实验一　　声音录制程序的设计

一、实验目的

1、熟悉和掌握使用 Windows api 或 Java media Framework（JMF）或 javax.media.sound 进行编程的基本原理和方法。
2、熟悉各种不同的音频文件格式。
3、了解音频压缩的实现方法。

二、实验内容

使用 windows api 或 Java media Framework（JMF）或 javax.media.sound，编程实现类似于 windows 录音机一类的小工具，要求有暂停、播放、停止、前进、后退等功能。

更高要求：能实现压缩录音

三、实验环境

硬件：耳机麦克风
软件：无/JMF/JDK

四、实验步骤

1、创建一个基于对话框的工程，如 play。
2、在对话框上添加相应的按钮，实现如录音、存盘、打开文件、暂停、播放、停止、前进、后退等项功能。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：MCI 编程知识介绍；
（2）MSDN 联机帮助；
（3）网络上相关介绍；

补充资料：MCI 编程知识介绍

Windows MCI（Media Control Interface）是控制多媒体设备的高层命令接口，提供了与设备无关的控制多媒体设备的方法。MCI 可控制所有 Windows 能驱动的多媒体设备，包括 CD 音频（CD Audio）、数字视频、动画、数字化波形声音、MIDI 音序器、录像机、影碟机等。

MCI 包含在 Windows 系统的 MMSYSTEM.DLL 动态连接库中，用以协调多媒体事件和 MCI 设备驱动程序之间的通讯。一些 MCI 设备驱动程序，如影碟机设备驱动程序，可以直接控制目标设备；而另外一些 MCI 设备驱动程序，如 Wave 和 MIDI 设备驱动程序，通过 MMSYSTEM 中的

函数间接控制目标设备；还有一些 MCI 设备驱动程序提供了与其他 Windows 动态连接库的高层接口。

1、MCI 设备

使用 MCI 的应用程序通过指定 MCI 的设备类型来区分 MCI 设备，设备类型说明了设备的物理类型。表 1-1 列出了可能用到的 MCI 设备类型。

表 1-1　MCI 设备类型

MCI 设备	说　　明
Cdaudio	激光唱机
Waveaudio	数字化波形声音设备
Sequencer	MIDI 音序器
Overlay	视频叠加设备（窗口中的模拟视频）
Dat	数字化磁带音频播放机
Digitalvideo	窗口中的数字视频
Scanner	图像扫描仪
Vcr	磁带录像机或播放机
Videodisc	影碟机
Mmmovie	多媒体影片播放器
Other	未定义的 MCI 设备

必须在 Windows 系统中安装 MCI 设备的设备驱动程序才能在 Windows 中使用 MCI 设备。表 1-2 列出的是常用的 MCI 设备驱动程序。

表 1-2　常用的 MCI 设备驱动程序

MCI 设备	设备驱动程序名	备　　注
cdaudio	MCICDA.DRV	
waveaudio	MCIWAVE.DRV	
sequencer	MCISEQ.DRV	
videosidc	MCIPIONR.DRV	对先锋 LD-V4200 影碟机而言
mmmovie	MCIMMP.DRV	

MCI 设备可分为两类，如下所述。

- 简单设备：描述简单 MCI 设备，只需指定 MCI 设备名，如描述 CD 音频只需指定 cdaudio。
- 复合设备：描述复合 MCI 设备不仅要说明设备类型，而且还要指定一个设备元素或媒体元素。对大多数复合设备来说，其设备元素是一个源或目标数据文件，其元素名就是相应的文件名。复合 MCI 设备的例子有 waveaudio，sequencer，mmmovie 等。比如要描述数字化波形声音，必须指定其设备名 waveaudio，还要说明存储数字化波形声音的文件名（如 c:\windows\tada.wav）。这样，waveaudio c:\windows\tada.wav 才完整地说明了一个可操作的数字化波形设备。

2、多媒体时间格式

多媒体中各种媒体都与时间密切相关，都可看成是随时间而变化的数据流。媒体播放的持续时间或媒体当前的播放位置等重要信息都用时间来标识。因此，在多媒体编程中，如何表示时间，即时间格式是非常重要的。

在多媒体编程中，常用的时间格式有毫秒（milliseconds）、轨（tracks，通常用于音频）和帧（frames，通常用于视频）。此外，活动图像和电视工程师协会（Society of Motion Picture and Television Engineer）定义的一种特殊的时间格式 SMTPE，也在多媒体编程中得到了广泛的应用。

Microsoft 提供的 MMSYSTEM.H 文件定义了一系列的关于时间格式的常量。以它们为参数，通过 MCI_SET 命令可以将 MCI 的时间格式设定为不同的形式。常用的 MCI 时间格式包括：

