QuadPart / 10000), 0) ;
                                   }
                    // If button is being unchecked, kill the timer.

                         else
                                   KillTimer (hwnd, ID_TIMER) ;

                         return 0 ;

                // The "Turn Off" button turns off the ringing alarm, and also
                // unchecks the "Set Alarm" button and disables itself.

                case  ID_PUSHBTN:
                PlaySound (NULL, NULL, 0) ;
                SendMessage (hwndCheck, BM_SETCHECK, 0, 0) ;
```

```
                EnableWindow (hwndDTP, TRUE) ;
                EnableWindow (hwndCheck, TRUE) ;
              EnableWindow (hwndPush, FALSE) ;
                SetFocus (hwndDTP) ;
                return 0 ;
                }
           return 0 ;

                // The WM_NOTIFY message comes from the date and time picker.
                // If the user has checked "Set Alarm" and then gone back to
                // change the alarm time, there might be a discrepancy between
                // the displayed time and the one-shot timer. So the program
                // unchecks "Set Alarm" and kills any outstanding timer.

     case  WM_NOTIFY:
                      switch (wParam)                          // control ID
                      {
                      case  ID_TIMEPICK:
                      switch (((NMHDR *) lParam)->code) // notification code
                                      {
                                      case  DTN_DATETIMECHANGE:
                      if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
                      {
                      KillTimer (hwnd, ID_TIMER) ;
                      SendMessage (hwndCheck, BM_SETCHECK, 0, 0) ;
                      }
                      return 0 ;
                                  }
                      }
                      return 0 ;

                // The WM_COMMAND message comes from the two buttons.


     case  WM_TIMER:

                // When the timer message comes, kill the timer (because we only
                // want a one-shot) and start the annoying alarm noise going.

                      KillTimer ( hwnd, ID_TIMER) ;
                      PlaySound ( (PTSTR) pwaveform,  NULL,
                                                                SND_MEMORY   |
SND_LOOP | SND_ASYNC);

                // Let the sleepy user turn off the timer by slapping the
                // space bar. If the window is minimized, it's restored; then
                // it's brought to the forefront; then the pushbutton is enabled
                            // and given the input focus.
```

```
                        EnableWindow (hwndDTP, FALSE) ;
                        EnableWindow (hwndCheck, FALSE) ;
                        EnableWindow (hwndPush, TRUE) ;

                        hwndFocus = hwndPush ;
                        ShowWindow (hwnd, SW_RESTORE) ;
                        SetForegroundWindow (hwnd) ;
                        return 0 ;

                // Clean up if the alarm is ringing or the timer is still set.

      case  WM_DESTROY:
                        free (pwaveform) ;

                        if (IsWindowEnabled (hwndPush))
                                        PlaySound (NULL, NULL, 0) ;

                        if (SendMessage (hwndCheck, BM_GETCHECK, 0, 0))
                                        KillTimer (hwnd, ID_TIMER) ;

                        PostQuitMessage (0) ;
                        return 0 ;
      }
      return DefWindowProc (hwnd, message, wParam, lParam) ;
}

LRESULT CALLBACK SubProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      int idNext, id = GetWindowLong (hwnd, GWL_ID) ;
      switch (message)
      {
      case  WM_CHAR:
                        if (wParam == '\t')
                        {
                                        idNext = id ;

                                        do
                                idNext = (idNext +
                                        (GetKeyState (VK_SHIFT) < 0 ? 2 : 1)) % 3 ;
                                while  (!IsWindowEnabled  (GetDlgItem  (GetParent
(hwnd), idNext)));

                                SetFocus (GetDlgItem (GetParent (hwnd), idNext)) ;
                                        return 0 ;
                        }
                        break ;
```

```
      case  WM_SETFOCUS:
                        hwndFocus = hwnd ;
                        break ;
      }
      return CallWindowProc (      SubbedProc     [id],     hwnd,     message,
wParam,lParam) ;
}
```

WAKEUP 使用的波形只有两个方波，但是变化迅速。实际的波形在 WndProc 的 WM_CREATE 讯息处理期间计算。所有的波形档案都储存在记忆体中。指向这个记忆体块的指标传递给 PlaySound 函式，该函式使用 SND_MEMORY、SND_LOOP 和 SND_ASYNC 参数。

WAKEUP 使用称为「Date-Time Picker」的通用控制项。这个控制项用来让使用者选择指定的日期和时间（WAKEUP 只使用时间挑选功能）。程式可以使用 SYSTEMTIME 结构来获得和设定时间,在获得和设定 PC 自身时钟时也使用该结构。要多方面了解 Date-Time Picker，请试著建立不带有任何 DTS 样式旗标的视窗。

注意 WM_CREATE 讯息结束时的处理方式：程式假定您在睡觉之前执行它，并希望它在 8 小时之後来唤醒您。

现在很明显，可以从 GetLocalTime 函式在 SYSTEMTIME 结构获得目前时间，而且可以「手工」增加时间。但在一般情况下，此计算将涉及检查大於 24 小时的结果时间，这意味著您必须增加天数栏位，然後可能涉及增加月（因此还必须有用於每月天数和闰年检查的逻辑），最後您可能还要增加年。

事实上，推荐的方法（来自/Platform SDK/Windows Base Services/General Library/Time/Time Reference/Time Structures/SYSTEMTIME）是将 SYSTEMTIME 转换为 FILETIME 结构（使用 SystemTimeToFileTime），将 FILETIME 结构强制转换为 LARGE_INTEGER 结构，在大整数上执行计算，再强制转换回 FILETIME 结构，然後转换回 SYSTEMTIME 结构（使用 FileTimeToSystemTime）。

顾名思义，FILETIME 结构用於获得和设定档案最後一次更新的时间。此结构如下：

```
type struct _FILETIME                    // ft
{
      DWORD dwLowDateTime ;
      DWORD dwHighDateTime ;
}
FILETIME ;
```

这两个栏位一起表示了从 1601 年 1 月 1 日起每隔 1000 亿分之一秒所显示的 64 位元值。

Microsoft C/C++编译器支援 64 位元整数作为 ANSI C 的非标准延伸语法。

资料型态是__int64。您可以对__int64 型态执行所有的常规算术运算，并且有一些执行时期程式库函式也支援它们。Windows 的 WINNT.H 表头档案定义如下：

```
typedef __int64 LONGLONG ;
typedef unsigned __int64 DWORDLONG ;
```

在 Windows 中，这有时称为「四字组」，或者更普遍地称为「大整数」。也有一个 union 定义如下：

```
typedef union _LARGE_INTEGER
{
      struct
      {
                  DWORD LowPart ;
                  LONG  HighPart ;
      } ;
      LONGLONG QuadPart ;
}
LARGE_INTEGER ;
```

这是 /Platform SDK/Windows Base Services/General Library/Large Integer Operations 中的全部文件。此 union 允许您使用 32 位元或者 64 位元的大整数。

MIDI 和音乐

由电子音乐合成器制造者协会在 19 世纪 80 年代早期开发了「乐器数位化介面」（MIDI：Musical Instrument Digital Interface）。MIDI 是用於将它们中的电子乐器与电脑连结起来的协定，也是电子音乐领域中相当重要的标准。MIDI 规范由 MIDI Manufacturers Association（MMA）维护，它的网站是 http://www.midi.org 。

使用 MIDI

MIDI 为透过电缆来传递数位命令定义了传输协定。MIDI 电缆使用 5 针 DIN 接头，但是只使用了三个接头。一个是遮罩，一个是回路，而第三个传输资料。MIDI 协定在每秒 31,250 位元的速度下是单向的。资料的每个位元组都由一个开始位元开始，以一个停止位元结束，用於每秒 3,125 位元组的有效传输速率。

重要的是要了解真实的声音——不论是类比格式还是数位格式——不是经由 MIDI 电缆传输的。通过电缆传输的通常都是简单的命令讯息，长度一般是 1、2 或 3 位元组。

简单的 MIDI 设定可以包括两片 MIDI 相容硬体。一个是本身不发声，但是单独产生 MIDI 讯息的 MIDI 键盘。此键盘有一个有标记有「MIDI Out」的 MIDI

埠。用 MIDI 电缆将这个埠与 MIDI 声音合成器的「MIDI In」埠连结起来。合成器看起来很像前面有几个按钮的小盒子。

按下键盘上的一个键时（假定是中音 C），键盘就将 3 个位元组发送给 MIDI Out 埠。在十六进位中，这些位元组是：

90 3C 40

第一个位元组（90）显示 Note On 讯息。第二个位元组是键号，其中 3C 是中音 C。第三个位元组是敲按键的速度，此速度范围是从 1 到 127。我们恰巧使用了一个对速度不敏感的键盘，因此它发送平均速度值。这个 3 位元组的讯息顺著 MIDI 电缆进入合成器的 Midi In 埠。通过播放中音 C 的音调来回应合成器。

释放键时，键盘会将另一个 3 位元组讯息发送给 MIDI Out 埠：

90 3C 00

这与 Note On 命令相同，但带有 0 速位元组。这个位元组值 0 表示 Note Off 命令，意味著应该关闭音符。合成器通过停止声音来回应。

如果合成器有复调音乐的能力（即，同时播放多个音符的能力），那么您就可以在键盘上演奏和弦。键盘产生多条 Note On 讯息，并且合成器将播放所有的音符。当您释放和弦时，键盘就将多条 Note Off 讯息发送给合成器。

一般来说，这种设定中的键盘称为「MIDI 控制器」，它负责产生 MIDI 讯息来控制合成器。MIDI 控制器看起来不像键盘。MIDI 控制器包括下面几种：看起来像单簧管或萨克斯管的 MID 管乐控制器、MIDI 吉他控制器、MIDI 弦乐控制器和 MIDI 鼓控制器。至少所有这些控制器都产生 3 位元组的 Note On 和 Note Off 讯息。

胜过类似的键盘或传统乐器，控制器也可以是「编曲器」，它是在记忆体中储存 Note On 和 Note Off 讯息顺序，然後再播放的硬体。现在单机编曲器已经比几年前少见多了，因为它们已经被电脑所替代。安装 MIDI 卡的电脑也可以生成 Note On 和 Note Off 讯息来控制合成器。MIDI 编辑软体，允许您在萤幕上作曲，还可以储存来自 MIDI 控制器的 MIDI 讯息，并处理这些讯息，然後将 MIDI 讯息发送给合成器。

合成器有时也称为「声音模组（sound module）」或「音源器（tone generator）」。MIDI 不指定如何真正产生这些声音的方法。合成器可以使用任何一种声音生成技术。

实际上，只有非常简单的 MIDI 控制器（例如管乐控制器）才只有 MIDI Out 电缆埠。通常键盘都有内建合成器，并且有三个 MIDI 电缆埠，分别标记为「MIDI In」、「MIDI Out」和「MIDI Thru」。MIDI In 埠接受 MIDI 讯息，从而播放键盘的内部合成器。MIDI Out 埠将 MIDI 讯息从键盘发送到外部合成器。MIDI Thru

埠是一个输出埠，它复制 MIDI In 埠的输入信号——无论从 MIDI In 埠获得什么都发送给 MIDI Thru 埠（MIDI Thru 埠不包括从 MIDI Out 埠发送的任何资讯）。

透过电缆连结 MIDI 硬体只有两种方法：将一个硬体上的 MIDI Out 连结到另一个的 MIDI In，或者将 MIDI Thru 与 MIDI In 连结。MIDI Thru 埠允许连结一系列的 MIDI 合成器。

程式更改

合成器能制作哪种声音？是钢琴声、小提琴声、喇叭声还是飞碟声？通常合成器能够生成的各种声音都储存在 ROM 或者其他地方。它们通常称为「声音」、「乐器」或者「音色」。（「音色」一词来自类比合成器的时代，当时通过将音色和弦插入合成器前面的插孔中来设定不同的声音）。

在 MIDI 中，合成器能够生成的各种声音称为「程式」。改变这个程式需要向合成器发送 MIDI Program Change 讯息

C0 pp

其中，pp 的范围是 0 到 127。通常 MIDI 键盘的顶部是一系列有限的按钮，这些按钮将产生 Program Change 讯息。透过按下这些按钮，您可以从键盘控制合成器的声音。这些按钮号通常由 1 开始，而不是由 0 开始，因此程式代号 1 与 Program Change 位元组的 0 对应。

MIDI 规格没有说明程式代号与乐器的对应关系。例如，著名的 Yamaha DX7 合成器上的前三个程式分别称为「Warm Strings」、「 Mellow Horn」和「Pick Guitar」。而在 Yamaha TX81Z 音调发生器上，它们是 Grand Piano、Upright Piano 和 Deep Grand。在 Roland MT-32 声音模组上，它们是 Acoustic Piano 1、Acoustic Piano 2 和 Acoustic Piano 3。因此，如果不希望在从键盘制作程式改变时感到吃惊，那么最好了解一下乐器声与您将使用的合成器的程式代号的对应关系。

这对於包含 Program Change 讯息的 MIDI 档案来说是一个实际问题——这些档案并不是装置无关的，因为它们的内容在不同的合成器上听起来是不一样的。然而，在最近几年，「General MIDI」（GM）标准已经把这些程式代号标准化。Windows 支援 General MIDI。如果合成器与 General MIDI 规格不一致，那么程式转换可使它模拟 General MIDI 合成器。

MIDI 通道

迄今为止，我已经讨论了两条 MIDI 讯息，第一条是 Note On：

90 kk vv

其中，kk 是键号（0 到 127），v v 是速度（0 到 127）。0 速度表示 Note Off

命令。第二条是 Program Change：

C0 pp

其中，pp 的范围是从 0 到 127。这些是典型的 MIDI 讯息。第一个位元组称作「状态」位元组。根据位元组的状态，它通常後跟 0、1 或 2 位元组的「资料」（我即将说明的「系统专有」讯息除外）。从资料位元组中分辨出状态位元组很容易：高位总是 1 用於状态位元组，0 用於资料位元组。

然而，我还没有讨论过这两个讯息的普通格式。Note On 讯息的普通格式如下：

9n kk vv

而 Program Change 是：

Cn pp

在这两种情况下，n 表示状态位元组的低四位元，其变化范围是 0 到 15。这就是 MIDI「通道」。通道一般从 1 开始编号，因此，如果 n 为 0，则代表通道 1。

使用 16 个不同通道允许一条 MIDI 电缆传输 16 种不同声音的讯息。通常，您将发现 MIDI 讯息的特殊字串以 Program Change 讯息开始，为所用的不同通道设定声音，而字串的後面是多条 Note On 和 Note Off 命令。再後面可能是其他的 Program Change 命令。但任何时候，每个通道都只与一种声音联系。

让我们作一个简单范例：假定我已经讨论过的键盘控制能够同时产生用於两条不同通道——通道 1 和通道 2——的 MIDI 讯息。透过按下键盘上的按钮将两条 Program Change 讯息发送给合成器：

C0 01

C1 05

现在设定通道 1 用於程式 2，并设定通道 2 用於程式 6（回忆通道代号和程式代号都是基於 1 的，但讯息中的编码是基於 0 的）。现在按下键盘上的键时，就发送两条 Note On 讯息，一条用於一个通道：

90 kk vv

91 kk vv

这就允许您和谐地同时播放两种乐器的声音。

另一种方法是「分开」键盘。低键可以在通道 1 上产生 Note On 讯息，高键可以在通道 2 上产生 Note On 讯息。这就允许您在一个键盘上独立播放两种乐器的声音。

当您考虑 PC 上的 MIDI 编曲软体时，使用 16 个通道将更为有利。每个通道都代表不同的乐器。如果有能够独立播放 16 种不同乐器的合成器，那么您就可

以编写用於 16 个波段的管弦乐曲，而且只使用一条 MIDI 电缆将 MIDI 卡与合成器连结起来。

MIDI 讯息

尽管 Note On 和 Program Change 讯息在任何 MIDI 执行中都是最重要的讯息，但并不是所有的 MIDI 都可以执行。表 22-2 是 MIDI 规格中定义的 MIDI 通道讯息表。我在前面提到过，状态位元组的高位元总是设定著，而状态位元组後面的资料位元组的高位元都等於 0。这意味著状态位元组的范围是 0x80 到 0xFF，而资料位元组的范围是 0 到 0x7F。

表 22-2 MIDI 通道讯息（n =通道代号，从 0 到 15）

MIDI 讯息	资料位元组	值
Note Off	8n kk vv	kk = 键号（0-127） vv = 速度（0-127）
Note On	9n kk vv	kk = 键号（0-127） vv = 速度（1-127, 0 = note off）
Polyphonic After Touch	An kk tt	kk = 键号（0-127） tt = 按下之後 （0-127）
Control Change	Bn cc xx	cc = 控制器（0-121） xx = 值（0-127）
Channel Mode Local Control	Bn 7A xx	xx = 0（关），127（开）
All Notes Off	Bn 7B 00	
Omni Mode Off	Bn 7C 00	
Omni Mode On	Bn 7D 00	
Mono Mode On	Bn 7E cc	cc = 频道数
Poly Mode On	Bn 7F 00	
Program Change	Cn pp	pp = 程式（0-127）
Channel After Touch	Dn tt	tt = 按下之後（0-127）
Pitch Wheel Change	En ll hh	ll = 低 7 位元（0-127） hh = 高 7 位元（0-127）

虽然没有严格的要求，键号通常还是与西方音乐的传统音符相对应（例如，

对於打击声音，每个键号码可以是不同的打击乐器）。当键号与钢琴类的键盘对应时，键 60（十进位）是中音 C。MIDI 键号在普通的 88 键钢琴范围的基础上向下扩展了 21 个音符，向上扩展了 19 个音符。速度代号是按下某键的速度，在钢琴上它控制声音的响度与和谐特徵。特殊的声音可以依这种方式或其他方式来回应键的速度。

前面展示的例子使用带有 0 速度位元组的 Note On 讯息来表示 Note Off 命令。对於键盘（或者其他控制器）还有一个单独的 Note Off 命令，该命令实作释放键的速度，不过，非常少见。

还有两个「接触後」讯息。接触後是一些键盘的特徵，按下某个键以後，再用力按下键可以在某些方式上改变声音。一个讯息（状态位元组 0xDn）是将接触後应用於通道中目前演奏的所有音符，这是最常见的。状态位元组 0xAn 表示独立应用每个单独键的接触後。

通常，键盘上都有一些用於进一步控制声音的刻度盘或开关。这些装置称为「控制器」，所有变化都由状态位元组 0xBn 表示。通过从 0 到 121 的号码确认控制器。0xBn 状态位元组也用於 Channel Mode 讯息，这些讯息显示了合成器如何在通道中回应同时发生的音符。

一个非常重要的控制器是上下转换音调的轮，它有一个单独的 MIDI 讯息，其状态位元组是 0xEn。

表 22-2 中所缺少的是状态位元组以从 F0 到 FF 开始的讯息。这些讯息称为系统讯息，因为它们适用於整个 MIDI 系统，而不是部分通道。系统讯息通常用於同步的目的、触发编曲器、重新设定硬体以及获得资讯。

许多 MIDI 控制器连续发送状态位元组 0xFE，该位元组称为 Active Sensing 讯息。这简单地表示了 MIDI 控制器仍依附於系统。

一条重要的系统讯息是以状态位元组 0xF0 开始的「系统专用」讯息。此讯息用於将资料块按厂商与合成器所依靠的格式传递给合成器（例如，用这种方法可以将新的声音定义从电脑传递给合成器）。系统专用讯息只是可以包含多於 2 个资料位元组的唯一讯息。实际上，资料位元组数是变化的，而每个资料位元组的高位都设定为 0。状态位元组 0xF7 表示系统专用讯息的结尾。

系统专用讯息也用於从合成器转储资料（例如，声音定义）。这些资料都是通过 MIDI Out 埠来自合成器。如果要用装置无关的方式对 MIDI 编写程式，则应该尽可能避免使用系统专用讯息。但是它们对於定义新的合成器声音是非常有用的。

MIDI 档案（副档名是.MDI）是带有定时资讯的 MIDI 资讯集，可以用 MCI 播放 MIDI 档案。不过，我将在本章的後面讨论低阶 midiOut 函式。

MIDI 编曲简介

低阶 MIDI 的 API 包括字首为 midiIn 和 midiOut 的函式，它们分别用於读取来自外部控制器的 MIDI 序列和在内部或外部的合成器上播放音乐。尽管其名称为「低阶」，但使用这些函式时并不需要了解 MIDI 卡上的硬体介面。

要在播放音乐的准备期间打开一个 MIDI 输出设备，可以呼叫 midiOutOpen 函式：

