实验二　　视频捕捉程序的设计

一、实验目的

1、熟悉和掌握使用 Windows API 进行编程的基本原理和方法。
2、熟悉各种不同的视频文件格式。

二、实验内容

利用 windows api 或 Java media Framework（JMF）等实现简单的视频捕捉功能。
更高要求：实现连续捕捉静止图像。

三、实验环境

硬件：摄像头
软件：摄像头驱动程序/JMF

四、实验步骤

1、创建一个基于对话框的工程，如 VideoPlay。
2、在对话框上添加相应的按钮，实现如动态捕捉、播放、连续捕捉等项功能。可考虑添加滑动条（slider 控件）用来反映目前的播放位置，等等。
3、为各个按钮添加相应的函数。
4、调试，运行。

五、参考资料

（1）补充资料：VFP 编程知识介绍；
（2）MSDN 联机帮助；
（3）网络上相关介绍；

补充资料　　VFW 编程知识介绍

目录：

（一） Video for Windows

Video for Windows 是一套基于 Windows 的视频软件工作平台，能在声卡和视频卡的支持下获取和编辑数字音频及与音频同步动态的视频，并能把音/视频同步数据存储成 AVI 文件。

在硬件和与其相联系的驱动程序的支持下，Video for Windows 可从录像机、摄像机、视盘机或其他的动画形式中捕获、播放和编辑达 30 帧每秒的全动视频。其主要特征如下：

● 能有效地把数字视频捕获到硬盘或者从硬盘或 CD-ROM 中播放数字视频。

● 在只有有限的内存的条件下，能在计算机中有效、快速地装载并播放数字视频。Video for Windows 在播放时不需把较长的数字视频预先装入内存。在一定时间内，它只需先装入几帧视频数据及相应的音频数据。

● 能够压缩视频数据，以减少存储和传输的信息量，并能在保证视频质量的情况下减少数据量。

Video for Windows 的基本工作流程如图 2-1 所示。可安装的压缩管理器（Installable Compressor Manager）把不同的压缩/解压缩编码驱动程序（Codec Driver）和数字视频应用程序（数字视频捕获/编辑程序）隔离并作为桥梁把它们联系起来。所有的待压缩或解压缩的数字视频数据都经可安装的压缩管理器在应用程序和压缩/解压缩编码驱动程序之间传送。这样，可以使数字视频应用程序与压缩/解压缩编码驱动程序无关。只要提供标准的可安装的压缩管理器接口，各种不同的压缩/解压缩编码方法都能方便地通过可安装的压缩管理器被应用程序使用。

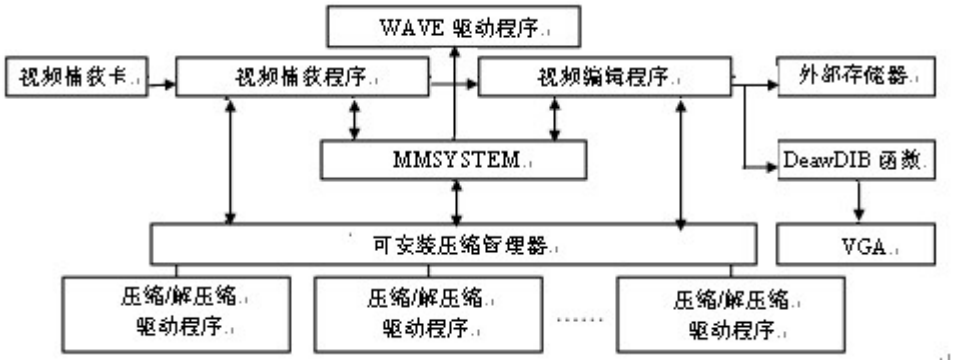

图 2-1 Video for Windows 的基本工作流程

Video for Windows 提供的压缩/解压缩编码驱动程序主要有 4 种，即 Microsoft RLE，Microsoft Video 1，Intel Indeo R3 和 Cinepak。它们之间的性能对比如表 2-1 所示。

表 2-1 4 种压缩/解压缩编码驱动程序的比较

驱动程序	压缩时间	视频质量	压缩比
Microsoft RLE8	快	差 160×120/12 帧每秒	低
Microsoft Video 1	快	好 160×120/12 帧每秒	中等
Intel Indeo R3	中等	中等 320×240/15 帧每秒	高
Cinepak	慢	好 320×240/15 帧每秒	高