MCI_FORMAT_FRAMES（帧）
MCI_FORMAT_MILLISECONDS（毫秒）
MCI_FORMAT_HMS（小时/分/秒）
MCI_FORMAT_MSF（分/秒/帧）
MCI_FORMAT_SMPTE_24（SMPTE24 帧）
MCI_FORMAT_SMPTE_25（SMPTE25 帧）
MCI_FORMAT_SMPTE_30（SMPTE30 帧）
MCI_FORMAT_SMPTE_30DROP（SMPTE30 帧，有掉帧）
MCI_FORMAT_TMSF（轨/分/秒/帧）
MCI_FORMAT_BYTES（字节，指定脉冲编码数据格式）
MCI_FORMAT_SAMPLES（样本）

3、MCI 函数与命令

Microsoft 提供的 MMSYSTEM.H 文件中定义了调用 MCI 功能的数据类型和函数原型。在使用 MCI 功能的任何源模块中都应包含该文件。

应用程序通过向 MCI 设备发送命令（命令消息或命令字符串）来控制 MCI 设备。MCI 命令可以分为 4 类，如下所述。

1）系统命令：直接由 MCI 解释并由系统处理，是不传送到 MCI 设备的命令。
2）通用命令：所有的 MCI 设备都支持的 MCI 命令。
3）可选命令：MCI 设备可选择使用的 MCI 命令。
4）专用命令：针对某类 MCI 设备或集合的专有 MCI 命令。

MCI 定义了两种接口方式，即命令消息方式和命令字符串方式。相应地，MCI 命令可分为命令消息和命令字符串。主要的 MCI 命令及分类如表 1-3 所示，这些命令可能具有其相应的扩展形式。

表 1-3　MCI 命令列表

MCI 命令消息	MCI 命令字符串	MCI 命令说明	MCI 命令类型
MCI_SYSINFO	SYSINFO	返回有关 MCI 设备的信息	系统命令
MCI_BREAK	BREAK	为一个指定的 MCI 设备设置一个终止键	
MCI_SOUND	SOUND	播放一段 Windows 指定的系统声音	
MCI_CLOSE	CLOSE	关闭一个 MCI 设备	通用命令
MCI_GETDEVCAPS	GETDEVCAPS	获得一个 MCI 设备的性能参数	
MCI_INFO	INFO	从一个 MCI 设备得到有关的信息	
MCI_OPEN	OPEN	初始化一个 MCI 设备	
MCI_STATUS	STATUS	从一个 MCI 设备返回有关的状态信息	
MCI_LOAD	LOAD	从一个磁盘文件中加载数据	可选命令

MCI_PAUSE	PAUSE	暂停播放或记录	
MCI_PLAY	PLAY	开始播放数据	
MCI_RECORD	RECORD	开始记录数据	
MCI_RESUME	RESUME	重新开始播放或记录	
MCI_SAVE	SAVE	将数据存储到磁盘文件中	
MCI_SEEK	SEEK	向前或向后检索	
MCI_SET	SET	设置设备信息	
MCI_STOP	STOP	停止播放或记录	

MCI 命令都可以带两个标志命令来控制命令的执行方式，即“wait”和“notify”。它们在命令消息和命令字符串接口中的形式和意义如表 1-4 所示。

表 1-4　MCI 标志命令

命 令 消 息	命令字符串	功　　能
MCI_WAIT	wait	通知 MCI 设备等，MCI 命令执行完后，才能将控制权还给应用程序
MCI_NOTIFY	notify	通知 MCI 设备等，立刻将控制权交给应用程序，但当命令执行完后，向应用程序发送 MM_MCINOTIFY 消息

所有的 MCI 函数名都以 mci 为前缀。对应于 MCI 命令消息和命令字符串接口方式，MCI 函数也分为两类，即命令消息函数和命令字符串函数。在 MMSYSTEM.H 中定义了这些函数的原型。MCI 函数如表 1-5 所示。

表 1-5　MCI 函数

函　数　名	功　　能	类　　型
mciSendCommand	发送命令消息	命令消息接口函数
mciGetDeviceID	获取 MCI 设备的 ID	
mciSetYieldProc	设定一个回调函数，在结束带 wait 标志的命令时调用	
mciGetYieldProc	获取当前的回调函数	
mciSendString	发送命令字符串	命令字符串接口函数
mciGetErrorString	获取当前 MCI 错误的字符串描述	公用函数