```
error = midiOutOpen (&hMidiOut, wDeviceID, dwCallBack,

      dwCallBackData, dwFlags) ;
```

如果呼叫成功，则函式传回 0，否则传回错误代码。如果参数设定正确，则常见的一种错误就是 MIDI 设备已被其他程式使用。

该函式的第一个参数是指向 HMIDIOUT 型态变数的指标，它接收後面用於 MIDI 输出函式的 MIDI 输出代号。第二个参数是设备 ID。要使用真实的 MIDI 设备，这个参数范围可以是从 0 到小於由 midiOutGetNumDevs 传回的数值。您还可以使用 MIDIMAPPER，它在 MMSYSTEM.H 中定义为-1。大多数情况下，函式的後三个参数设定为 NULL 或 0。

一旦打开一个 MIDI 输出设备并获得了其代号，您就可以向该设备发送 MIDI 讯息。此时可以呼叫：

```
error = midiOutShortMsg (hMidiOut, dwMessage) ;
```

第一个参数是从 midiOutOpen 函式获得的代号。第二个参数是包装在 32 位元 DWORD 中的 1 位元组、2 位元组或者 3 位元组的讯息。我在前面讨论过，MIDI 讯息以状态位元组开始，後面是 0、1 或 2 位元组的资料。在 dwMessage 中，状态位元组是最不重要的，第一个资料位元组次之，第二个资料位元组再次之，最重要的位元组是 0。

例如，要在 MIDI 通道 5 上以 0x7F 的速度演奏中音 C（音符是 0x3C），则需要 3 位元组的 Note On 讯息：

0x95 0x3C 0x7F

midiOutShortMsg 的参数 dwMessage 等於 0x007F3C95。

三个基础的 MIDI 讯息是 Program Change（可为某一特定通道而改变乐器声音）、Note On 和 Note Off。打开一个 MIDI 输出设备後，应该从一条 Program Change 讯息开始，然後发送相同数量的 Note On 和 Note Off 讯息。

当您一直演奏您想演奏的音乐时，您可以重置 MIDI 输出设备以确保关闭所有的音符：

```
midiOutReset (hMidiOut) ;
```

然後关闭设备：

```
midiOutClose (hMidiOut) ;
```

使用低阶的 MIDI 输出 API 时，midiOutOpen、midiOutShortMsg、midiOutReset 和 midiOutClose 是您需要的四个基础函式。

现在让我们演奏一段音乐。BACHTOCC，如程式 22-8 所示，演奏了 J. S. Bach 著名的风琴演奏的 D 小调《Toccata and Fugue》中托卡塔部分的第一小节。

程式 22-8 BACHTOCC

```
BACHTOCC.C
/*-------------------------------------------------------------------------
---
      BACHTOCC.C --          Bach Toccata in D Minor (First Bar)
                                                (c) Charles Petzold, 1998
-------------------------------------------------------------------------
-*/

#include <windows.h>
#define ID_TIMER    1
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM) ;
TCHAR szAppName[] = TEXT ("BachTocc") ;
int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                PSTR    szCmdLine,    int
iCmdShow)
{
      HWND                  hwnd ;
      MSG                   msg ;
      WNDCLASS          wndclass ;

      wndclass.style                    = CS_HREDRAW | CS_VREDRAW ;
      wndclass.lpfnWndProc              = WndProc ;
      wndclass.cbClsExtra               = 0 ;
      wndclass.cbWndExtra               = 0 ;
      wndclass.hInstance                = hInstance ;
      wndclass.hIcon                    = LoadIcon (NULL, IDI_APPLICATION) ;
      wndclass.hCursor                  = LoadCursor (NULL, IDC_ARROW) ;
      wndclass.hbrBackground       = GetStockObject (WHITE_BRUSH) ;
      wndclass.lpszMenuName        = NULL ;
      wndclass.lpszClassName       = szAppName ;
      if (!RegisterClass (&wndclass))
      {
            MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                                szAppName, MB_ICONERROR) ;
                  return 0 ;
    }

      hwnd = CreateWindow (  szAppName, TEXT ("Bach Toccata in D Minor (First
Bar)"),
                        WS_OVERLAPPEDWINDOW,
```

```
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    CW_USEDEFAULT, CW_USEDEFAULT,
                    NULL, NULL, hInstance, NULL) ;

     if (!hwnd)
                         return 0 ;
     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg, NULL, 0, 0))
     {
               TranslateMessage (&msg) ;
               DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                                          int iData1,  int iData2)
{
     DWORD dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
     return midiOutShortMsg (hMidi, dwMessage) ;
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
     static struct
     {
               int iDur ;
               int iNote [2] ;
     }
     noteseq [] = {   110, 69, 81,  110, 67, 79,  990, 69, 81,  220, -1, -1,
               110, 67, 79,  110, 65, 77,  110, 64, 76,  110, 62, 74,
               220, 61, 73,  440, 62, 74, 1980, -1, -1,  110, 57, 69,
               110, 55, 67,  990, 57, 69,  220, -1, -1,  220, 52, 64,
               220, 53, 65,  220, 49, 61,  440, 50, 62, 1980, -1, -1 } ;

     static HMIDIOUT        hMidiOut ;
     static int                         iIndex ;
     int                                        i ;

     switch (message)
     {
     case  WM_CREATE:
                                  // Open MIDIMAPPER device

                    if (midiOutOpen (&hMidiOut, MIDIMAPPER, 0, 0, 0))
```

```
                        {
                                    MessageBeep (MB_ICONEXCLAMATION) ;
                                    MessageBox (     hwnd, TEXT ("Cannot open
MIDI output device!"),
                        szAppName, MB_ICONEXCLAMATION | MB_OK) ;
                                    return -1 ;
                        }
                              // Send Program Change messages for "Church Organ"

                        MidiOutMessage (hMidiOut, 0xC0,  0, 19, 0) ;
                        MidiOutMessage (hMidiOut, 0xC0, 12, 19, 0) ;

                        SetTimer (hwnd, ID_TIMER, 1000, NULL) ;
                        return 0 ;

      case  WM_TIMER:
                                    // Loop for 2-note polyphony

                        for (i = 0 ; i < 2 ; i++)
                        {
                                          // Note Off messages for previous note

                  if    (iIndex != 0 && noteseq[iIndex - 1].iNote[i] != -1)
           {
           MidiOutMessage (hMidiOut, 0x80,  0, noteseq[iIndex - 1].iNote[i], 0) ;

           MidiOutMessage (hMidiOut, 0x80, 12, noteseq[iIndex - 1].iNote[i], 0) ;
           }
           // Note On messages for new note

          if (iIndex != sizeof (noteseq) / sizeof (noteseq[0]) &&
              noteseq[iIndex].iNote[i] != -1)
            {
          MidiOutMessage (hMidiOut, 0x90,  0, noteseq[iIndex].iNote[i], 127) ;

          MidiOutMessage (hMidiOut, 0x90, 12,noteseq[iIndex].iNote[i], 127) ;
                                        }
                        }

                        if (iIndex != sizeof (noteseq) / sizeof (noteseq[0]))
                        {
                  SetTimer (hwnd, ID_TIMER, noteseq[iIndex++].iDur - 1, NULL) ;
                        }
                        else
                        {
                                          KillTimer (hwnd, ID_TIMER) ;
                                          DestroyWindow (hwnd) ;
                        }
```

```
                    return 0 ;

     case  WM_DESTROY:
                    midiOutReset (hMidiOut) ;
                    midiOutClose (hMidiOut) ;
                    PostQuitMessage (0) ;
                    return 0 ;
    }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

图 22-1 显示了 Bach 的 D 小调 Toccata 的第一小节。

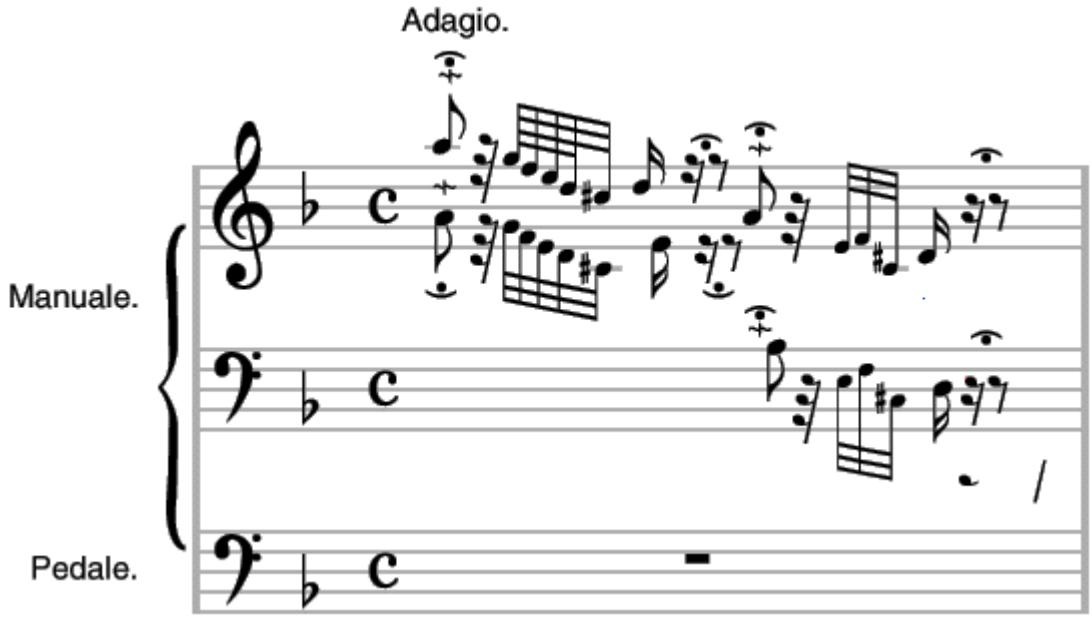

图 22-1 Bach 的 D 小调 Toccata and Fugue 的第一小节

在这里要做的就是把音乐转换成一系列的数值——基本键号和定时资讯，其中定时资讯表示发送 Note On（对应於风琴键按下）和 Note Off（释放键）讯息的时间。由於风琴键盘对速度不敏感，所以我们用相同的速度来演奏所有的音符。另外一个简化是忽略断奏（即，在连续的音符之间留下一个很短的停顿，以达到尖硬的效果）和连奏（在连续的音符之间有更圆润的重叠）之间的区别。我们假定一个音符结束後面紧接著下一个音符开始。

如果看得懂乐谱，那么您就会注意到托卡塔曲以两个平行的八度音阶开始。因此 BACHTOCC 建立了一个资料结构 noteseq 来储存一系列的音符持续时间以及两个键号。不幸的是，音乐持续进入第二小节就需要更特殊的方法来储存此资讯。我将四分音符的持续时间定义为 1760 毫秒，也就是说，八分音符（在音符或者休止符上有一个符尾）的持续时间是 880 毫秒，十六分音符（两个符尾）是 440 毫秒，三十二分音符（三个符尾）是 220 毫秒，六十四分音符（四个符尾）是 110 毫秒。

这第一小节中有两个波音——一个在第一个音符处，另一个在小节的中间。

这在乐谱上用带一条短竖线的曲线表示。在结构复杂的乐曲中，波音符号表示此音符实际应演奏为三个音符——标出的音符、比它低一个全音的音符，然後还是标出的音符。前两个音符演奏得要快，第三个音符要持续剩余的时间。例如，第一个音符是带波音的 A，则应演奏为 A、G、A。我将波音的前两个音符定义为六十四分音符，所以每个音符都持续 110 毫秒。

在第一小节还有四个延长符号。乐谱上表示为中间带点的半圆形。延长符号表示该音符在演奏时所持续的时间比标记的时间要长，通常由演奏者决定具体的时间。我对於延长符号延长了 50%的时间。

可以看到，即使是转换一小段看来简单直接的乐曲，例如 D 小调《Toccata》的开头，也并不是件容易的事！

noteseq 结构阵列包含了这一小节中平行的音符和休止符的三个数位。音符持续时间的後面是用於平行八度音阶的两个 MIDI 键号。例如，第一个音符是 A，持续时间是 110 毫秒。因为中音 C 的 MIDI 键号是 60，所以中音 C 上面的 A 的键号是 69，比 A 高一个八度音阶的键号是 81。因此，noteseq 阵列的前三个数是 110、69 和 81。我用音符值-1 表示休止符。