（二） Visual C++与 MCIWnd 窗口类

Microsoft 公司为开发 Video for Windows 产品发布了开发工具，即 Video for Windows Developer Kit。其中包含了 MCIWnd 窗口类编程和 AVI 音/视频文件操作函数。

在 Windows 中，Video for Windows 不再是一个独立的产品，而是操作系统组成的一部分。因此，Microsoft 已将 Video for Windows Developer Kit 包含在 Visual C++中。也就是说，利用 Visual C++可以进行 MCIWnd 窗口类和 AVI 音/视频文件操作编程。

MCIWnd 窗口类定义了一种特殊的窗口类别（Windows Class，不是 C++的类），可用来控制 MCI 设备和文件的操作。在使用时，可以显示需要的控制界面，如窗口、带控制按钮的工具条、菜单及位置滑竿等。用户可利用这些界面实现对 MCI 的操作，如播放、停止、定位等。也可以不显示任何界面，仅使用相关的操作控制。

AVI 音/视频文件操作包含在一组 AVIFile 函数中，提供具有时间特性的文件操作，包括对 WAVE 文件和 AVI 文件的读写操作。

使用 MCIWnd 窗口类和 AVI 音/视频文件操作的程序项目必须在源文件中包含头文件 vfw.h，在 Project Settings\Link\Object/library modules 中加入 vfw32.lib 输入库。

（三）使用 AVIFile 函数

WAVE 和 AVI 文件都是 RIFF 格式的媒体文件，它允许程序以灵活的可扩展方式存储数据块，并使复杂数据被构造成能被快速访问和没有内存局限的形式。但处理 RIFF 文件需对数据块做较复杂的处理，使用起来很复杂。使用 AVIFile 函数能大大简化 RIFF 文件的读写操作。

AVIFile 函数将 RIFF 文件看成由一个或多个数据流所组成。它定义的数据流类型如表 2-2 所示。进行 AVIFile 操作主要涉及 11 个常用的 AVIFile 函数，如表 2-3 所示。

表 2-2 AVIFile 文件中的数据流类型

类型	标志	识别码
视频	streamtypeVIDEO	Vids
音频	streamtypeAUDIO	Auds
MIDI	streamtypeMIDI	Mids
文字	streamtypeTEXT	Txts

表 2-3 主要的 AVIFile 函数

函数	功能
void AVIFileInit()	初始化 AVIFile 操作库
LONG AVIFileOpen(PAVIFILE *ppFile, LPCSTR szFile,	打开文件获取文件指针

UINT uMode, CLSID * pclsidHandler);	PAVIFILE *ppFile
LONG AVIFileInfo(PAVIFILE pFile, AVIFILEINFO *pfi, LONG lSize);	获取文件信息结构指针 AVIFILEINFO *pfi
LONG AVIFileGetStream(PAVIFILE pFile, PAVISTREAM *ppAVI, DWORD fccType, LONG lParam);	获取数据流指针 PAVISTREAM *ppAVI
LONG AVIFileStreamInfo(PAVISTREAM pavi, AVISTREAMINFO *psi, LONG lSize);	读取数据流信息结构指针 AVISTREAMINFO *psi
LONG AVIFileStreamReadFormat(PAVISTREAM pavi, LONG lPos, LPVOID lpFormat, LONG *lpcbFormat);	读取数据流格式 lpFormat
LONG AVIFileStreamRead(PAVISTREAM pavi, LONG lStart, LONG lSamples, LPVOID lpBuffer, LONG cbBuffer, LONG *plBytes, LONG *plSamples);	读取数据流数据 lpBuffer
LONG AVIFileStreamSetFormat(PAVISTREAM pavi, LONG lPos, LPVOID lpFormat, LONG cdFormat);	设置数据流格式为 lpFormat
LONG AVIFileCreateStream(PAVIFILE pFile, PAVISTREAM *ppavi, AVISTREAMINFO *psi);	创建数据流 PAVISTREAM *ppavi，并写入数据 AVISTREAMINFO *psi
LONG AVIFileRelease(PAVIFILE pFile);	关闭文件 PAVIFILE pFile
void AVIFileExit();	释放 AVIFile 操作资源