WM_CREATE 讯息处理期间，BACHTOCC 设定一个 Windows 计时器用於定时 1000 毫秒——表示乐曲从第 1 秒开始演奏——然後用 MIDIMAPPER 设备 ID 呼叫 midiOutOpen。

BACHTOCC 只需要一种乐器（风琴）的声音，所以只需要一个通道。为了简化 MIDI 讯息的发送，BACHTOCC 中还定义了一个小函式 MidiOutMessage。此函式接收 MIDI 输出代号、状态位元组、通道代号和两个位元组资料。其功能是把这些数字打包到一条 32 位元的讯息并呼叫 midiOutShortMsg。

在 WM_CREATE 讯息处理程序的後期，BACHTOCC 发送一条 Program Change 讯息来选择「教堂风琴」的声音。在 General MIDI 声音配置中，教堂风琴声音在 Program Change 讯息中用数位位元组 19 表示。实际演奏的音符出现在 WM_TIMER 讯息处理期间。用回圈来处理两个音符的多音。如果前一个音符还在演奏，BACHTOCC 就为该音符发送 Note Off 讯息。然後，如果下一个音符不是休止符，则向通道 0 和 12 发送 Note On 讯息。随後，重置 Windows 计时器，使其与 noteseq 结构中音符的持续时间一致。

音乐演奏完後，BACHTOCC 删除视窗。在 WM_DESTROY 讯息处理期间，程式呼叫 midiOutReset 和 midiOutClose，然後终止程式。

尽管 BACHTOCC 合理地处理和计算声音（即使还不完全像真人演奏风琴），但一般情况下用 Windows 计时器按这种方式来演奏音乐并不管用。问题在於 Windows 计时器是依据 PC 的系统时钟，其解析度不能满足音乐的要求。而且，

Windows 计时器不是同步的。这样，如果其他程式正忙於执行，则获得 WM_TIMER 讯息就会有轻微的延迟。如果程式不能立即处理这些讯息，就会放弃 WM_TIMER 讯息，这时的声音听起来一团糟。

因此，当 BACHTOCC 显示了如何呼叫低阶 MIDI 输出函式时，使用 Windows 计时器显然不适合精确的音乐创作。所以，Windows 还提供了一系列附加的计时器函式，使用低阶的 MIDI 输出函式时可以利用这些函式。这些函式的字首为 time，您可以利用它们将计时器的解析度设定到最小 1 毫秒。我将在本章结尾的 DRUM 程式向您展示使用这些函式的方法。

通过键盘演奏 MIDI 合成器

因为大多数 PC 使用者可能都没有连结在机器上的 MIDI 键盘，所以可以用每个人都有的键盘（上面全部的字母键和资料键）来代替。程式 22-9 所示的程式 KBMIDI 允许您用 PC 键盘来演奏电子音乐合成器——不管是连结在音效卡上的，还是挂接在 MIDI Out 埠的外部合成器。KBMIDI 让您完全控制 MIDI 输出设备（即内部或外部的合成器）、MIDI 通道和乐器声音。除了演奏时的趣味性以外，我还发现此程式对於开发 Windows 如何实作 MIDI 支援很有用。