应按一定的次序和流程来使用 AVIFile 函数，其基本流程如下：
1）调用 VIFileInit 函数，初始化函数相应的动态链接库；
2）调用 AVIFileOpen 函数，打开文件，取得文件指针；
3）调用 AVIFileInfo 函数，获取文件中数据流数量和种类等文件信息；
4）调用 AVIFileGetStream 函数，打开数据流；
5）调用 AVIFileStreamInfo 函数，获取数据流有关信息；
6）调用 AVIFileStreamReadFormat，AVIFileStreamRead，AVIFileStreamSetFormat 及 AVIFileCreateStream 等函数，读写文件；
7）调用 AVIFileRelease 函数，关闭文件；
8）重复第 2）～第 7）步，进行不同文件的读写操作；
9）不再进行文件操作时，调用 AVIFileExit，结束 AVIFile 文件操作，释放进行 AVIFile 文件操作所需占用的资源。

下面的函数将读出给定文件的信息和数据流信息指针。

```
//////////////////////////////////////////////////////////////////////
// GetFileInfo
//
// 读出给定文件的信息和数据流信息指针
//
// 参数：szFile —— 传入的文件名
//       pFileInfo —— 传出的文件信息结构指针
//       ppStream —— 传出的指向数据流结构指针的指针
//
// 返回值：TRUE 表示操作成功
//         FALSE 表示操作失败
//////////////////////////////////////////////////////////////////////
BOOL GetFileInfo(LPCSTR szFile,
```

```
    AVIFILEINFO *pFileInfo,
    PAVISTREAM *ppStream)
{
    PAVIFILE pAvi;

    // 打开文件
    if (0 != AVIFileOpen(&pAvi, szFile, OF_READ, NULL))
        return FALSE;  // 打开文件出错

    // 读取文件信息
    if (0 != AVIFileInfo(pAvi, pFileInfo, sizeof(AVIFILEINFO)))
        return FALSE;  // 读文件信息出错

    // 获取数据流信息
    for (int i=0; i<pFileInfo->dwStreams; ++I)
    {
        AVISTREAMINFO info;

        // 打开第 i 个数据流
        if (0 != AVIFileGetStream(pAvi,&ppStream[i],0,i))
            return FALSE; // 打开数据流出错

        // 读取第 i 个数据流信息
        if (0 != AVIStreamInfo(ppStream[i],&info,sizeof(AVISTREAMINFO)))
            return FALSE; // 读数据流出错
    }
}
```

（四） 使用 MCIWnd 窗口类

使用 MCIWnd 的第一步就是创建 MCIWnd 类窗口，这可通过调用 MCICreateWnd 函数来实现，其函数原型如下：

```
HWND MCIWndCreate(
HWND hWndParent,    // MCIWnd 窗口的父窗口句柄
HINST hInstance,    // 应用程序实例句柄
DWORD dwStyle, // 窗口风格
LPSTR szFile    // 相关的 MCI 文件名
);
```

如果该函数调用成功，将返回所创建窗口的句柄，以后大多数的 MCI 操作函数都必须以该句柄为第一个参数。

1）hWndParent 参数说明了 MCIWnd 窗口的父窗口，如果其 dwStyle 中包含 WS_CHILD 风格，则该参数不能为 NULL。

2)hInstance 指明了使用 MCIWnd 窗口的应用程序实例,可通过调用 AfxGetInstance- Handle() 函数获得。

3）dwStyle 声明 MCIWnd 窗口的风格，可供使用的风格包括两部分，如下所述。

● WS_类窗口风格：MCIWndCreate 函数最终使用 Win 32 SDK 的 CreateWindow 函数来创建窗口，所以 MCIWndCreate 的 dwStyle 参数中可以包括任何的 WS_类窗口风格。如果 dwStyle

中没有指定任何的 WS_类窗口风格，则当 hWndParent 为 NULL 时，函数将自动使用 WS_OVERLAPPED|WS_VISIBLE 风格；当 hWndParent 不为 NULL 时，函数将自动使用 WS_CHILD|WS_BORDER|WS_VISIBLE 风格。

● MCIWnd 定义的窗口风格（如表 2-4 所示）：这些风格可用符号“|”组合起来使用。如果未指定任何的这类风格，函数将按默认设置使用所有的 MCIWnd 窗口控件。

提示：MCIWnd 窗口的风格也可在窗口创建之后，调用函数 MCIWndChangeStyles 进行修改。

4）szFile 指与 MCIWnd 窗口对应的媒体设备名或文件名。例如，szFile 为“cdaudio”时将创建播放 CD 音频的窗口；szFile 为“tada.wav”时将创建播放数字化波形声音“tada.wav”的窗口；szFile 为“dancing.avi”时将创建播放数字视频“dancing.avi”的窗口。

提示：szFile 可设为 NULL。在窗口创建之后调用 MCIWndOpenDialog 函数将出现一个文件选取对话框，可为窗口选取新的对应文件。

表 2-4 MCIWnd 窗口专用风格

风　　格	意　　义
MCIWND_NOAUTOSIZEWINDOW	当视频大小改变时，窗口大小不变
MCIWND_NOPLAYBAR	不在窗口中显示工具条
MCIWND_NOAUTOSIZEMOVIE	当窗口大小改变时，不自动改变视频大小
MCIWND_NOMENU	当点击鼠标右键时，不弹出上下文菜单

风　　格	意　　义
MCIWND_SHOWNAME	在窗口标题上显示文件名
MCIWND_SHOWPOS	在窗口标题上显示当前播放位置
MCIWND_SHOWMODE	在窗口标题上显示目前状态
MCIWND_SHOWALL	在窗口标题上显示以上 3 项内容
MCIWND_NOTIFYMODE	当播放状态改变时，向父窗口发送消息
MCIWND_NOTIFYPOS	当播放位置改变时，向父窗口发送消息
MCIWND_NOTIFYSIZE	当播放视频大小改变时，向父窗口发送消息
MCIWND_NOTIFYMDIA	当播放媒体文件改变时，向父窗口发送消息
MCIWND_NOTIFYERROR	当发生错误时，向父窗口发送消息
MCIWND_NOTIFYALL	以上 5 项中任一项发生时，向父窗口发送消息
MCIWND_RECORD	在工具条中显示录制键
MCIWND_NOERRORDLG	当出现错误时，不显示错误提示对话框
MCIWNDF_NOOPEN	在工具条菜单按钮中不显示打开和关闭菜单项

如果创建了具有默认风格的 MCIWnd 窗口，窗口中将具备许多的控制，如图 2-2 所示。播放/停止按钮能控制媒体的播放/停止操作；位置滑竿不仅能显示当前媒体播放的位置，而且可拖曳滑竿控制块直接设置媒体的位置；点击工具条中的菜单按钮和在窗口用户区中点击鼠标右键都将出现图中所示的菜单；菜单中“命令”菜单项可用于直接发送 MCI 命令。

图 2-2 默认的带完整控制的 MCIWnd 窗口

如果想用自己的界面来控制媒体操作，就必须使用 MCIWnd 窗口类提供的一系列操作宏（这些宏的形式与使用和 API 函数完全一样）。它们都需要 MCIWnd 窗口的句柄作为参数，有的还需其他的附加参数。这些宏的数量很多，但其中许多宏很少用到，表 2-5 中列出了常用的一些宏。

表 2-5 常用的 MCIWnd 窗口类宏

宏	功 能	分 类
MCIWndHome(hwnd)	操作位置移到媒体的开头	媒体控制宏
MCIWndPause(hwnd)	暂停播放	
MCIWndPlay(hwnd)	从当前开始播放	
MCIWndPlayFrom(hwnd, lPos)	从 lPos 位置播放	
MCIWndPlayTo(hwnd, lEnd)	从当前播放到 lEnd	
MCIWndPlayFromTo(hwnd,lStart, lEnd)	从 lStart 播放到 lEnd	
MCIWndResume(hwnd)	暂停后重新播放	
MCIWndSeek(hwnd, lPos)	寻找到 lPos 位置处	
MCIWndSetVolume(hwnd, iVolume)	设置音量为 iVolume	
MCIWndStop(hwnd)	停止播放	
MCIWndCanPlay(hwnd)	返回媒体是否可播放	获取信息宏
MCIWndCanEject(hwnd)	返回媒体是否可弹出	
MCIWndGetEnd(hwnd)	返回媒体结束的位置	
MCIWndGetLength(hwnd)	返回媒体的长度	
MCIWndGetPosition(hwnd)	返回媒体的当前位置	
MCIWndGetVolume(hwnd)	返回媒体的当前音量	
MCIWndClose(hwnd)	关闭 MCI 设备	辅助功能宏
MCIWndDestroy(hwnd)	关闭 MCIWnd 类窗口	
MCIWndSendString(hwnd, sz)	给 MCI 设备发送 MCI 命令字符串	