程式 22-9 KBMIDI

```
KBMIDI.C
/*------------------------------------------------------------------------
      KBMIDI.C -- Keyboard MIDI Player
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------*
/

#include <windows.h>
// Defines for Menu IDs
// --------------------

#define           IDM_OPEN          0x100
#define           IDM_CLOSE               0x101
#define           IDM_DEVICE              0x200
#define           IDM_CHANNEL             0x300
#define           IDM_VOICE               0x400

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);
TCHAR                   szAppName [] = TEXT ("KBMidi") ;
HMIDIOUT          hMidiOut ;
int               iDevice = MIDIMAPPER, iChannel = 0, iVoice = 0, iVelocity = 64 ;
int                     cxCaps, cyChar, xOffset, yOffset ;

      // Structures and data for showing families and instruments on menu
```

```
      // ------------------------------------------------------------------------

typedef struct
{
      TCHAR * szInst ;
      int   iVoice ;
}
INSTRUMENT ;
typedef    struct
{
      TCHAR     * szFam ;
      INSTRUMENT  inst [8] ;
}
FAMILY ;
FAMILY     fam [16] = {

      TEXT ("Piano"),

                  TEXT ("Acoustic Grand Piano"), 0,
                  TEXT ("Bright Acoustic Piano"),1,
                  TEXT ("Electric Grand Piano"), 2,
                  TEXT ("Honky-tonk Piano"),    3,
                  TEXT ("Rhodes Piano"),        4,
                  TEXT ("Chorused Piano"),      5,
                  TEXT ("Harpsichord"),         6,
                  TEXT ("Clavinet"),           7,
            TEXT ("Chromatic Percussion"),
                  TEXT ("Celesta"),            8,
                  TEXT ("Glockenspiel"),        9,
                  TEXT ("Music Box"),          10,
                  TEXT ("Vibraphone"),         11,
                  TEXT ("Marimba"),            12,
                  TEXT ("Xylophone"),          13,
                  TEXT ("Tubular Bells"),      14,
                  TEXT ("Dulcimer"),           15,
            TEXT ("Organ"),
                  TEXT ("Hammond Organ"),      16,
                  TEXT ("Percussive Organ"),   17,
                  TEXT ("Rock Organ"),         18,
                  TEXT ("Church Organ"),       19,
                  TEXT ("Reed Organ"),         20,
                  TEXT ("Accordian"),          21,
                  TEXT ("Harmonica"),          22,
                  TEXT ("Tango Accordian"),    23,
            TEXT ("Guitar"),
                  TEXT ("Acoustic Guitar (nylon)"), 24,
                  TEXT ("Acoustic Guitar (steel)"), 25,
                  TEXT ("Electric Guitar (jazz)"),  26,
```

```
                TEXT ("Electric Guitar (clean)"), 27,
                TEXT ("Electric Guitar (muted)"), 28,
                TEXT ("Overdriven Guitar"),      29,
                TEXT ("Distortion Guitar"),      30,
                TEXT ("Guitar Harmonics"),       31,
           TEXT ("Bass"),
                TEXT ("Acoustic Bass"),          32,
                TEXT ("Electric Bass (finger)"),  33,
                TEXT ("Electric Bass (pick)"),    34,
                TEXT ("Fretless Bass"),          35,
                TEXT ("Slap Bass 1"),            36,
                TEXT ("Slap Bass 2"),            37,
                TEXT ("Synth Bass 1"),           38,
                TEXT ("Synth Bass 2"),           39,
                TEXT ("Strings"),
                TEXT ("Violin"),                40,
                TEXT ("Viola"),                 41,
                TEXT ("Cello"),                 42,
                TEXT ("Contrabass"),             43,
                TEXT ("Tremolo Strings"),        44,
                TEXT ("Pizzicato Strings"),      45,
                TEXT ("Orchestral Harp"),        46,
                TEXT ("Timpani"),               47,
      TEXT ("Ensemble"),
                TEXT ("String Ensemble 1"),      48,
                TEXT ("String Ensemble 2"),      49,
                TEXT ("Synth Strings 1"),        50,
                TEXT ("Synth Strings 2"),        51,
                TEXT ("Choir Aahs"),             52,
                TEXT ("Voice Oohs"),             53,
                TEXT ("Synth Voice"),            54,
                TEXT ("Orchestra Hit"),          55,
           TEXT ("Brass"),
                TEXT ("Trumpet"),               56,
                TEXT ("Trombone"),              57,
                TEXT ("Tuba"),                  58,
                TEXT ("Muted Trumpet"),          59,
                TEXT ("French Horn"),            60,
                TEXT ("Brass Section"),          61,
                TEXT ("Synth Brass 1"),          62,
                TEXT ("Synth Brass 2"),          63,
           TEXT ("Reed"),
                TEXT ("Soprano Sax"),            64,
                TEXT ("Alto Sax"),              65,
                TEXT ("Tenor Sax"),             66,
                TEXT ("Baritone Sax"),           67,
                TEXT ("Oboe"),                  68,
                TEXT ("English Horn"),           69,
```

```
                TEXT ("Bassoon"),               70,
                TEXT ("Clarinet"),              71,
           TEXT ("Pipe"),
                TEXT ("Piccolo"),               72,
                TEXT ("Flute "),                73,
                TEXT ("Recorder"),              74,
                TEXT ("Pan Flute"),             75,
                TEXT ("Bottle Blow"),           76,
                TEXT ("Shakuhachi"),            77,
                TEXT ("Whistle"),               78,
                TEXT ("Ocarina"),               79,
      TEXT ("Synth Lead"),
                TEXT ("Lead 1 (square)"),        80,
                TEXT ("Lead 2 (sawtooth)"),      81,
                TEXT ("Lead 3 (caliope lead)"),   82,
                TEXT ("Lead 4 (chiff lead)"),    83,
                TEXT ("Lead 5 (charang)"),       84,
                TEXT ("Lead 6 (voice)"),         85,
                TEXT ("Lead 7 (fifths)"),        86,
                TEXT ("Lead 8 (brass + lead)"),   87,
           TEXT ("Synth Pad"),
                TEXT ("Pad 1 (new age)"),        88,
                TEXT ("Pad 2 (warm)"),           89,
                TEXT ("Pad 3 (polysynth)"),      90,
                TEXT ("Pad 4 (choir)"),          91,
                TEXT ("Pad 5 (bowed)"),          92,
                TEXT ("Pad 6 (metallic)"),       93,
                TEXT ("Pad 7 (halo)"),           94,
                TEXT ("Pad 8 (sweep)"),          95,
           TEXT ("Synth Effects"),
                TEXT ("FX 1 (rain)"),            96,
                TEXT ("FX 2 (soundtrack)"),      97,
                TEXT ("FX 3 (crystal)"),         98,
                TEXT ("FX 4 (atmosphere)"),      99,
                TEXT ("FX 5 (brightness)"),     100,
                TEXT ("FX 6 (goblins)"),        101,
                TEXT ("FX 7 (echoes)"),         102,
                TEXT ("FX 8 (sci-fi)"),         103,
           TEXT ("Ethnic"),
                TEXT ("Sitar"),                 104,
                TEXT ("Banjo"),                 105,
                TEXT ("Shamisen"),              106,
                TEXT ("Koto"),                  107,
                TEXT ("Kalimba"),               108,
                TEXT ("Bagpipe"),               109,
                TEXT ("Fiddle"),                110,
                TEXT ("Shanai"),                111,
           TEXT ("Percussive"),
```

```
                TEXT ("Tinkle Bell"),          112,
                TEXT ("Agogo"),                113,
                TEXT ("Steel Drums"),          114,
                TEXT ("Woodblock"),            115,
                TEXT ("Taiko Drum"),           116,
                TEXT ("Melodic Tom"),          117,
                TEXT ("Synth Drum"),           118,
                TEXT ("Reverse Cymbal"),       119,
           TEXT ("Sound Effects"),
                TEXT ("Guitar Fret Noise"),     120,
                TEXT ("Breath Noise"),         121,
                TEXT ("Seashore"),             122,
                TEXT ("Bird Tweet"),           123,
                TEXT ("Telephone Ring"),       124,
                TEXT ("Helicopter"),           125,
                TEXT ("Applause"),             126,
                TEXT ("Gunshot"),              127
} ;

     // Data for translating scan codes to octaves and notes
     // ----------------------------------------------------

#define NUMSCANS   (sizeof key / sizeof key[0])
struct
{
        int                iOctave ;
     int                   iNote ;
     int                   yPos ;
     int                   xPos ;
     TCHAR *          szKey ;
}
key [] =
{
                              // Scan  Char  Oct  Note
                              // ----  ----  ---  ----
     -1,   -1,   1,    -1,   NULL, //    0     None
     -1,   -1,   -1,   -1,   NULL, //    1     Esc
     -1,   -1,   0,    0,    TEXT (""),   //       2     1
     5,    1,    0,    2,    TEXT ("C#"),     //    3     2   5       C#
     5,    3,    0,    4,    TEXT ("D#"),     //    4     3     5     D#
     -1,   -1,   0,    6,    TEXT (""), //    5     4
     5,    6,    0,    8,    TEXT ("F#"),     //    6     5     5     F#
     5,    8,    0,    10,   TEXT ("G#"),     //    7     6     5     G#
     5,    10,   0,    12,   TEXT ("A#"),     //    8     7     5     A#
     -1,   -1,   0,    14,   TEXT (""), //    9     8
     6,    1,    0,    16,   TEXT ("C#"),     //    10    9     6     C#
     6,    3,    0,    18,   TEXT ("D#"),     //    11          0     6     D#
     -1,   -1,   0,    20,   TEXT (""), //    12          -
```

```
     6,    6,    0,    22,   TEXT ("F#"),      //    13    =     6     F#
     -1,   -1,   -1,   -1,   NULL, //    14    Back

     -1,   -1,   -1,   -1,   NULL, //    15    Tab
     5,    0,    1,    1,    TEXT ("C"), //    16          q     5     C
     5,    2,    1,    3,    TEXT ("D"), //    17          w     5     D
          5,    4,    1,    5,    TEXT ("E"),       //    18          e     5
E
          5,    5,    1,    7,    TEXT ("F"),       //    19          r     5
F
          5,    7,    1,    9,    TEXT ("G"),       //    20          t     5
G
          5,    9,    1,    11,   TEXT ("A"),       //    21          y     5
A
          5,    11,   1,    13,   TEXT ("B"),       //    22          u     5
B
          6,    0,    1,    15,   TEXT ("C"),       //    23          i     6
C
          6,    2,    1,    17,   TEXT ("D"),       //    24          o     6
D
          6,    4,    1,    19,   TEXT ("E"),       //    25          p     6
E
          6,    5,    1,    21,   TEXT ("F"),       //    26          [     6
F
          6,    7,    1,    23,   TEXT ("G"),       //    27          ]     6
G
          -1,   -1,   -1,   -1,   NULL,             //    28    Ent
          -1,   -1,   -1,   -1,   NULL,             //    29    Ctrl
          3,    8,    2,    2,    TEXT ("G#"),      //    30          a     3
G#
          3,    10,   2,    4,    TEXT ("A#"),      //    31          s     3
A#
          -1,   -1,   2,    6,    TEXT (""),        //    32          d
          4,    1,    2,    8,    TEXT ("C#"),      //    33          f     4
C#
          4,    3,    2,    10,   TEXT ("D#"),      //    34          g     4
D#
          -1,   -1,   2,    12,   TEXT (""),        //    35          h
          4,    6,    2,    14,   TEXT ("F#"),      //    36          j     4
F#
          4,    8,    2,    16,   TEXT ("G#"),      //    37          k     4
G#
          4,    10,   2,    18,   TEXT ("A#"),      //    38          l     4
A#
          -1,   -1,   2,    20,   TEXT (""),        //    39          ;
          5,    1,    2,    22,   TEXT ("C#"),      //    40          '     5
C#
          -1,   -1,   -1,   -1,   NULL,             //    41          `
```

```
          -1,   -1,   -1,   -1,   NULL,            //    42    Shift
          -1,   -1,   -1,   -1,   NULL,            //    43          \
     (not line continuation)
          3,    9,    3,    3,    TEXT ("A"),      //    44          z     3
     A
          3,    11,   3,    5,    TEXT ("B"),      //    45          x     3
     B
          4,    0,    3,    7,    TEXT ("C"),      //    46          c     4
     C
     4,   2,    3,    9,    TEXT ("D"),      //    47          v     4     D
          4,    4,    3,    11,   TEXT ("E"),      //    48          b     4
     E
          4,    5,    3,    13,   TEXT ("F"),      //    49          n     4
     F
          4,    7,    3,    15,   TEXT ("G"),      //    50          m     4
     G
     4,   9,    3,    17,   TEXT ("A"),      //    51          ,     4     A
          4,    11,   3,    19,   TEXT ("B"),      //    52          .     4
     B
          5,    0,    3,    21,   TEXT ("C")       //    53          /     5
     C
} ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                                  PSTR    szCmdLine,    int
iCmdShow)
{
     MSG                         msg;
     HWND                        hwnd ;
     WNDCLASS               wndclass ;

     wndclass.style                              = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc              = WndProc ;
     wndclass.cbClsExtra               = 0 ;
     wndclass.cbWndExtra               = 0 ;
     wndclass.hInstance                = hInstance ;
     wndclass.hIcon                              =     LoadIcon     (NULL,
IDI_APPLICATION) ;
     wndclass.hCursor                         = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground       = GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName  = NULL ;
     wndclass.lpszClassName       = szAppName ;

     if (!RegisterClass (&wndclass))
     {
               MessageBox (NULL, TEXT ("This program requires Windows NT!"),
                                                       szAppName,
MB_ICONERROR) ;
```

```
                return 0 ;
    }

     hwnd = CreateWindow (  szAppName, TEXT ("Keyboard MIDI Player"),
           WS_OVERLAPPEDWINDOW | WS_HSCROLL | WS_VSCROLL,
           CW_USEDEFAULT, CW_USEDEFAULT,
           CW_USEDEFAULT, CW_USEDEFAULT,
           NULL, NULL, hInstance, NULL) ;

     if (!hwnd)
                     return 0 ;
     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd);

     while (GetMessage (&msg, NULL, 0, 0))
     {
                TranslateMessage (&msg) ;
                DispatchMessage (&msg) ;
     }
     return msg.wParam ;
}

// Create the program's menu (called from WndProc, WM_CREATE)
// ------------------------------------------------------------------------

HMENU CreateTheMenu (int iNumDevs)
{
     TCHAR                         szBuffer [32] ;
     HMENU                         hMenu, hMenuPopup, hMenuSubPopup ;
     int                           i, iFam, iIns ;
     MIDIOUTCAPS            moc ;

     hMenu = CreateMenu () ;
                           // Create "On/Off" popup menu
     hMenuPopup = CreateMenu () ;
     AppendMenu (hMenuPopup, MF_STRING, IDM_OPEN, TEXT ("&Open")) ;
     AppendMenu (hMenuPopup, MF_STRING | MF_CHECKED, IDM_CLOSE,
                TEXT ("&Closed")) ;
     AppendMenu (hMenu,     MF_STRING | MF_POPUP, (UINT) hMenuPopup,
                TEXT ("&Status")) ;

                     // Create "Device" popup menu

     hMenuPopup = CreateMenu () ;
                // Put MIDI Mapper on menu if it's installed
     if (!midiOutGetDevCaps (MIDIMAPPER, &moc, sizeof (moc)))
                     AppendMenu (hMenuPopup, MF_STRING, IDM_DEVICE + (int)
MIDIMAPPER,
```

```
                                                                moc.szPname) ;
     else
                    iDevice = 0 ;
                    // Add the rest of the MIDI devices
     for (i = 0 ; i < iNumDevs ; i++)
     {
               midiOutGetDevCaps (i, &moc, sizeof (moc)) ;
               AppendMenu   (hMenuPopup,   MF_STRING,   IDM_DEVICE   +   i,
moc.szPname) ;
    }

     CheckMenuItem (hMenuPopup, 0, MF_BYPOSITION | MF_CHECKED) ;
     AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
                                                  TEXT ("&Device")) ;
                    // Create "Channel" popup menu
     hMenuPopup = CreateMenu () ;
     for (i = 0 ; i < 16 ; i++)
     {
               wsprintf (szBuffer, TEXT ("%d"), i + 1) ;
               AppendMenu  (hMenuPopup,  MF_STRING  |  (i  ?  MF_UNCHECKED  :
MF_CHECKED),
                                IDM_CHANNEL + i, szBuffer) ;
     }

     AppendMenu (hMenu, MF_STRING | MF_POPUP, (UINT) hMenuPopup,
                                                  TEXT ("&Channel")) ;
                           // Create "Voice" popup menu
     hMenuPopup = CreateMenu () ;
     for (iFam = 0 ; iFam < 16 ; iFam++)
     {
     hMenuSubPopup = CreateMenu () ;
     for (iIns = 0 ; iIns < 8 ; iIns++)
          {
                           wsprintf (szBuffer, TEXT ("&%d.\t%s"), iIns + 1,

     fam[iFam].inst[iIns].szInst) ;
                           AppendMenu (hMenuSubPopup,
                  MF_STRING | (fam[iFam].inst[iIns].iVoice ?
                  MF_UNCHECKED : MF_CHECKED),
                  fam[iFam].inst[iIns].iVoice + IDM_VOICE,
                  szBuffer) ;
                     }

    wsprintf (szBuffer, TEXT ("&%c.\t%s"), 'A' + iFam,
    fam[iFam].szFam) ;
    AppendMenu (hMenuPopup, MF_STRING | MF_POPUP, (UINT) hMenuSubPopup,
    szBuffer) ;
   }
```

```
      AppendMenu (hMenu,     MF_STRING | MF_POPUP, (UINT) hMenuPopup,
       TEXT ("&Voice")) ;
      return hMenu ;
}

// Routines for simplifying MIDI output
// ------------------------------------

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                                                              int    iData1,
int iData2)
{
      DWORD dwMessage ;
      dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
      return midiOutShortMsg (hMidi, dwMessage) ;
}

DWORD MidiNoteOff (    HMIDIOUT hMidi, int iChannel, int iOct, int iNote, int
iVel)
{
      return MidiOutMessage (hMidi, 0x080, iChannel, 12 * iOct + iNote, iVel) ;
}

DWORD MidiNoteOn (     HMIDIOUT hMidi, int iChannel, int iOct, int iNote, int
iVel)
{
      return MidiOutMessage (      hMidi, 0x090, iChannel, 12 * iOct + iNote,
iVel) ;
}

DWORD MidiSetPatch (HMIDIOUT hMidi, int iChannel, int iVoice)
{
      return MidiOutMessage (hMidi, 0x0C0, iChannel, iVoice, 0) ;
}

DWORD MidiPitchBend (HMIDIOUT hMidi, int iChannel, int iBend)
{
      return MidiOutMessage (hMidi, 0x0E0, iChannel, iBend & 0x7F, iBend >> 7) ;
}

// Draw a single key on window
// ---------------------------------

VOID DrawKey (HDC hdc, int iScanCode, BOOL fInvert)
{
      RECT rc ;
      rc.left          = 3 * cxCaps * key[iScanCode].xPos / 2 + xOffset ;
      rc.top           = 3 * cyChar * key[iScanCode].yPos / 2 + yOffset ;
```

```
     rc.right         = rc.left + 3 * cxCaps ;
     rc.bottom   = rc.top  + 3 * cyChar / 2 ;

     SetTextColor     (hdc, fInvert ? 0x00FFFFFFul : 0x00000000ul) ;
     SetBkColor             (hdc, fInvert ? 0x00000000ul : 0x00FFFFFFul) ;

     FillRect (hdc, &rc, GetStockObject (fInvert ? BLACK_BRUSH : WHITE_BRUSH)) ;
     DrawText (hdc, key[iScanCode].szKey, -1, &rc,
                                        DT_SINGLELINE    |    DT_CENTER    |
DT_VCENTER) ;
     FrameRect (hdc, &rc, GetStockObject (BLACK_BRUSH)) ;
}

// Process a Key Up or Key Down message
// ------------------------------------

VOID ProcessKey (HDC hdc, UINT message, LPARAM lParam)
{
     int iScanCode, iOctave, iNote ;
     iScanCode = 0x0FF & HIWORD (lParam) ;
     if (iScanCode >= NUMSCANS)         // No scan codes over 53
                       return ;

     if ((iOctave = key[iScanCode].iOctave) == -1)             //   Non-music
key
                       return ;

     if (GetKeyState (VK_SHIFT) < 0)
                       iOctave += 0x20000000 & lParam ? 2 : 1 ;
     if (GetKeyState (VK_CONTROL) < 0)
                       iOctave -= 0x20000000 & lParam ? 2 : 1 ;
     iNote = key[iScanCode].iNote ;
     if (message == WM_KEYUP)           // For key up
    {
                MidiNoteOff (hMidiOut, iChannel, iOctave, iNote, 0) ;   // Note
off
                                             DrawKey (hdc, iScanCode, FALSE) ;
                                             return ;
    }

     if (0x40000000 & lParam)           // ignore typematics
                return ;

     MidiNoteOn (hMidiOut, iChannel, iOctave, iNote, iVelocity) ; // Note on
     DrawKey (hdc, iScanCode, TRUE) ;              // Draw the inverted key
}

// Window Procedure
```

```
// ---------------------

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static BOOL bOpened = FALSE ;
      HDC                         hdc ;
      HMENU                       hMenu ;
      int                         i, iNumDevs, iPitchBend, cxClient, cyClient ;
      MIDIOUTCAPS           moc ;
      PAINTSTRUCT      ps ;
      SIZE                        size ;
      TCHAR                       szBuffer [16] ;

      switch (message)
      {
      case  WM_CREATE:
                                  // Get size of capital letters in system font

                        hdc = GetDC (hwnd) ;

                        GetTextExtentPoint (hdc, TEXT ("M"), 1, &size) ;
                        cxCaps = size.cx ;
                        cyChar = size.cy ;

                        ReleaseDC (hwnd, hdc) ;

                                  // Initialize "Volume" scroll bar

                        SetScrollRange         (hwnd, SB_HORZ, 1, 127, FALSE) ;
                  SetScrollPos           (hwnd, SB_HORZ, iVelocity, TRUE) ;

                                  // Initialize "Pitch Bend" scroll bar

                        SetScrollRange         (hwnd, SB_VERT, 0, 16383, FALSE) ;
                        SetScrollPos           (hwnd, SB_VERT, 8192, TRUE) ;

                              // Get number of MIDI output devices and set up menu

                        if (0 == (iNumDevs = midiOutGetNumDevs ()))
                        {
                        MessageBeep (MB_ICONSTOP) ;
                        MessageBox (      hwnd, TEXT ("No MIDI output devices!"),
                            szAppName, MB_OK | MB_ICONSTOP) ;
                                        return -1 ;
                        }
                        SetMenu (hwnd, CreateTheMenu (iNumDevs)) ;
                        return 0 ;
```

```
case  WM_SIZE:
                cxClient = LOWORD (lParam) ;
                cyClient = HIWORD (lParam) ;

                xOffset = (cxClient - 25 * 3 * cxCaps / 2) / 2 ;
                yOffset = (cyClient - 11 * cyChar) / 2 + 5 * cyChar ;
                return 0 ;

case  WM_COMMAND:
                hMenu = GetMenu (hwnd) ;

                        // "Open" menu command

                if (LOWORD (wParam) == IDM_OPEN && !bOpened)
                {
                        if (midiOutOpen (&hMidiOut, iDevice, 0, 0, 0))
                        {
                    MessageBeep (MB_ICONEXCLAMATION) ;
                    MessageBox (hwnd, TEXT ("Cannot open MIDI device"),
                        szAppName, MB_OK | MB_ICONEXCLAMATION) ;
                        }
                        else
                        {
                    CheckMenuItem (hMenu, IDM_OPEN,  MF_CHECKED) ;
                    CheckMenuItem (hMenu, IDM_CLOSE, MF_UNCHECKED) ;

                    MidiSetPatch (hMidiOut, iChannel, iVoice) ;
                                bOpened = TRUE ;
       }
                        }

                                // "Close" menu command
                else if (LOWORD (wParam) == IDM_CLOSE && bOpened)
                {
                CheckMenuItem (hMenu, IDM_OPEN,  MF_UNCHECKED) ;
                CheckMenuItem (hMenu, IDM_CLOSE, MF_CHECKED) ;

                                // Turn all keys off and close device
                        for (i = 0 ; i < 16 ; i++)
                        MidiOutMessage (hMidiOut, 0xB0, i, 123, 0) ;
                        midiOutClose (hMidiOut) ;
                        bOpened = FALSE ;
                }

                                // Change MIDI "Device" menu command
                else if (   LOWORD (wParam) >= IDM_DEVICE - 1 &&
                                        LOWORD (wParam) <  IDM_CHANNEL)
```

```
                    {
                CheckMenuItem (hMenu, IDM_DEVICE + iDevice, MF_UNCHECKED) ;
                iDevice = LOWORD (wParam) - IDM_DEVICE ;
                CheckMenuItem (hMenu, IDM_DEVICE + iDevice, MF_CHECKED) ;

                                    // Close and reopen MIDI device

                                    if (bOpened)
                                    {
                          SendMessage (hwnd, WM_COMMAND, IDM_CLOSE, 0L) ;
                          SendMessage (hwnd, WM_COMMAND, IDM_OPEN,  0L) ;
                                    }
                          }

                                    // Change MIDI "Channel" menu command

                          else if (   LOWORD (wParam) >= IDM_CHANNEL &&
                                            LOWORD (wParam) <  IDM_VOICE)
                          {
                CheckMenuItem (hMenu, IDM_CHANNEL + iChannel, MF_UNCHECKED);
                iChannel = LOWORD (wParam) - IDM_CHANNEL ;
                CheckMenuItem (hMenu, IDM_CHANNEL + iChannel, MF_CHECKED) ;

                                    if (bOpened)
                                    MidiSetPatch (hMidiOut, iChannel, iVoice) ;
                          }

                                    // Change MIDI "Voice" menu command

                          else if (LOWORD (wParam) >= IDM_VOICE)
                          {
                CheckMenuItem (hMenu, IDM_VOICE + iVoice, MF_UNCHECKED) ;
                iVoice = LOWORD (wParam) - IDM_VOICE ;
                CheckMenuItem (hMenu, IDM_VOICE + iVoice, MF_CHECKED) ;

                                    if (bOpened)
                                    MidiSetPatch (hMidiOut, iChannel, iVoice) ;
                          }

                          InvalidateRect (hwnd, NULL, TRUE) ;
                          return 0 ;

                                // Process a Key Up or Key Down message

     case  WM_KEYUP:
     case  WM_KEYDOWN:
                          hdc = GetDC (hwnd) ;
```

```
                    if (bOpened)
                                        ProcessKey (hdc, message, lParam) ;

                    ReleaseDC (hwnd, hdc) ;
                    return 0 ;

                         // For Escape, turn off all notes and repaint

     case  WM_CHAR:
                    if (bOpened && wParam == 27)
                    {
                              for (i = 0 ; i < 16 ; i++)
                              MidiOutMessage (hMidiOut, 0xB0, i, 123, 0) ;

                              InvalidateRect (hwnd, NULL, TRUE) ;
                    }
                    return 0 ;

                    // Horizontal scroll: Velocity

     case  WM_HSCROLL:
                    switch (LOWORD (wParam))
                    {
                    case SB_LINEUP:             iVelocity -= 1 ;  break ;
                    case SB_LINEDOWN:           iVelocity += 1 ;  break ;
                    case SB_PAGEUP:             iVelocity -= 8 ;  break ;
                    case SB_PAGEDOWN:      iVelocity += 8 ;  break ;
               case SB_THUMBPOSITION:  iVelocity = HIWORD (wParam) ;  break ;
                         default:
     return 0 ;
        }
                    iVelocity = max (1, min (iVelocity, 127)) ;
                    SetScrollPos (hwnd, SB_HORZ, iVelocity, TRUE) ;
                    return 0 ;

                              // Vertical scroll:  Pitch Bend
     case  WM_VSCROLL:
                    switch (LOWORD (wParam))
                    {
          case SB_THUMBTRACK:    iPitchBend = 16383 - HIWORD (wParam) ; break ;
          case SB_THUMBPOSITION: iPitchBend = 8191 ;  break ;
          default:                                  return 0 ;
          }
                    iPitchBend = max (0, min (iPitchBend, 16383)) ;
                    SetScrollPos (hwnd, SB_VERT, 16383 - iPitchBend, TRUE) ;

                    if (bOpened)
                         MidiPitchBend (hMidiOut, iChannel, iPitchBend) ;
```

```
                    return 0 ;

     case  WM_PAINT:
                    hdc = BeginPaint (hwnd, &ps) ;

                    for (i = 0 ; i < NUMSCANS ; i++)
                                   if (key[i].xPos != -1)
                                                 DrawKey (hdc, i, FALSE) ;

                    midiOutGetDevCaps(iDevice,&moc,sizeof(MIDIOUTCAPS));
                    wsprintf (szBuffer, TEXT ("Channel %i"), iChannel + 1) ;

                          TextOut (  hdc, cxCaps, 1 * cyChar,
               Opened ? TEXT ("Open") : TEXT ("Closed"),
               bOpened ? 4 : 6) ;
                          TextOut (  hdc, cxCaps, 2 * cyChar, moc.szPname,
               lstrlen (moc.szPname)) ;
                          TextOut    (hdc, cxCaps, 3 * cyChar, szBuffer, lstrlen
(szBuffer)) ;
                          TextOut    (hdc, cxCaps, 4 * cyChar,
               fam[iVoice / 8].inst[iVoice % 8].szInst,
                          lstrlen (fam[iVoice / 8].inst[iVoice % 8].szInst)) ;

                          EndPaint (hwnd, &ps) ;
                          return 0 ;

     case  WM_DESTROY :
                          SendMessage (hwnd, WM_COMMAND, IDM_CLOSE, 0L) ;
                          PostQuitMessage (0) ;
                          return 0 ;
    }
     return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

执行 KBMIDI 时,视窗显示了键盘上的键与传统钢琴或风琴按键的对应方式。左下角的 Z 键以 110 Hz 的频率演奏 A。键盘的最下行，右边是中音 C，倒数第二行为其升音或降音。上面两行键继续按此规律变化，从中音 C 到 G#。这样，整个范围是三个八度音阶。另外，分别按 Shift 键和 Ctrl 键可使整个音域上升或下降 1 个八度音阶，这样有效的音域就是 5 个八度音阶。

不过，如果立即开始演奏，那么您将听不到任何声音。您必须先从「Status」功能表中选择「Open」，打开一个 MIDI 输出设备。如果埠打开成功，则按下一个键就向合成器发送一条 MIDI Note On 讯息，释放键则产生一条 Note Off 讯息。取决於键盘的按键特性，您可以同时演奏几个音符。

从「Status」功能表里选择「Close」来关闭 MIDI 设备。这对於需要在不终止 KBMIDI 程式的情况下执行 Windows 下的其他 MIDI 软体来说是很方便的。

「Device」功能表列出了已安装的 MIDI 输出设备，这些设备通过呼叫 midiOutGetDevCaps 函式获得。其中有些设备可能是 MIDI Out 埠连结的实际存在或不存在的外部合成器。列表还包括 MIDI Mapper 设备。这是从「控制台」的「多媒体」中选择的 MIDI 合成器。

「Channel」功能表用来选择从 1 到 16 的 MIDI 通道，内定状态下选择通道 1。KBMIDI 程式产生的所有 MIDI 讯息都发送到所选的通道。

KBMIDI 最後一个功能表项是「Voice」，它是一个双层功能表，用於选择 128 种乐器声音，这些声音在 General MIDI 规范中定义并在 Windows 中实作。这 128 种乐器声音分为 16 乐器组，每个乐器组有 8 种乐器。由於不同的 MIDI 键号对应於不同的泛音，所以这 128 种乐器声音也称为有旋律的声音。

General MIDI 中还定义了大量无旋律的打击乐器。要演奏打击乐器，可以从「Channel」功能表选择通道 10，还可以从「Voice」功能表选择第一种乐器声音（「Acoustic Grand Piano」）。这样，按不同的键就可以得到不同打击乐器的声音。从 MIDI 键号 35（低於中音 C 两个八度音阶的 B）到 81（高於中音 C 近两个八度音阶的 A），共有 47 种不同的打击乐器声音。在下面的 DRUM 程式中就利用了打击乐器通道。

KBMIDI 程式有水平和垂直卷动列。由於 PC 键盘对按键速度不敏感，所以用水平卷动列来控制音符速度。一般来说，这与演奏音符的音量一致。设定完水平卷动列以後，所有的 Note On 讯息都将使用这个速度。

垂直卷动列将产生一条称为「Pitch Bend」的 MIDI 讯息。要使用此特性，请按下一个或多个键，然後用滑鼠拖动卷动列。向上拖动卷动列音符频率将上升，向下拖动则频率下降。释放卷动列後将恢复正常的基音。

这两个卷动列要小心使用：因为拖动卷动列时，键盘讯息将不进入程式的讯息回圈。因此，如果按下一个键後就开始拖动卷动列，然後在完成拖动之前就释放了该键，那么音符仍将发声。所以，拖动卷动列时不要按下或者释放任何键。对功能表也有类似的规则：按著键时不要进行功能表选择。另外，在按下与释放某个键期间，不要用 Ctrl 或 Shift 键来改变八度音阶。

如果一个或者多个音符出现「粘滞现象」，即释放後继续发声，那么请按下 Esc 键。按下此键将通过向 MIDI 合成器的 16 个通道发送 16 条 All Notes Off 讯息，来关闭声音。

KBMIDI 没有资源描述档，而是通过搜索来建立的功能表。设备名称从 midiOutGetDevCaps 函式获得，乐器种类和名称则储存在程式的一个大资料结构中。

KBMIDI 定义了几个小函式来简化 MIDI 讯息。除了 Pitch Bend 讯息以外，

其他讯息都在前面讨论过了。Pitch Bend 讯息用两个 7 位元值组成一个 14 位元的音调弯曲等级：0 到 0x1FFF 之间的值降低基音，0x2001 到 0x3FFF 之间的值升高基音。

从「Status」功能表选择「Open」时，KBMIDI 为选择的设备呼叫 midiOutOpen；如呼叫成功，则呼叫 MidiSetPatch 函式。设备改变时，KBMIDI 必须关闭前一个设备，必要时再打开新设备。当改变 MIDI 设备、MIDI 通道、乐器声音时，KBMIDI 也必须呼叫 MidiSetPatch。

KBMIDI 通过处理 WM_KEYUP 讯息和 WM_KEYDOWN 讯息来控制音符的发音。KBMIDI 中用一个阵列把键盘扫描码映射成八度音阶和音符。例如，美国英语键盘上 Z 键的扫描码是 44，阵列将其标记为八度音阶是 3，音符是 9（即 A）。在 KBMIDI 的 MidiNoteOn 函式里，这些组合成了 MIDI 键号 45（即 12 乘以 3 再加上 9）。此资料结构也用於在视窗中画出键——每个键都有特定的水平和垂直位置，以及显示在矩形中的文字字串。

水平卷动列的处理是很直接的：所有需要做的就是储存新的速度级并设定新的卷动列的位置。但是处理垂直卷动列以控制音调弯曲的操作稍有一点特殊，它处理的卷动列命令只有两个：用滑鼠拖动卷动列时发生的 SB_THUMBTRACK，以及释放卷动列时的 SB_THUMBPOSITION。处理 SB_THUMBPOSITION 命令时，KBMIDI 将卷动列位置设定为中间等级，并呼叫 MidiPitchBend，其中参数值是 8192。

MIDI 击鼓器

有些打击乐器，如木琴或定音鼓，是「有旋律的」或「半音阶的」，因为它们可以用不同的音阶演奏乐曲。木琴用木板来对应不同的音阶，定音鼓也可以演奏曲调。这两种乐器及其他的有旋律的打击乐器都可以在 KBMIDI 的「Voice」功能表里选择。

但是，其他许多打击乐器都没有旋律，它们不能调音，而且通常含有太多的噪音，以致不能与某个基音相联系。在「General MIDI」规范中，这些没旋律的打击乐器声在通道 10 有效。不同的键号对应 47 种不同的打击乐器。

DRUM 程式，如程式 22-10 所示，是一个电脑击鼓器。此程式让您用 47 种不同的打击乐器的声音来构造最大到 32 个音符的一个序列，然後在选择的速度和音量下反复演奏这个序列。

程式 22-10　DRUM