此外，还有 3 个函数在进行文件选择时很有用，它们是 GetOpenFileNamePreview，GetSaveFileNamePreview 和 AVIBuildFilter。前两个函数的用法与 Win 32 SDK 中的 GetOpenFileName 和 GetSaveFileName 的用法完全一样，但在它们对话框的右侧多了一个预

览窗口，可以对选定的文件做预览操作。AVIBuildFilter 可以产生 AVIFile 支持的 MCI 文件的字符串过滤器，通常与 GetOpenFileNamePreview，GetSaveFileNamePreview 或 GetOpenFile- Name，GetSaveFileName 结合使用。

（五）MCIWnd 窗口类编程

MCIWnd 窗口类编程支持波形音频、MIDI、CD 音频和数字视频 AVI 的操作，只需简单地在调用 MCIWndCreate 函数时，设置其最后一个参数为相应的文件或媒体设备类型（对 CD 音频而言）即可。设置 MCIWnd 窗口的不同风格，可以在窗口直接显示和控制对媒体操作的工具条；也可以不显示操作工具条，而调用相应的宏来对媒体进行操作。

1、音频编程

播放波形音频的示例如下所示：

```
//利用工具条直接播放
HWND hMciWnd = MCIWndCreate(NULL, AfxGetInstanceHandle(),
    WS_OVERLAPPED | MCIWNDF_NOOPEN,
     “Sample.wav”);

//不显示工具条，利用宏完整播放
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
     “Sample.wav”);
MCIWndPlay(hMciWnd);

//不显示工具条，利用宏部分播放
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
     “Sample.wav”);
MCIWndPlayFromTo (hMciWnd, 1000, 3000);

//不显示工具条，利用宏播放，播放时，不断通知父窗口当前的播放位置
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|
    MCIWNDF_NOPLAYBAR| MCIWNDF_NOTIFYPOS
     “Sample.wav”);
MCIWndPlayFromTo (hMciWnd, 1000, 3000);

……

// 响应 MCIWNDM_NOTIFYPOS 消息
void CMyWnd::OnMciWndNotifyPos(WPARAM wParam, LPARAM lParam)
{
```

```
    ……
}
```

录制示例如下所示：

```
//利用工具条直接播放
HWND hMciWnd = MCIWndCreate(NULL, AfxGetInstanceHandle(),
    WS_OVERLAPPED | MCIWNDF_NOOPEN| MCIWNDF_RECORD,
    “Sample.wav”);
//不显示工具条，利用宏录音，存为指定的文件名
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
    “Sample.wav”);
MCIWndRecord(hMciWnd);
MCIWndSave (hMciWnd, “NewSample.wav”);

//不显示工具条，利用宏录音，让用户选择文件名
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
    “Sample.wav”);
MCIWndRecord(hMciWnd);
MCIWndSaveDialog(hMciWnd);
```

获取波形文件信息示例如下所示：

```
//获取长度
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
    “Sample.wav”);
LONG lLength = MCIWndGetLength(hMciWnd);
CString s;
s.Format(“Sample.wav 的长度是%ld 毫秒”, lLength);
AfxMessageBox(s);

//获取波形文件当前状态
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
    “Sample.wav”);
char szMode[256];
MCIWndGetMode(hMciWnd, szMode, 256);
CString s = “Sample.wav 的当前状态是”;
s += szMode;
AfxMessageBox(s);

//获取当前播放位置
```

```
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
    “Sample.wav”);
LONG lLength = MCIWndGetPosition (hMciWnd);
CString s;
s.Format(“Sample.wav 的当前播放位置是%ld 毫秒”, lLength);
AfxMessageBox(s);
```

播放 MIDI 的示例如下所示：

```
//利用工具条直接播放
HWND hMciWnd = MCIWndCreate(NULL, AfxGetInstanceHandle(),
    WS_OVERLAPPED | MCIWNDF_NOOPEN,
    “Sample.mid”);

//不显示工具条，利用宏完整播放
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
           WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
        “Sample.mid”);
MCIWndPlay(hMciWnd);

//不显示工具条，利用宏部分播放
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
    “Sample.mid”);
MCIWndPlayFromTo (hMciWnd, 1000, 3000);

//不显示工具条，利用宏播放，播放时，不断通知父窗口当前的播放位置
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|
    MCIWNDF_NOPLAYBAR| MCIWNDF_NOTIFYPOS
    “Sample.mid”);
MCIWndPlayFromTo (hMciWnd, 1000, 3000);

……

// 响应 MCIWNDM_NOTIFYPOS 消息
void CMyWnd::OnMciWndNotifyPos(WPARAM wParam, LPARAM lParam)
{
    ……
}
```

播放 CD 音频的示例如下所示：

```
//利用工具条直接播放
HWND hMciWnd = MCIWndCreate(NULL, AfxGetInstanceHandle(),
    WS_OVERLAPPED | MCIWNDF_NOOPEN,
     “cdaudio”);

//不显示工具条，利用宏完整播放
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
     “cdaudio”);
MCIWndPlay(hMciWnd);

//不显示工具条，利用宏播放，播放时，不断通知父窗口当前的播放位置
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
    WS_CHILD|WS_VISIBLE|
    MCIWNDF_NOPLAYBAR| MCIWNDF_NOTIFYPOS
     “cdaudio”);
MCIWndPlay(hMciWnd);

……

// 响应 MCIWNDM_NOTIFYPOS 消息
void CMyWnd::OnMciWndNotifyPos(WPARAM wParam, LPARAM lParam)
{
    ……
}
```

2、视频编程

下面的示例演示了用 MCIWnd 播放 AVI 文件的方法：

利用工具条直接播放：

```
HWND hMciWnd = MCIWndCreate(NULL, AfxGetInstanceHandle(),
              WS_OVERLAPPED | MCIWNDF_NOOPEN,
                     “Sample.avi”);
                   不显示工具条，利用宏完整播放：
        // m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
         WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
                 “Sample. avi”);
MCIWndPlay(hMciWnd);
                   不显示工具条，利用宏部分播放：
        // m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
```

```
        WS_CHILD|WS_VISIBLE|MCIWNDF_NOPLAYBAR,
                “Sample. avi”);
MCIWndPlayFromTo (hMciWnd, 1000, 3000);
```

不显示工具条，利用宏播放，播放时，不断通知父窗口当前的播放位置：

```
// m_hWnd 为使用本代码段的窗口对象的句柄
HWND hMciWnd = MCIWndCreate(m_hWnd, AfxGetInstanceHandle(),
WS_CHILD|WS_VISIBLE|
MCIWNDF_NOPLAYBAR| MCIWNDF_NOTIFYPOS
                “Sample. avi”);
MCIWndPlayFromTo (hMciWnd, 1000, 3000);

        ……

// 响应 MCIWNDM_NOTIFYPOS 消息
void CMyWnd::OnMciWndNotifyPos(WPARAM wParam, LPARAM lParam)
{
    ……
}
```

（六） 示例：基于 MCIWnd 窗口类的媒体播放器

MCIWnd 窗口类函数的功能非常完善，使用也很简单。本节将利用它们来实现另一个媒体播放器，其步骤如下所述。

1）用 AppWizard 生成项目框架。

在 Visual Studio 中选取 File→New→Projects 命令，在出现的 New Project 对话框中选择创建 Visual C++ Projects\Win32 Projects 目录下的 MFC Application，并输入项目名为 MCIWndPlayer。设定应用程序类型为基于对话框（Dialog based），资源语种为简体中文。其余选项按默认值创建项目和源文件。

AppWizard 将创建如下所述的主要 MCIPlayer 框架类和资源：

- 应用程序类 CMCIWndPlayerApp，父类为 CWinApp，源文件为 MCIWndPlayer.h 和 MCIWndPlayer.cpp。
- 主框窗口类 CMCIWndPlayerDlg，父类为 CDialog，源文件为 MCIPlayerWndDlg.h 和 MCIPlayeWndrDlg.cpp。
- 对话框模板 IDD_ABOUTBOX 和 IDD_MCIWndPLAYER_DIALOG。
- 图标 IDR_MAINFRAME。
- 字符串表。