```
DRUM.C
/*------------------------------------------------------------------------
-
      DRUM.C -- MIDI Drum Machine
```

```
                                    (c) Charles Petzold, 1998
------------------------------------------------------------------------------
*/

#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "drumtime.h"
#include "drumfile.h"
#include "resource.h"

LRESULT     CALLBACK WndProc             (HWND, UINT, WPARAM, LPARAM) ;
BOOL             CALLBACK AboutProc      (HWND, UINT, WPARAM, LPARAM) ;

void        DrawRectangle          (HDC, int, int, DWORD *, DWORD *) ;
void        ErrorMessage           (HWND, TCHAR *, TCHAR *) ;
void        DoCaption              (HWND, TCHAR *) ;
int         AskAboutSave           (HWND, TCHAR *) ;

TCHAR * szPerc [NUM_PERC] =
{
      TEXT ("Acoustic Bass Drum"),       TEXT ("Bass Drum 1"),
      TEXT ("Side Stick"),                     TEXT ("Acoustic Snare"),
      TEXT ("Hand Clap"),                TEXT ("Electric Snare"),
      TEXT ("Low Floor Tom"),            TEXT ("Closed High Hat"),
      TEXT ("High Floor Tom"),           TEXT ("Pedal High Hat"),
      TEXT ("Low Tom"),                  TEXT ("Open High Hat"),
      TEXT ("Low-Mid Tom"),              TEXT ("High-Mid Tom"),
      TEXT ("Crash Cymbal 1"),           TEXT ("High Tom"),
      TEXT ("Ride Cymbal 1"),            TEXT ("Chinese Cymbal"),
      TEXT ("Ride Bell"),                TEXT ("Tambourine"),
      TEXT ("Splash Cymbal"),            TEXT ("Cowbell"),
      TEXT ("Crash Cymbal 2"),           TEXT ("Vibraslap"),
      TEXT ("Ride Cymbal 2"),            TEXT ("High Bongo"),
      TEXT ("Low Bongo"),                TEXT ("Mute High Conga"),
      TEXT ("Open High Conga"),          TEXT ("Low Conga"),
      TEXT ("High Timbale"),             TEXT ("Low Timbale"),
      TEXT ("High Agogo"),               TEXT ("Low Agogo"),
      TEXT ("Cabasa"),                   TEXT ("Maracas"),
      TEXT ("Short Whistle"),            TEXT ("Long Whistle"),
      TEXT ("Short Guiro"),              TEXT ("Long Guiro"),
      TEXT ("Claves"),                   TEXT ("High Wood Block"),
      TEXT ("Low Wood Block"),           TEXT ("Mute Cuica"),
      TEXT ("Open Cuica"),               TEXT ("Mute Triangle"),
      TEXT ("Open Triangle")
} ;
```

```
TCHAR            szAppName   []    = TEXT ("Drum") ;
TCHAR            szUntitled  []    = TEXT ("(Untitled)") ;
TCHAR            szBuffer [80 + MAX_PATH] ;
HANDLE     hInst ;
int              cxChar, cyChar ;

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance,
                                              PSTR szCmdLine, int iCmdShow)
{
     HWND                              hwnd ;
     MSG                               msg ;
     WNDCLASS                     wndclass ;

     hInst = hInstance ;
     wndclass.style                    = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc              = WndProc ;
     wndclass.cbClsExtra               = 0 ;
     wndclass.cbWndExtra               = 0 ;
     wndclass.hInstance                = hInstance ;
     wndclass.hIcon                    = LoadIcon (hInstance, szAppName) ;
     wndclass.hCursor                  = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground       = GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName        = szAppName ;
     wndclass.lpszClassName       = szAppName ;

     if (!RegisterClass (&wndclass))
     {
          MessageBox (     NULL, TEXT ("This program requires Windows NT!"),
                                szAppName, MB_ICONERROR) ;
               return 0 ;
    }

     hwnd = CreateWindow (szAppName, NULL,
WS_OVERLAPPED | WS_CAPTION | WS_SYSMENU |
                         WS_MINIMIZEBOX | WS_HSCROLL | WS_VSCROLL,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         CW_USEDEFAULT, CW_USEDEFAULT,
                         NULL, NULL, hInstance, szCmdLine) ;

     ShowWindow (hwnd, iCmdShow) ;
     UpdateWindow (hwnd) ;

     while (GetMessage (&msg, NULL, 0, 0))
     {
               TranslateMessage (&msg) ;
               DispatchMessage (&msg) ;
     }
     return msg.wParam ;
```

```
}

LRESULT CALLBACK WndProc (   HWND hwnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
      static      BOOL        bNeedSave ;
      static      DRUM        drum ;
      static      HMENU             hMenu ;
      static      int         iTempo = 50, iIndexLast ;
      static      TCHAR             szFileName       [MAX_PATH],    szTitleName
[MAX_PATH] ;
      HDC                           hdc ;
      int                           i, x, y ;
      PAINTSTRUCT                   ps ;
      POINT                               point ;
      RECT                                rect ;
      TCHAR                         *     szError ;

      switch (message)
     {
      case  WM_CREATE:
                                    // Initialize DRUM structure

                        drum.iMsecPerBeat = 100 ;
                        drum.iVelocity    =  64 ;
                        drum.iNumBeats    =  32 ;

                        DrumSetParams (&drum) ;

                              // Other initialization

                        cxChar = LOWORD (GetDialogBaseUnits ()) ;
                        cyChar = HIWORD (GetDialogBaseUnits ()) ;

                        GetWindowRect (hwnd, &rect) ;
                        MoveWindow (hwnd,      rect.left, rect.top,
                  77 * cxChar, 29 * cyChar, FALSE) ;

                        hMenu = GetMenu (hwnd) ;

                              // Initialize "Volume" scroll bar

                        SetScrollRange         (hwnd, SB_HORZ, 1, 127, FALSE) ;
                        SetScrollPos     (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;

                              // Initialize "Tempo" scroll bar

                        SetScrollRange         (hwnd, SB_VERT, 0, 100, FALSE) ;
```

```
                    SetScrollPos           (hwnd, SB_VERT, iTempo, TRUE) ;

                    DoCaption (hwnd, szTitleName) ;
                    return 0 ;

     case  WM_COMMAND:
                    switch (LOWORD (wParam))
                    {
                    case  IDM_FILE_NEW:
                                   if    (  bNeedSave  &&  IDCANCEL  ==
AskAboutSave (hwnd, szTitleName))
                                         return 0 ;

                                   // Clear drum pattern

                                   for (i = 0 ; i < NUM_PERC ; i++)
                                   {
                                         drum.dwSeqPerc [i] = 0 ;
                                         drum.dwSeqPian [i] = 0 ;
                                   }

                                   InvalidateRect (hwnd, NULL, FALSE) ;
                                   DrumSetParams (&drum) ;
                                   bNeedSave = FALSE ;
                                   return 0 ;

                    case  IDM_FILE_OPEN:
                                                // Save previous file

                              if (bNeedSave && IDCANCEL ==
                                   AskAboutSave (hwnd, szTitleName))
                                         return 0 ;

                                                // Open the selected file

                    if (DrumFileOpenDlg (hwnd, szFileName, szTitleName))
                              {
                              szError = DrumFileRead (&drum, szFileName) ;

                              if (szError != NULL)
                              {
                              ErrorMessage (hwnd, szError, szTitleName) ;
                                                szTitleName [0] = '\0' ;
                                         }
                                         else
                                         {
                                                // Set new parameters
```

```
                                Tempo = (int) (50 *
                                (log10 (drum.iMsecPerBeat) - 1)) ;

            SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;
            SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;

            DrumSetParams (&drum) ;
            InvalidateRect (hwnd, NULL, FALSE) ;
            bNeedSave = FALSE ;
        }

        DoCaption (hwnd, szTitleName) ;
                        }
                        return 0 ;
        case  IDM_FILE_SAVE:
        case  IDM_FILE_SAVE_AS:
                                    // Save the selected file

        if ((LOWORD (wParam) == IDM_FILE_SAVE && szTitleName [0]) ||
              DrumFileSaveDlg (hwnd, szFileName, szTitleName))
                        {
              szError = DrumFileWrite (&drum, szFileName) ;

              if (szError != NULL)
              {
              ErrorMessage (hwnd, szError, szTitleName) ;
                                szTitleName [0] = '\0' ;
                                    }
                                    else
                                                bNeedSave = FALSE ;

                                    DoCaption (hwnd, szTitleName) ;
                        }
                        return 0 ;

        case  IDM_APP_EXIT:
                    SendMessage (hwnd, WM_SYSCOMMAND, SC_CLOSE, 0L) ;
                        return 0 ;

              case  IDM_SEQUENCE_RUNNING:
                        // Begin sequence

                        if (!DrumBeginSequence (hwnd))
                        {
                        ErrorMessage (hwnd,
                        TEXT ("Could not start MIDI sequence -- ")
                        TEXT ("MIDI Mapper device is unavailable!"),
                                        szTitleName) ;
```

```
                                    }
                                    else
                                    {
                                          CheckMenuItem                 (hMenu,
IDM_SEQUENCE_RUNNING,   MF_CHECKED) ;
                                          CheckMenuItem                 (hMenu,
IDM_SEQUENCE_STOPPED, MF_UNCHECKED) ;
                                    }
                                    return 0 ;

                              case  IDM_SEQUENCE_STOPPED:
                                          // Finish at end of sequence

                                          DrumEndSequence (FALSE) ;
                                          return 0 ;