2）用资源编辑器修改接口资源。

- 编辑修改对话框模板 IDD_MCIPLAYER_DIALOG，先删除所有的默认控制，然后居中增加一个静态文本控制，并设定其 ID 为 IDC_STATIC_NOMEDIA，文本为“没有打开媒体文件”。
- 编辑图标 IDR_MAINFRAME 为一音响设备形状。
- 增加一个 ID，IDM_OPEN。
- 修改或增加字符串表中的字符串如下所示：

```
IDS_ABOUTBOX  “关于 MCIWnd 媒体播放器...”
IDS_TITLE  “MCIWnd 媒体播放器”
IDS_OPEN   “打开 MCI 媒体文件(&O)...”
```

IDS_OPENTITLE“打开 MCI 媒体文件”

3）修改 OnInitDialog 函数进行程序初始化。

在 AppWizard 生成的 OnInitDialog 函数模板中增加代码，以在对话框的系统菜单中增加如下两个菜单项：

IDM_ABOUT 关于 MCIWnd 媒体播放器...

IDM_OPEN 打开 MCI 媒体文件(&O)...

OnInitDialog 函数的代码如下：

```
BOOL CMCIWndPlayerDlg::OnInitDialog()
{
        CDialog::OnInitDialog();

        // Add "About..." menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);
if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
        pSysMenu->AppendMenu(MF_SEPARATOR);
        pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
            strAboutMenu);
        }
        // 增加“打开 MCI 媒体文件(&O)...”菜单项
        CString strOpen;
        strOpen.LoadString(IDS_OPEN);
        if (!strOpen.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_OPEN,
        strOpen);
        }
    }

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);    // Set big icon
    SetIcon(m_hIcon, FALSE);   // Set small icon

    return TRUE;  // return TRUE  unless you set the focus to a control
    }
```

4）在 OnSysCommand 函数中增加处理菜单项 IDM_OPEN 和处理关闭对话框的代码。

```
void CMCIWndPlayerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
if ((nID & 0xFFF0) == IDM_ABOUTBOX)
{
    CAboutDlg dlgAbout;
    dlgAbout.DoModal();
}
else if (nID == IDM_OPEN)
    DoOpen();
else if (nID == SC_CLOSE)
EndDialog(0);
else
{
    CDialog::OnSysCommand(nID, lParam);
}
}
```

5）增加函数 DoOpen()的声明和实现。

在 MCIWndPlayerDlg.h 中的 CMCIWndPlayerDlg 类声明中增加该函数的声明。在 MCIWndPlayerDlg.cpp 中的 CMCIWndPlayerDlg 类实现中增加该函数的实现。

该函数调用可进行预览的 GetOpenFileNamePreview 函数以选取媒体文件；调用 MCIWndCreate 以创建 MCIWnd 窗口，设置窗口风格为 WS_CHILD|WS_VISIBLE|MCIWNDF_NOOPEN|MCIWNDF_NOERRORDLG |MCIWNDF_NOTIFYSIZE；创建的 MCIWnd 窗口将作为 CMCIWndPlayerDlg 对象的子窗，它带有工具条，在工具条菜单按钮菜单中不显示“打开”和“关闭”菜单项，MCI 操作出错时不显示错误信息，当 MCIWnd 窗口大小改变时，通知父窗。