                              case  IDM_APP_ABOUT:
                              DialogBox (hInst, TEXT ("AboutBox"), hwnd, AboutProc) ;
                                          return 0 ;
                              }
                              return 0 ;

      case  WM_LBUTTONDOWN:
      case  WM_RBUTTONDOWN:
                              hdc = GetDC (hwnd) ;

                                    // Convert mouse coordinates to grid coordinates

                              x =             LOWORD (lParam) / cxChar - 40 ;
                              y = 2 *     HIWORD (lParam) / cyChar -  2 ;
                                          // Set a new number of beats of sequence

                              if (x > 0 && x <= 32 && y < 0)
                              {
                              SetTextColor (hdc, RGB (255, 255, 255)) ;
            TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0, TEXT (":|"), 2);
                                          SetTextColor (hdc, RGB (0, 0, 0)) ;

                                          if (drum.iNumBeats % 4 == 0)
                              TextOut     (     hdc, (40 + drum.iNumBeats) * cxChar, 0,

      TEXT ("."), 1) ;

                                          drum.iNumBeats = x ;

            TextOut (hdc, (40 + drum.iNumBeats) * cxChar, 0, TEXT (":|"), 2);

                                          bNeedSave = TRUE ;
```

```
                    }

                              // Set or reset a percussion instrument beat

                    if (x >= 0 && x < 32 && y >= 0 && y < NUM_PERC)
                    {
                              if (message == WM_LBUTTONDOWN)
                                        drum.dwSeqPerc[y] ^= (1 << x) ;
                              else
                                        drum.dwSeqPian[y] ^= (1 << x) ;

               DrawRectangle (hdc, x, y, drum.dwSeqPerc, drum.dwSeqPian) ;

                              bNeedSave = TRUE ;
                    }

                    ReleaseDC (hwnd, hdc) ;
               DrumSetParams (&drum) ;
               return 0 ;

     case  WM_HSCROLL:
                              // Change the note velocity

                    switch (LOWORD (wParam))
                    {
                    case SB_LINEUP:
drum.iVelocity -= 1 ;  break ;
                    case SB_LINEDOWN:   drum.iVelocity += 1 ;  break ;
               case SB_PAGEUP:        drum.iVelocity -= 8 ;  break ;
                    case SB_PAGEDOWN:    drum.iVelocity += 8 ;  break ;
                    case SB_THUMBPOSITION:
                                        drum.iVelocity = HIWORD (wParam) ;
                                        break ;

                    default:
                                        return 0 ;
                    }

                    drum.iVelocity = max (1, min (drum.iVelocity, 127)) ;
                    SetScrollPos (hwnd, SB_HORZ, drum.iVelocity, TRUE) ;
                    DrumSetParams (&drum) ;
                    bNeedSave = TRUE ;
                    return 0 ;

     case  WM_VSCROLL:
                                        // Change the tempo

                    switch (LOWORD (wParam))
```

```
                    {
                    case SB_LINEUP:             iTempo -=  1 ;  break ;
                    case SB_LINEDOWN:           iTempo +=  1 ;  break ;
                    case SB_PAGEUP:             iTempo -= 10 ;  break ;
                    case SB_PAGEDOWN:           iTempo += 10 ;  break ;
                    case SB_THUMBPOSITION:
                                    iTempo = HIWORD (wParam) ;
                                    break ;

                    default:
                                return 0 ;
                    }

                    iTempo = max (0, min (iTempo, 100)) ;
                    SetScrollPos (hwnd, SB_VERT, iTempo, TRUE) ;

                    drum.iMsecPerBeat = (WORD) (10 * pow (100, iTempo / 100.0)) ;

                    DrumSetParams (&drum) ;
                    bNeedSave = TRUE ;
                    return 0 ;

    case  WM_PAINT:
                    hdc = BeginPaint (hwnd, &ps) ;

                    SetTextAlign (hdc, TA_UPDATECP) ;
                    SetBkMode (hdc, TRANSPARENT) ;

                                // Draw the text strings and horizontal lines
                    for (i = 0 ; i < NUM_PERC ; i++)
                    {
                                MoveToEx (hdc,    i & 1 ? 20 * cxChar : cxChar,
                     (2 * i + 3) * cyChar / 4, NULL) ;

                    TextOut (hdc, 0, 0, szPerc [i], lstrlen (szPerc [i])) ;

                                GetCurrentPositionEx (hdc, &point) ;

                MoveToEx    (hdc,  point.x + cxChar, point.y + cyChar / 2, NULL) ;
                LineTo      (hdc,       39 * cxChar, point.y + cyChar / 2) ;
                    }

                    SetTextAlign (hdc, 0) ;

                          // Draw rectangular grid, repeat mark, and beat marks

                    for (x = 0 ; x < 32 ; x++)
                    {
```

```
                                for (y = 0 ; y < NUM_PERC ; y++)
            DrawRectangle (hdc, x, y, drum.dwSeqPerc, drum.dwSeqPian) ;

                SetTextColor (   hdc, x == drum.iNumBeats - 1 ?
            RGB (0, 0, 0) : RGB (255, 255, 255)) ;

                TextOut (hdc, (41 + x) * cxChar, 0, TEXT (":|"), 2) ;

                SetTextColor (hdc, RGB (0, 0, 0)) ;

                if (x % 4 == 0)
            TextOut (hdc, (40 + x) * cxChar, 0, TEXT ("."), 1) ;
                }

                EndPaint (hwnd, &ps) ;
                return 0 ;

     case  WM_USER_NOTIFY:
                                        // Draw the "bouncing ball"

                     hdc = GetDC (hwnd) ;

                     SelectObject (hdc, GetStockObject (NULL_PEN)) ;
                     SelectObject (hdc, GetStockObject (WHITE_BRUSH)) ;

                     for (i = 0 ; i < 2 ; i++)
                     {
                                x = iIndexLast ;
                                y = NUM_PERC + 1 ;

                Ellipse (hdc, (x + 40) * cxChar, (2 * y + 3) * cyChar / 4,
                                (x + 41) * cxChar, (2 * y + 5) * cyChar / 4);

                                iIndexLast = wParam ;
                                SelectObject      (hdc,      GetStockObject
(BLACK_BRUSH)) ;
                     }

                     ReleaseDC (hwnd, hdc) ;
                     return 0 ;

     case  WM_USER_ERROR:
                ErrorMessage (hwnd, TEXT ("Can't set timer event for tempo"),
                                szTitleName) ;
               // fall through
     case  WM_USER_FINISHED:
                     DrumEndSequence (TRUE) ;
                     CheckMenuItem      (hMenu,      IDM_SEQUENCE_RUNNING,
```

```
      MF_UNCHECKED) ;
                        CheckMenuItem      (hMenu,      IDM_SEQUENCE_STOPPED,
      MF_CHECKED) ;
                        return 0 ;

      case  WM_CLOSE:
                        if  (!bNeedSave  ||  IDCANCEL  !=  AskAboutSave  (hwnd,
szTitleName))
                              DestroyWindow (hwnd) ;

                        return 0 ;

      case  WM_QUERYENDSESSION:
                        if  (!bNeedSave  ||  IDCANCEL  !=  AskAboutSave  (hwnd,
szTitleName))
                              return 1L ;

                        return 0 ;

      case  WM_DESTROY:
                        DrumEndSequence (TRUE) ;
                        PostQuitMessage (0) ;
                        return 0 ;
      }
      return DefWindowProc (hwnd, message, wParam, lParam) ;
}

BOOL CALLBACK AboutProc (    HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
{
      switch (message)
      {
      case  WM_INITDIALOG:
                        return TRUE ;

      case  WM_COMMAND:
                        switch (LOWORD (wParam))
                        {
                        case  IDOK:
                                          EndDialog (hDlg, 0) ;
                                          return TRUE ;
                        }
                        break ;
      }
      return FALSE ;
}

void DrawRectangle (   HDC hdc, int x, int y, DWORD * dwSeqPerc,
```

```
                                        DWORD * dwSeqPian)
{
      int iBrush ;
      if (dwSeqPerc [y] & dwSeqPian [y] & (1L << x))
                        iBrush = BLACK_BRUSH ;
      else if (dwSeqPerc [y] & (1L << x))
                                    iBrush = DKGRAY_BRUSH ;
      else if (dwSeqPian [y] & (1L << x))
                                    iBrush = LTGRAY_BRUSH ;
      else
                        iBrush = WHITE_BRUSH ;
      SelectObject (hdc, GetStockObject (iBrush)) ;
      Rectangle (hdc, (x + 40) * cxChar , (2 * y + 4) * cyChar / 4,
                        (x + 41) * cxChar + 1, (2 * y + 6) * cyChar / 4 + 1) ;
}

void ErrorMessage (HWND hwnd, TCHAR * szError, TCHAR * szTitleName)
{
      wsprintf (szBuffer, szError,
                                    (LPSTR) (szTitleName [0] ? szTitleName :
szUntitled)) ;
      MessageBeep (MB_ICONEXCLAMATION) ;
      MessageBox (hwnd, szBuffer, szAppName, MB_OK | MB_ICONEXCLAMATION) ;
}

void DoCaption (HWND hwnd, TCHAR * szTitleName)
{
      wsprintf (szBuffer, TEXT ("MIDI Drum Machine - %s"),
                                        (LPSTR) (szTitleName [0] ? szTitleName :
szUntitled)) ;
      SetWindowText (hwnd, szBuffer) ;
}

int AskAboutSave (HWND hwnd, TCHAR * szTitleName)
{
      int iReturn ;
      wsprintf (szBuffer, TEXT ("Save current changes in %s?"),
                                        (LPSTR) (szTitleName [0] ? szTitleName :
szUntitled)) ;
      iReturn = MessageBox ( hwnd, szBuffer, szAppName,
                              MB_YESNOCANCEL | MB_ICONQUESTION) ;

      if (iReturn == IDYES)
                        if (!SendMessage (hwnd, WM_COMMAND, IDM_FILE_SAVE, 0))
                                        iReturn = IDCANCEL ;
      return iReturn ;
}
```

DRUMTIME.H

```
/*------------------------------------------------------------------------
      DRUMTIME.H Header File for Time Functions for DRUM Program
------------------------------------------------------------------------
*/

#define NUM_PERC                                    47
#define WM_USER_NOTIFY            (WM_USER + 1)
#define WM_USER_FINISHED          (WM_USER + 2)
#define WM_USER_ERROR                   (WM_USER + 3)

#pragma pack(push, 2)
typedef struct
{
      short iMsecPerBeat ;
      short iVelocity ;
      short iNumBeats ;
      DWORD dwSeqPerc [NUM_PERC] ;
      DWORD dwSeqPian [NUM_PERC] ;
}
DRUM, * PDRUM ;
#pragma pack(pop)
void DrumSetParams                        (PDRUM) ;
BOOL DrumBeginSequence       (HWND)  ;
void DrumEndSequence               (BOOL)  ;
 DRUMTIME.C
/*------------------------------------------------------------------------
---
      DRUMFILE.C --    Timer Routines for DRUM
                                              (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/

#include <windows.h>
#include "drumtime.h"

#define minmax(a,x,b) (min (max (x, a), b))
#define TIMER_RES   5
void CALLBACK DrumTimerFunc (UINT, UINT, DWORD, DWORD, DWORD) ;
BOOL                  bSequenceGoing, bEndSequence ;
DRUM                  drum ;
HMIDIOUT         hMidiOut ;
HWND                  hwndNotify ;
int                   iIndex ;
UINT                  uTimerRes, uTimerID ;

DWORD MidiOutMessage ( HMIDIOUT hMidi, int iStatus, int iChannel,
                                                              int iData1, int
iData2)
```

```
{
      DWORD dwMessage ;
      dwMessage = iStatus | iChannel | (iData1 << 8) | (iData2 << 16) ;
      return midiOutShortMsg (hMidi, dwMessage) ;
}

void DrumSetParams (PDRUM pdrum)
{
      CopyMemory (&drum, pdrum, sizeof (DRUM)) ;
}

BOOL DrumBeginSequence (HWND hwnd)
{
      TIMECAPS tc ;
      hwndNotify = hwnd ;                       //  Save  window  handle  for
notification
      DrumEndSequence (TRUE) ;     // Stop current sequence if running

                        // Open the MIDI Mapper output port

      if (midiOutOpen (&hMidiOut, MIDIMAPPER, 0, 0, 0))
                        return FALSE ;
                  // Send Program Change messages for channels 9 and 0
      MidiOutMessage (hMidiOut, 0xC0, 9, 0, 0) ;
      MidiOutMessage (hMidiOut, 0xC0, 0, 0, 0) ;

                  // Begin sequence by setting a timer event
      timeGetDevCaps (&tc, sizeof (TIMECAPS)) ;
      uTimerRes = minmax (tc.wPeriodMin, TIMER_RES, tc.wPeriodMax) ;
      timeBeginPeriod (uTimerRes) ;

      uTimerID = timeSetEvent(max ((UINT) uTimerRes, (UINT) drum.iMsecPerBeat),
                uTimerRes, DrumTimerFunc, 0, TIME_ONESHOT) ;

      if (uTimerID == 0)
      {
                  timeEndPeriod (uTimerRes) ;
                  midiOutClose (hMidiOut) ;
                  return FALSE ;
      }

      iIndex = -1 ;
      bEndSequence = FALSE ;
      bSequenceGoing = TRUE ;

      return TRUE ;
}
```

```
void DrumEndSequence (BOOL bRightAway)
{
      if (bRightAway)
      {
                  if (bSequenceGoing)
                  {
                                          // stop the timer
                              if (uTimerID)
                                          timeKillEvent (uTimerID) ;
                                          timeEndPeriod (uTimerRes) ;

      // turn off all notes
                                       MidiOutMessage (hMidiOut, 0xB0, 9, 123, 0) ;
                                       MidiOutMessage (hMidiOut, 0xB0, 0, 123, 0) ;
     // close the MIDI port midiOutClose (hMidiOut) ; bSequenceGoing = FALSE ;
                              }
      }
      else
            bEndSequence = TRUE ;
}

void CALLBACK DrumTimerFunc (     UINT  uID, UINT uMsg, DWORD dwUser,
                              DWORD dw1, DWORD dw2)
{
      static DWORD     dwSeqPercLast [NUM_PERC], dwSeqPianLast [NUM_PERC] ;
      int                         i ;

                              // Note Off messages for channels 9 and 0

      if (iIndex != -1)
      {
                  for (i = 0 ; i < NUM_PERC ; i++)
            {
                              if (dwSeqPercLast[i] & 1 << iIndex)
                                    MidiOutMessage (hMidiOut, 0x80, 9, i + 35, 0) ;
                              if (dwSeqPianLast[i] & 1 << iIndex)
                                    MidiOutMessage (hMidiOut, 0x80, 0, i + 35, 0) ;
                  }
      }

                  // Increment index and notify window to advance bouncing ball
      iIndex = (iIndex + 1) % drum.iNumBeats ;
      PostMessage (hwndNotify, WM_USER_NOTIFY, iIndex, timeGetTime ()) ;

                  // Check if ending the sequence
      if (bEndSequence && iIndex == 0)
      {
                  PostMessage (hwndNotify, WM_USER_FINISHED, 0, 0L) ;
```

```
                return ;
     }

                // Note On messages for channels 9 and 0
     for (i = 0 ; i < NUM_PERC ; i++)
     {
                if (drum.dwSeqPerc[i] & 1 << iIndex)
                                MidiOutMessage (hMidiOut, 0x90, 9, i + 35,
drum.iVelocity) ;
                if (drum.dwSeqPian[i] & 1 << iIndex)
                                MidiOutMessage (hMidiOut, 0x90, 0, i + 35,
drum.iVelocity) ;
                           dwSeqPercLast[i] = drum.dwSeqPerc[i] ;
                           dwSeqPianLast[i] = drum.dwSeqPian[i] ;
           }
                           // Set a new timer event
     uTimerID = timeSetEvent (max ((int) uTimerRes, drum.iMsecPerBeat),
               uTimerRes, DrumTimerFunc, 0, TIME_ONESHOT) ;
     if (uTimerID == 0)
     {
                PostMessage (hwndNotify, WM_USER_ERROR, 0, 0) ;
     }
}
```

DRUMFILE.H

```
/*------------------------------------------------------------------------
-
     DRUMFILE.H Header File for File I/O Routines for DRUM
------------------------------------------------------------------------
-*/

BOOL             DrumFileOpenDlg   (HWND, TCHAR *, TCHAR *) ;
BOOL             DrumFileSaveDlg   (HWND, TCHAR *, TCHAR *) ;

TCHAR *     DrumFileWrite          (DRUM *, TCHAR *) ;
TCHAR *     DrumFileRead           (DRUM *, TCHAR *) ;
```

DRUMFILE.C

```
/*------------------------------------------------------------------------
--
     DRUMFILE.C --          File I/O Routines for DRUM
                                                (c) Charles Petzold, 1998
------------------------------------------------------------------------
-*/
#include <windows.h>
#include <commdlg.h>
#include "drumtime.h"
#include "drumfile.h"

OPENFILENAME ofn = { sizeof (OPENFILENAME) } ;
```

```
TCHAR * szFilter[] = { TEXT ("Drum Files (*.DRM)"),
                       TEXT ("*.drm"), TEXT ("") } ;

TCHAR szDrumID       [] = TEXT ("DRUM") ;
TCHAR szListID         []    = TEXT ("LIST") ;
TCHAR szInfoID         []    = TEXT ("INFO") ;
TCHAR szSoftID         []    = TEXT ("ISFT") ;
TCHAR szDateID         []    = TEXT ("ISCD") ;
TCHAR szFmtID    []    = TEXT ("fmt ") ;
TCHAR szDataID         []    = TEXT ("data") ;
char  szSoftware     [] = "DRUM by Charles Petzold, Programming Windows" ;

TCHAR szErrorNoCreate        []    =     TEXT  ("File %s could not be opened for
writing.");
TCHAR szErrorCannotWrite          []    =     TEXT  ("File  %s  could  not  be
written to. ") ;
TCHAR szErrorNotFound        []    =     TEXT  ("File %s not found or cannot be
opened.") ;
TCHAR szErrorNotDrum         []    =     TEXT  ("File %s is not a standard DRUM
file.") ;
TCHAR szErrorUnsupported          []    =     TEXT  ("File %s is not a supported
DRUM file.") ;
TCHAR szErrorCannotRead           []    =     TEXT  ("File   %s   cannot   be
read.") ;

BOOL DrumFileOpenDlg (HWND hwnd, TCHAR * szFileName, TCHAR * szTitleName)
{
           ofn.hwndOwner                    = hwnd ;
           ofn.lpstrFilter                        = szFilter [0] ;
           ofn.lpstrFile                          = szFileName ;
           ofn.nMaxFile                     = MAX_PATH ;
           ofn.lpstrFileTitle                   = szTitleName ;
           ofn.nMaxFileTitle                      = MAX_PATH ;
           ofn.Flags                                          =
OFN_CREATEPROMPT ;
           ofn.lpstrDefExt                        = TEXT ("drm") ;

     return GetOpenFileName (&ofn) ;
}

BOOL DrumFileSaveDlg ( HWND hwnd, TCHAR * szFileName,
                                          TCHAR * szTitleName)
{
           ofn.hwndOwner                    = hwnd ;
           ofn.lpstrFilter                        = szFilter [0] ;
           ofn.lpstrFile                          = szFileName ;
           ofn.nMaxFile                     = MAX_PATH ;
           ofn.lpstrFileTitle                   = szTitleName ;
```

```
            ofn.nMaxFileTitle                       = MAX_PATH ;
            ofn.Flags                                           =
OFN_OVERWRITEPROMPT ;
            ofn.lpstrDefExt                         = TEXT ("drm") ;

      return GetSaveFileName (&ofn) ;
}

TCHAR * DrumFileWrite (DRUM * pdrum, TCHAR * szFileName)
{
      char                        szDateBuf [16] ;
      HMMIO                       hmmio ;
      int                         iFormat = 2 ;
      MMCKINFO              mmckinfo [3] ;
      SYSTEMTIME            st ;
      WORD                        wError = 0 ;

      memset (mmckinfo, 0, 3 * sizeof (MMCKINFO)) ;
                                  // Recreate the file for writing
      if ((hmmio = mmioOpen (szFileName, NULL,
                                  MMIO_CREATE | MMIO_WRITE | MMIO_ALLOCBUF)) == NULL)
                        return szErrorNoCreate ;
                                  // Create a "RIFF" chunk with a "CPDR" type
      mmckinfo[0].fccType = mmioStringToFOURCC (szDrumID, 0) ;
      wError |= mmioCreateChunk (hmmio, &mmckinfo[0], MMIO_CREATERIFF) ;
                                  // Create "LIST" sub-chunk with an "INFO" type
      mmckinfo[1].fccType = mmioStringToFOURCC (szInfoID, 0) ;
      wError |= mmioCreateChunk (hmmio, &mmckinfo[1], MMIO_CREATELIST) ;
                                  // Create "ISFT" sub-sub-chunk
      mmckinfo[2].ckid = mmioStringToFOURCC (szSoftID, 0) ;
      wError |= mmioCreateChunk (hmmio, &mmckinfo[2], 0) ;
      wError |= (mmioWrite (hmmio, szSoftware,      sizeof (szSoftware)) !=
                           sizeof (szSoftware)) ;
      wError |= mmioAscend (hmmio, &mmckinfo[2], 0) ;
                                  // Create a time string
      GetLocalTime (&st) ;
      wsprintfA (szDateBuf, "%04d-%02d-%02d", st.wYear, st.wMonth, st.wDay) ;
                                  // Create "ISCD" sub-sub-chunk
      mmckinfo[2].ckid = mmioStringToFOURCC (szDateID, 0) ;
      wError |= mmioCreateChunk (hmmio, &mmckinfo[2], 0) ;
      wError |= (mmioWrite (hmmio, szDateBuf, (strlen (szDateBuf) + 1)) !=
                                                      (int)       (strlen
(szDateBuf) + 1)) ;
      wError |= mmioAscend (hmmio, &mmckinfo[2], 0) ;
      wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;

                                  // Create "fmt " sub-chunk
      mmckinfo[1].ckid = mmioStringToFOURCC (szFmtID, 0) ;
```

```
      wError |= mmioCreateChunk (hmmio, &mmckinfo[1], 0) ;
      wError |= (mmioWrite (hmmio, (PSTR) &iFormat,       sizeof (int)) !=
                sizeof (int)) ;
      wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;
                                  // Create the "data" sub-chunk
      mmckinfo[1].ckid = mmioStringToFOURCC (szDataID, 0) ;
      wError |= mmioCreateChunk (hmmio, &mmckinfo[1], 0) ;
      wError |= (mmioWrite (hmmio, (PSTR) pdrum, sizeof (DRUM)) !=
                sizeof (DRUM)) ;
      wError |= mmioAscend (hmmio, &mmckinfo[1], 0) ;
      wError |= mmioAscend (hmmio, &mmckinfo[0], 0) ;

                                  // Clean up and return
      wError |= mmioClose (hmmio, 0) ;
      if (wError)
      {
                  mmioOpen (szFileName, NULL, MMIO_DELETE) ;
                  return szErrorCannotWrite ;
      }
      return NULL ;
}

TCHAR * DrumFileRead (DRUM * pdrum, TCHAR * szFileName)
{
      DRUM                    drum ;
      HMMIO                   hmmio ;
      int                     i, iFormat ;
      MMCKINFO          mmckinfo [3] ;

      ZeroMemory  (mmckinfo, 2 * sizeof (MMCKINFO)) ;

                              // Open the file

      if ((hmmio = mmioOpen (szFileName, NULL, MMIO_READ)) == NULL)
                  return szErrorNotFound ;
                              // Locate a "RIFF" chunk with a "DRUM" form-type
      mmckinfo[0].ckid = mmioStringToFOURCC (szDrumID, 0) ;
      if (mmioDescend (hmmio, &mmckinfo[0], NULL, MMIO_FINDRIFF))
      {
                  mmioClose (hmmio, 0) ;
            return szErrorNotDrum ;
     }

                  // Locate, read, and verify the "fmt " sub-chunk
      mmckinfo[1].ckid = mmioStringToFOURCC (szFmtID, 0) ;
      if (mmioDescend (hmmio, &mmckinfo[1], &mmckinfo[0], MMIO_FINDCHUNK))
      {
                  mmioClose (hmmio, 0) ;
```

```
                return szErrorNotDrum ;
     }

     if (mmckinfo[1].cksize != sizeof (int))
     {
                mmioClose (hmmio, 0) ;
          return szErrorUnsupported ;
   }

     if (mmioRead (hmmio, (PSTR) &iFormat, sizeof (int)) != sizeof (int))
   {
                mmioClose (hmmio, 0) ;
                return szErrorCannotRead ;
   }

     if (iFormat != 1 && iFormat != 2)
   {
                mmioClose (hmmio, 0) ;
                return szErrorUnsupported ;
   }

                // Go to end of "fmt " sub-chunk
     mmioAscend (hmmio, &mmckinfo[1], 0) ;
                // Locate, read, and verify the "data" sub-chunk
     mmckinfo[1].ckid = mmioStringToFOURCC (szDataID, 0) ;
     if (mmioDescend (hmmio, &mmckinfo[1], &mmckinfo[0], MMIO_FINDCHUNK))
   {
                mmioClose (hmmio, 0) ;
                return szErrorNotDrum ;
   }

     if (mmckinfo[1].cksize != sizeof (DRUM))
     {
                mmioClose (hmmio, 0) ;
                return szErrorUnsupported ;
   }

     if (mmioRead (hmmio, (LPSTR) &drum, sizeof (DRUM)) != sizeof (DRUM))
     {
                mmioClose (hmmio, 0) ;
                return szErrorCannotRead ;
   }

                      // Close the file
     mmioClose (hmmio, 0) ;
                // Convert format 1 to format 2 and copy the DRUM structure data
     if (iFormat == 1)
     {
```

```
                for (i = 0 ; i < NUM_PERC ; i++)
                {
                        drum.dwSeqPerc [i] = drum.dwSeqPian [i] ;
                        drum.dwSeqPian [i] = 0 ;
                }
        }

            memcpy (pdrum, &drum, sizeof (DRUM)) ;
            return NULL ;
}
```

DRUM.RC （摘录）

```
//Microsoft Developer Studio generated resource script.
#include "resource.h"
#include "afxres.h"

//////////////////////////////////////////////////////////////////////////
/
// Menu
DRUM  MENU DISCARDABLE
BEGIN
      POPUP "&File"
      BEGIN
       MENUITEM "&New",      IDM_FILE_NEW
       MENUITEM "&Open...",    IDM_FILE_OPEN
       MENUITEM "&Save",             IDM_FILE_SAVE
       MENUITEM "Save &As...", IDM_FILE_SAVE_AS
       MENUITEM SEPARATOR
       MENUITEM "E&xit",      IDM_APP_EXIT
      END
      POPUP "&Sequence"
      BEGIN
       MENUITEM "&Running",    IDM_SEQUENCE_RUNNING
       MENUITEM "&Stopped",    IDM_SEQUENCE_STOPPED
       , CHECKED
      END
      POPUP "&Help"
      BEGIN
       MENUITEM "&About...",   IDM_APP_ABOUT
      END
END

//////////////////////////////////////////////////////////////////////////
/
// Icon
DRUM         ICON          DISCARDABLE        "drum.ico"

//////////////////////////////////////////////////////////////////////////
/
```

```
// Dialog
ABOUTBOX DIALOG DISCARDABLE  20, 20, 160, 164
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
DEFPUSHBUTTON  "OK",IDOK,54,143,50,14
ICON       "DRUM",IDC_STATIC,8,8,21,20
CTEXT      "DRUM",IDC_STATIC,34,12,90,8
CTEXT       "MIDI Drum Machine",IDC_STATIC,7,36,144,8
CONTROL     "",IDC_STATIC,"Static",SS_BLACKFRAME,8,88,144,46
LTEXT       "Left Button:\t\tDrum sounds",IDC_STATIC,12,92,136,8
LTEXT       "Right Button:\t\tPiano sounds",IDC_STATIC,12,102,136,8
LTEXT            "Horizontal Scroll:\t\tVelocity",IDC_STATIC,12,112,136,8
LTEXT            "Vertical Scroll:\t\tTempo",IDC_STATIC,12,122,136,8
CTEXT       "Copyright (c) Charles Petzold, 1998",IDC_STATIC,8,48, 144,8
CTEXT       """Programming Windows,"" 5th Edition",IDC_STATIC,8,60, 144,8
END
```

RESOURCE.H （摘录）

```
// Microsoft Developer Studio generated include file.
// Used by Drum.rc

#define          IDM_FILE_NEW                                    40001
#define          IDM_FILE_OPEN                                   40002
#define          IDM_FILE_SAVE                                   40003
#define          IDM_FILE_SAVE_AS                                40004
#define          IDM_APP_EXIT                                    40005
#define          IDM_SEQUENCE_RUNNING                            40006
#define          IDM_SEQUENCE_STOPPED                            40007
#define          IDM_APP_ABOUT                                   40008
```

当第一次执行 DRUM 时，您将看到在视窗中有两列，左边一列按名称列出了 47 种不同的打击乐器。右边的网格是打击乐器的声音与时间的二维阵列。每一个打击器都对应网格中的一列。32 行就是 32 拍。如果要让这 32 拍出现在一个 4/4 拍的小节中（即每小节 4 个四分音符），那么每 1 拍对应一个三十二分音符。

从「Sequence」功能表选择「Running」时，程式将试图打开 MIDI Mapper 设备。如果失败，萤幕将出现一个讯息方块。否则，您将看到一个「跳动的小球」随演奏的节拍在网格底部跳过。

在网格的任何位置单击滑鼠左键可以在此拍中演奏打击乐器的声音，这时区域将变成暗灰色。用滑鼠右键还可以添加钢琴的拍子，这时区域将会变成亮灰色。如果按下两个键（同时或分别），此区域将变成黑色，而且可以同时听到打击乐器和钢琴的声音。再次单击其中的一个键或双键将关闭该拍中的声音。

网格上部是每 4 拍一个点。这些点使我们不用过多的计算就可以很简易地确定单击的位置。网格的右上角是一个冒号和一条竖线（:|），它们看起来像

传统音乐符号中的反复记号。这个符号表示序列的长度。您可以通过单击滑鼠来将反复记号放置於网格内的任意位置。该序列最多（但不包括）只能演奏反复记号以内的拍子。如果要建立华尔兹节奏，则应将反复记号设定为 3 拍的若干倍。

水平卷动列控制 MIDI Note On 讯息中的速率位元组。这虽然能改变一些合成器的音质，但一般会影响音量。程式起初将速率卷动列设定在中间位置。竖直卷动列控制拍子。这是对数刻度，范围从每拍 1 秒（卷动列在底部）到每拍 10 毫秒（卷动列在顶部）。程式最初将拍子设定为每拍 100 毫秒（1/10 秒），这时卷动列在中间。

「File」功能表允许您储存和读取副档名为.DRM 的档案，这是我定义的一种格式。这些档案很小并采用了 RIFF 的档案格式，这是一种所有新的多媒体资料档案推荐使用的格式。「Help」功能表中的「About」选项显示一个对话方块，该对话方块用一段非常简明的摘要来说明滑鼠在网格中的用法以及两个卷动列的功能。

最後，「Sequence」功能表中的「Stopped」选项用於目前序列结束後终止乐曲并关闭 MIDI Mapper 设备。

多媒体 time 函式

您可能会注意到 DRUM.C 没有呼叫任何多媒体函式。而所有的实际操作都发生在 DRUMTIME 模组中。

虽然普通的 Windows 计时器使用起来很简单，但它对即时时间应用却有灾难性的影响。就像我们在 BACHTOCC 程式中所看到的一样，演奏音乐就是这样的一种即时时间应用，对此 Windows 计时器是不合适的。为了提供在 PC 上演奏 MIDI 所需要的精确度，多媒体 API 还包括一个高解析度的计时器，此计时器通过 7 个字首是 time 的函式实作。这些函式有一个是多余的，而 DRUMTIME 展示了其余 6 个函式的用途。计时器函式将处理执行在一个单独执行绪中的 callback 函式。系统将按照程式指定的计时器延迟时间来呼叫计时器。

处理多媒体计时器时，可以用毫秒指定两种不同的时间。第一个是延迟时间，第二个称为解析度。您可以认为解析度是容错误差。如果指定一个延迟 100 毫秒，而解析度是 10 毫秒，则计时器的实际延迟范围在 90 到 110 毫秒之间。

使用计时器之前，应获得计时器的设备能力：