如 MCIWnd 窗口创建成功，则删除窗口中预定义的静态文本，并将打开的 MCI 媒体文件名加到标题中。实现的代码如下所示。

```
void CMCIWndPlayerDlg::DoOpen()
{
    // 选取 MCI 媒体文件
    char szFile[256];
    char szFilter[] = "All multimedia Files\0*.avi;*.wav;*.mid;*.au\ 0Microsoft
    AVI Files\0*.avi\0Waveform Audio Files\0*.wav\0MIDI Files\0*.mid\ 0All
    Files\0*.*\0\0";

    HINSTANCE hInst = AfxGetInstanceHandle();
    OPENFILENAME ofn;
    CString strTitle;
    strTitle.LoadString(IDS_OPENTITLE);

    szFile[0] = 0;
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.hwndOwner = m_hWnd;
```

```
ofn.hInstance = hInst;
ofn.lpstrTitle = strTitle;
ofn.lpstrFilter = szFilter;
ofn.lpstrCustomFilter = NULL;
ofn.nMaxCustFilter = 0;
ofn.nFilterIndex = 0;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = NULL;
ofn.nMaxFileTitle = 0;
ofn.lpstrInitialDir = NULL;
ofn.Flags = OFN_FILEMUSTEXIST | OFN_PATHMUSTEXIST |
                OFN_HIDEREADONLY;
ofn.nFileOffset = 0;
ofn.nFileExtension = 0;
ofn.lpstrDefExt = NULL;
ofn.lCustData = 0;
ofn.lpfnHook = NULL;
ofn.lpTemplateName = NULL;

if (GetOpenFileNamePreview(&ofn))
{
    // 撤销原有的 MCIWnd 窗口
    if (m_hMciWnd != NULL)
    MCIWndDestroy(m_hMciWnd);
    else
    {
        // 删除预定义静态文本
        CWnd* pWnd = GetDlgItem(IDC_STATIC_NOMEDIA);
        if (::IsWindow(pWnd->m_hWnd))
        pWnd->DestroyWindow();
    }

    // 创建 MCIWnd
    m_hMciWnd = MCIWndCreate(m_hWnd, hInst,
    WS_CHILD |WS_VISIBLE | MCIWNDF_NOOPEN |
    MCIWNDF_NOERRORDLG | MCIWNDF_NOTIFYSIZE,
    szFile);

    // 如创建成功
    if (m_hMciWnd != NULL)
    {
// 设置新标题
CString s;
s.LoadString(IDS_TITLE);
CString strTitle = szFile;
strTitle += " - ";
```

```
    strTitle += s;
    SetWindowText(strTitle);

    //  根据 MCIWnd 的大小修改对话框的大小
    CRect rc;
    ::GetWindowRect(m_hMciWnd, &rc);
    AdjustWindowSize(rc);
            }
        }
    }
```

函数 AdjustWindowSize 通过调用 SetWindowPos 来完成修改对话框大小的任务。

```
    BOOL CMCIWndPlayerDlg::AdjustWindowSize(CRect rcMciWnd)
    {
        ::AdjustWindowRect(&rcMciWnd,
            ::GetWindowLong(m_hWnd, GWL_STYLE),
            FALSE);

        return SetWindowPos(NULL, 0, 0,
            rcMciWnd.Width(), rcMciWnd.Height(),
            SWP_NOZORDER|SWP_NOMOVE|SWP_NOACTIVATE);
    }
```

6）响应 MCIWnd 改变大小的通知消息 MCIWNDM_NOTIFYSIZE。

函数 OnMci- WndNotify Size 用来响应该消息，修改消息映射：

```
    BEGIN_MESSAGE_MAP(CMCIWndPlayerDlg, CDialog)
    //{{AFX_MSG_MAP(CMCIWndPlayerDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_MESSAGE(MCIWNDM_NOTIFYSIZE, OnMciWndNotifySize)
    //}}AFX_MSG_MAP
    END_MESSAGE_MAP()
```

在 MCIWndPlayerDlg.h 中的 CMCIWndPlayerDlg 类声明中增加 OnMciWndNotifySize 函数的声明如下：

```
afx_msg void OnMciWndNotifySize(WPARAM wParam, LPARAM lParam);
```

该函数根据 MCIWnd 大小的变化，调用 AdjustWindowSize 函数来矫正对话框的大小，其代码如下：

```
    void CMCIWndPlayerDlg::OnMciWndNotifySize(WPARAM wParam,
                LPARAM lParam)
    {
        if (m_hMciWnd != NULL)
```

```
    {
        CRect rc;
        ::GetWindowRect(m_hMciWnd, &rc);

        AdjustWindowSize(rc);
    }
}
```

7）重载 OnOK 和 OnCancel 函数，使它们为空函数。这样，当用户按回车键或 ESC 键时，程序不会退出。

注意：应在 MCIWndPlayerDlg.cpp 中包括头文件 vfw.h，在 Project Settings\Link\Object/library modules 中加入 vfw32.lib 输入库。

MCIWndPlayer 的运行界面如图 2-3 所示。图中（a）图所示是没有打开媒体文件时的状况；（b）图所示打开的是 WAVE 文件，其中的菜单是 MCIWnd 窗口工具条中的菜单按钮菜单；（c）图所示是已打开 AVI 文件，其中的菜单是上下文菜单。

该项目的完整源代码存放在\实验三\Source\MCIWndPlayer 目录中。

图 2-3 MCIWndPlayer 的运行界面