```
timeGetDevCaps (&timecaps, uSize) ;
```

第一个参数是 TIMECAPS 型态结构的指标，第二个参数是此结构的大小。TIMECAPS 结构只有两个栏位，wPeriodMin 和 wPeriodMax。这是计时器装置驱动程式所支援的最小和最大的解析度值。如果呼叫 timeGetDevCaps 後再查看这些

值，会发现 wPeriodMin 是 1 而 wPeriodMax 是 65535，所以此函式并不是很重要。不过，得到这些解析度值并用於其他计时器函式呼叫是个好主意。

下一步呼叫

```
timeBeginPeriod (uResolution) ;
```

来指出程式所需要的计时器解析度的最低值。该值应在 TIMECAPS 结构所确定的范围之内。此呼叫允许为可能使用计时器的多个程式提供最好的计时器装置驱动程式。呼叫 timeBeginPeriod 及 timeEndPeriod 必须成对出现，我将在後面对 timeEndPeriod 作简短的描述。

现在可以真正设定一个计时器事件：

```
idTimer = timeSetEvent (     uDelay, uResolution, CallBackFunc, dwData, uFlag) ;
```

如果发生错误，从呼叫传回的 idTimer 将是 0。在呼叫的下面，将从 Windows 里用 uDelay 毫秒来呼叫 CallBackFunc 函式，其中允许的误差由 uResolution 指定。uResolution 值必须大於或等於传递给 timeBeginPeriod 的解析度。dwData 是程式定义的资料，後来传递给 CallBackFunc。最後一个参数可以是 TIME_ONESHOT，也可以是 TIME_PERIODIC。前者用於在 uDelay 毫秒数中获得一次 CallBackFunc 呼叫，而後者用於每个 uDelay 毫秒都获得一次 CallBackFunc 呼叫。

要在呼叫 CallBackFunc 之前终止只发生一次的计时器事件，或者暂停周期性的计时器事件，请呼叫

```
timeKillEvent (idTimer) ;
```

呼叫 CallBackFunc 後不必删除只发生一次的计时器事件。在程式中用完计时器以後，请呼叫

```
timeEndPeriod (wResolution) ;
```

其中的参数与传递给 timeBeginPeriod 的相同。

另两个函式的字首是 time。函式

```
dwSysTime = timeGetTime () ;
```

传回从 Windows 第一次启动到现在的系统时间，单位是毫秒。函式

```
timeGetSystemTime (&mmtime, uSize) ;
```

需要一个 MMTIME 结构的指标（与第一个参数一样），以及此结构的大小（与第二个参数一样）。虽然 MMTIME 结构可以在其他环境中用来得到非毫秒格式的系统时间，但此例中它都传回毫秒时间。所以 timeGetSystemTime 是多余的。

Callback 函式只限於它所能做的 Windows 函式呼叫中。Callback 函式可以呼叫 PostMessage，PostMessage 包含有四个计时器函式（timeSetEvent、timeKillEvent、timeGetTime 和多余的 timeGetSystemTime）、两个 MIDI 输出函式（midiOutShortMsg 和 midiOutLongMsg）以及调试函式 OutputDebugStr。

很明显，设计多媒体计时器主要是用於 MIDI 序列而很少用於其他方面。当

然，可以使用 PostMessage 来通知计时器事件的视窗讯息处理程式，而且视窗讯息处理程式可以做任何它想做的事，只是不能回应计时器 callback 自身的准确性。

Callback 函式有五个参数，但只使用了其中两个参数：从 timeSetEvent 传回的计时器 ID 和最初作为参数传递给 timeSetEvent 的 dwData 值。

DRUM.C 模组呼叫 DRUMTIME.C 中的 DrumSetParams 函式有很多次——建立 DRUM 视窗时、使用者在网格上单击或者移动卷动列时、从磁片上载入.DRM 档案时以及清除网格时。DrumSetParams 的唯一的参数是指向 DRUM 型态结构的指标，此结构型态在 DRUMTIME.H 定义。该结构以毫秒为单位储存拍子时间、速度（通常对应於音量）、序列中的拍数以及用於储存网格（为打击乐器和钢琴声设定）的两套 47 个 32 位元组的整数。这些 32 位元整数中的每一位元都对应序列的一拍。DRUM.C 模组将在静态记忆体中维护一个 DRUM 型态的结构，并在呼叫 DrumSetParams 时向它传递一个指标。DrumSetParams 只简单地复制此结构的内容。

要启动序列，DRUM 呼叫 DRUMTIME 中的 DrumBeginSequence 函式。唯一的参数就是视窗代号，其作用是通知。DrumBeginSequence 打开 MIDI Mapper 输出设备，如果成功，则发送 Program Change 讯息来为 MIDI 通道 0 和 9 选择乐器声音（这些通道是基於 0 的，所以 9 实际指的是 MIDI 通道 10，即打击乐器通道。另一个通道用於钢琴声）。DrumBeginSequence 透过呼叫 timeGetDevCaps 和 timeBeginPeriod 来继续工作。在 TIMER_RES 定义的理想计时器解析度通常是 5 毫秒，但我定义了一个称作 minmax 的巨集来计算从 timeGetDevCaps 传回的限制范围以内的解析度。

下一个呼叫是 timeSetEvent，用於确定拍子时间，计算解析度、callback 函式 DrumTimerFunc 以及 TIME_ONESHOT 常数。DRUMTIME 用的是只发生一次的计时器，而不是周期性计时器，所以速度可以随序列的执行而动态变化。timeSetEvent 呼叫之後，计时器装置驱动程式将在延迟时间结束以後呼叫 DrumTimerFunc。

DrumTimerFunccallback 是 DRUMTIME.C 中的函式，在 DRUMTIME.C 中有许多重要的操作。变数 iIndex 储存序列中目前的拍子。Callback 从为目前演奏的声音发送 MIDI Note Off 讯息开始。iIndex 的初始值-1 以防止第一次启动序列时发生这种情况。

接下来，iIndex 递增并将其值连同使用者定义的一个 WM_USER_NOTIFY 讯息一起传递给 DRUM 中的视窗代号。wParam 讯息参数设定为 iIndex，以便在 DRUM.C 中，WndProc 能够移动网格底部的「跳动的小球」。

DrumTimerFunc 将下列事件作为结束：把 Note On 讯息发送给通道 0 和 9 的合成器上，并储存网格值以便下一次可以关闭声音，然後透过呼叫 timeSetEvent 来设定新的只发生一次的计时器事件。

要停止序列，DRUM 呼叫 DrumEndSequence，其中唯一的参数可以设定为 TRUE 或 FALSE。如果是 TRUE，则 DrumEndSequence 按下面的程序立即结束序列：删除所有待决的计时器事件，呼叫 timeEndPeriod，向两个 MIDI 通道发送「all notes off」讯息，然後关闭 MIDI 输出埠。当使用者决定终止程式时，DRUM 用 TRUE 参数呼叫 DrumEndSequence。

然而，当使用者在 DRUM 里的「Sequence」功能表中选择「Stop」时，程式将用 FALSE 作为参数呼叫 DrumEndSequence。这就允许序列在结束之前完成目前的回圈。DrumEndSequence 透过把 bEndSequence 整体变数设定为 NULL 来回应此呼叫。如果 bEndSequence 是 TRUE，并且拍子的索引值设定为 0，则 DrumTimerFunc 把使用者定义的 WM_USER_FINISHED 讯息发送给 WndProc。WndProc 必须通过用 TRUE 作为参数呼叫 DrumEndSequence 来回应该讯息，以便正确地结束计时器和 MIDI 埠的使用。

RIFF 档案 I/O

DRUM 程式也可以储存和检索储存在 DRUM 结构中资讯的档案。这些档案格式都是 RIFF（Resource Interchange File Format：资源交换档案格式），即一般建议使用的多媒体档案型态。当然，您可以用标准档案 I/O 函式来读写 RIFF 档案，但更简便的方法是使用字首是 mmio（对「多媒体输入/输出」）的函式。

检查.WAV 格式时我们发现，RIFF 是标记档案格式，这意味著档案中的资料由不同长度的资料块组成。每个资料块都用一个标记来识别。一个标记就是一个 4 位元组的 ASCII 字串。这与 32 位元整数的标记名称相比要容易些。标记的後面是资料块长度及其资料。因为档案中的资讯不是位於档案开头固定的偏移量而是用标记定义，所以标记档案格式是通用的。这样，可以透过添加附加标记来增强档案格式。在读档案时，程式可以很容易地找到所需要的资料并跳过不需要的或者不理解的标记。

Windows 中的 RIFF 档案由独立的资料块组成。一个资料块可以分为资料块类型、资料块大小以及资料本身。资料块类型是 4 字元的 ASCII 码标记，标记中间不能有空格，但末尾可以有。资料块大小是一个 4 位元组（32 位元）的值，用於显示资料块的大小。资料本身必须占用偶数个位元组，必要时可以在结尾补 0。这样，资料块的每个部分都是从档案开头就字组对齐好了的。资料块大小不包括资料块类型和资料块大小所需要的 8 位元组，并且不反映添加的资料。

对於一些资料块类型，资料块大小与特定档案无关，是相同的。在资料块是包含资讯的固定长度的结构时，就是这种情况。其他情况下，资料块大小根据特定档案变化。

有两个特殊型态的资料块分别称为 RIFF 资料块和 LIST 资料块。其中，资料以一个 4 字元 ASCII 形式型态开始，後面是一个或多个子资料块。LIST 资料块与 RIFF 资料块类似，只是资料以 4 字元的 ASCII 列表型态开始。RIFF 资料块用於所有的 RIFF 档案，而 LIST 资料块只在档案内部用来合并相关子资料块。

一个 RIFF 档案就是一个 RIFF 资料块。因此，RIFF 档案以字串「RIFF」和一个表示档案长度减去 8 位元组的 32 位元值开始。（实际上，如果需要补充资料则档案可能会长一个位元组。）

多媒体 API 包括 16 个字首是 mmio 的函式，这些函式是专门为 RIFF 档案设计的。DRUMFILE.C 中已经用到其中几个函式来读写 DRUM 资料档案。

要用 mmio 函式打开档案，则第一步是呼叫 mmioOpen。函式传回一个档案代号。mmioCreateChunk 函式在档案中建立一个资料块，这使用 MMCKINFO 定义的资料块名称和特徵。mmioWrite 函式写入资料块。写完资料块以後，呼叫 mmioAscend。传递给 mmioAscend 的 MMCKINFO 结构必须与前面通过传递给 mmioCreateChunk 来建立资料块的 MMCKINFO 结构相同。通过从目前档案指标中减去结构的 dwDataOffset 栏位来执行 mmioAscend 函式，此档案指标现在位於资料块的结尾，并且此值储存在资料的前面。如果资料块在长度上不是 2 位元组的倍数，则 mmioAscend 函式也填补资料。

RIFF 档案由巢状组织的资料块套叠组成。为使 mmioAscend 正常工作，必须维护多个 MMCKINFO 结构，每个结构与档案中的一个曾级相联系。DRUM 资料档案共有三级。因此，在 DRUMFILE.C 中的 DrumFileWrite 函式中，我为三个 MMCKINFO 结构定义了一个阵列，可以分别标记为 mmckinfo[0]、mmckinfo[1]和 mmckinfo[2]。在第一次 mmioCreateChunk 呼叫中，mmckinfo[0]结构与 DRUM 形式型态一起用於建立 RIFF 型态的块。其後是第二次 mmioCreateChunk 呼叫，它用 mmckinfo[1]与 INFO 列表型态一起建立 LIST 型态的资料块。

第三次 mmioCreateChunk 呼叫用 mmckinfo[2]建立一个 ISFT 型态的资料块，此资料块用於识别建立资料档案的软体。下面的 mmioWrite 呼叫用於写字串 szSoftware，呼叫 mmioAscent 可用 mmckinfo[2]来填充此资料块的资料块大小栏位。这是第一个完整的资料块。下一个资料块也在 LIST 资料块内。程式继续用另一个 mmioCreateChunk 来呼叫建立 ISCD（creation data：建立资料）资料块，并再次使用 mmckinfo[2]。在 mmioWrite 呼叫来写入资料块以後，使用 mmckinfo[2]呼叫 mmioAscend 来填充资料块大小。现在写到了此资料块的结尾，

也是 LIST 块的结尾。所以，要填充 LIST 资料块的资料块大小栏位，可再次呼叫 mmioAscend，这次使用 mmckinfo[1]，它最初用於建立 LIST 资料块。

要建立「fmt」和「data」资料块，mmioCreateChunk 使用 mmckinfo[1]；mmioWrite 呼叫的後面也使用 mmckinfo[1]的 mmioAscend。在这一点上，除了 RIFF 资料块本身以外,所有的资料块大小都填好了。这需要多次使用 mmckinfo[0]来呼叫 mmioAscend。虽然有多次呼叫，但只呼叫 mmioClose 一次。

看起来好像 mmioAscend 呼叫改变了目前的档案指标，而且它的确填充了资料块大小，但在函式传回时，在资料块结束（或可能因补充资料而增加 1 位元组）以後，档案指标恢复到以前的位置。从应用的观点来看，所有的档案写入都是按从头到尾的顺序。

mmioOpen 呼叫成功後，除了磁碟空间耗尽之外，不会发生其他错误。使用变数 wError 从 mmioCreateChunk、mmioWrite、mmioAscend 和 mmioClose 呼叫累计错误代码，如果磁碟空间不足则每个呼叫都会失败。如果发生了错误，则 mmioOpen 以 MMIO_DELETE 常数为参数来删除档案，并传回错误资讯。

读 RIFF 档案与建立 RIFF 档案类似，只不过是呼叫 mmioRead 而不是 mmioWrite，呼叫 mmioDescend 而不是 mmioCreateChunk。「下降」（descend）到一个资料块，是指找到资料块位置，并把档案指标移动到资料块大小之後（或者在 RIFF 或 LIST 资料块类型的形式型态或者列表型态的後面）。从资料块「上升」指的是把档案指标移动到资料块的结尾。mmioDescend 和 mmioAscend 函式都不能把档案指标移到档案的前一个位置。

DRUM 以前的版本在 1992 年的《PC Magazine》发表。那时，Windows 支援两个不同等级的 MIDI 合成器（称为「基本的」和「扩展的」）。那个程式写的档案有格式识别字 1。本章的 DRUM 程式将格式识别字设定为 2。不过，它可以读取并转换早期的格式。这在 DrumFileRead 常式中完成。

第二十三章　领略 Internet

Internet——全世界电脑透过不同协定交换资讯的大型连结体——近几年重新定义了个人计算的几个领域。虽然拨接资讯服务和电子邮件系统在 Internet 流行开来之前就已经存在，但它们通常局限於文字模式，并且根本没有连结而是各自分隔的。例如，每一种资讯服务都需要拨不同的电话号码，用不同的使用者 ID 和密码登录。每一种电子邮件系统仅允许在特定系统的缴款使用者之间发送和接收邮件。

现在，往往只需要拨单一支电话就可以连结整个 Internet，而且可以和有电子邮件位址的人进行全球通信。特别是在 World Wide Web 上，超文字、图形和多媒体（包括声音、音乐和视讯）的使用已经扩展了线上资讯的范围和功能。

如果要提供涵盖 Windows 中所有与 Internet 相关程式设计问题的彻底介绍，可能还需要再加上几本书才够。所以，本章实际上主要集中在如何让小型的 Microsoft Windows 应用程式能够有效地从 Internet 上取得资讯的两个领域。这两个领域分别是 Windows Sockets (Winsock) API 和 Windows Internet（WinInet）API 支援的档案传输协定（FTP：File Transfer Protocol）的部分。

Windows Sockets

Socket 是由 University of California 在 Berkeley 分校开发的概念，用於在 UNIX 作业系统上添加网路通讯支援。那里开发的 API 现在称为「Berkeley socket interface」。

Sockets 和 TCP/IP

Socket 通常（但不专用於）与主宰 Internet 通信的传输控制项协定/网际网路协定（TCP/IP：Transmission Control Protocol/Internet Protocol）牵连在一起。网际网路协定（IP：Internet Protocol），作为 TCP/IP 的组成部分之一，用来将资料打包成「资料封包（datagram）」，该资料封包包含用於标识资料来源和目的地的表头资讯。而传输控制协定（TCP：Transmission Control Protocol）则提供了可靠的传输和检查 IP 资料封包正确性的方法。

在 TCP/IP 下，通讯端点由 IP 位址和埠号定义。IP 位址包括 4 个位元组，用於确定 Internet 上的伺服器。IP 位址通常按「由点连结的四个小於 255 的数字」的格式显示，例如「209.86.105.231」。埠号确定了特定的服务或伺服器提供的服务。其中一些埠号已经标准化，以提供众所周知的服务。