（1）MCI 命令消息接口方式

MCI 命令消息接口方式利用消息和数据结构来给多媒体设备发送命令和接收 MCI 设备传来的的信息。这种方式的接口函数主要有 3 个，即 mciSendCommand，mciGetDeviceID 和 mciGetErrorString。它们的函数原型如下：

```
MCIERROR mciSendCommand(
MCIDEVICEID wDevice,        // 设备 ID
UINT uMsg,                  // 命令消息
DWORD fdwCommand,           // 命令消息标志
DWORD dwParam               // 命令消息使用的结构参数地址
);
```

```
MCIDEVICEID mciGetDeviceID( LPCTSTR lpszDevice );    // 设备类型
BOOL mciGetErrorString(
DWORD fdwError,                 // 错误代码
LPTSTR lpszErrorText,           // 错误描述
UINT cchErrorText               // 错误描述长度
);
```

mciGetDeviceID 通过传送 MCI 设备名 lpszDevice 来获取用于 MCI_OPEN 命令消息打开 MCI 设备的标识号 wDeviceID，其值可用于 mciSendCommand 的参数 wDeviceID。mciSendCommand 用于向标识号为 wDeviceID 的 MCI 设备发送命令消息 uMsg。当用 mciSendCommand 发送 MCI_OPEN 命令消息打开一个设备时，将自动创建一个设备标识号。如果设备打开成功，可以从 MCI_OPEN_PARMS 结构的 wDeviceID 数据域中取得该设备的标识号，该值将保存以供后续的 MCI 命令使用。如果 mciSendCommand 调用成功，则返回值为 0；否则表示设备驱动出错，这时可用 mciGetErrorString 来取得错误信息的文字描述。

MCI 提供一个名为 MCI_ALL_DEVICE_ID 的特殊设备标识号。当前所有已打开的 MCI 设备都将接收到对 MCI_ALL_DEVICE_ID 发送的任何 MCI 命令。

使用任何一个 MCI 设备前都应先用 MCI_OPEN 打开它。打开 MCI 设备时，要求指定相应的 MCI_OPEN_PARMS 结构。如果打开设备成功，则该结构的 wDeviceID 域返回 MCI 设备的标识号 ID。

使用 MCI_OPEN 命令消息时可使用的命令消息标志如表 1-6 所示。MCI_OPEN_PARMS 结构定义如下：

```
typedef struct
{
    DWORD          dwCallback;          // 回调窗口句柄
    MCIDEVICEID    wDeviceID;           // 设备打开成功，返回的设备号
    LPCSTR         lpstrDeviceType;     // 设备类型
    LPCSTR         lpstrElementName;    // 复合设备的设备元素，通常为文件名
    LPCSTR         lpstrAlias;          // 指定的设备别名
} MCI_OPEN_PARMS;
```

表 1-6 MCI_OPEN 命令消息标志

消 息 标 志	意 义
MC_OPEN_ALIAS	MCI_OPEN_PARMS 结构的 lpstrAlias 域中指定了设备别名
MC_OPEN_ELEMENT	MCI_OPEN_PARMS 结构的 lpstrElementName 域中指定了设备元素
MC_OPEN_SHAREABLE	按共享设备方式打开设备
MC_OPEN_TYPE	MCI_OPEN_PARMS 结构的 lpstrDeviceType 域中指定了设备类型
MC_OPEN_TYPE_ID	MCI_OPEN_PARMS 结构的 lpstrDeviceType 域中指定了设备类型 ID

打开一个简单的 MCI 设备不需要指定设备元素，即不需要指定一个数据文件，所以可以仅仅指定 MCI_OPEN_PARMS 结构中的 wDeviceID 和 lpstrDeviceType 两个数据域。要打开一个复合的 MCI 设备，必须指定设备元素数据域 lpstrElementName 和设备类型数据域 lpstrDeviceType。对于打开复合 MCI 设备，有以下 3 种方式可供选择：

1）为确定 MCI 设备的性能，可以只指定设备的类型来打开 MCI 设备。这时，只允许确定 MCI 设备的性能，然后关闭设备，一般不能进行其他的操作。

2）为使一个设备元素与指定设备相联系，应同时指定设备元素（数据文件名）和设备类型。这时可对设备进行相应的各种操作。

3）在使用隐含的 MCI 设备时，可只指定 MCI 设备元素（数据文件名），而把设备类型指定为 NULL；MCI 将根据设备元素的扩展名从系统定义中选择隐含约定的 MCI 设备。

应用程序在使用完一个 MCI 设备后应明确地关闭该 MCI 设备。MCI_CLOSE 命令消息用于关闭并释放 MCI 设备，即取消应用程序对 MCI 设备或设备元素的访问权。

当用 MCI_SYSINFO 命令消息获取 MCI 设备系统信息时，需要在 mciSendCommand 的 dwParam 参数中指定 MCI_SYSINFO_PARMS 结构的地址，系统信息将通过该结构返回。与 MCI_SYSINFO 命令消息相关的消息标志如表 1-7 所示。 MCI_SYSINFO_PARMS 结构定义如下：

```
typedef struct
{
    DWORD dwCallback;          // 回调窗口句柄
    LPSTR lpstrReturn;         // 返回信息缓冲区地址
    DWORD dwRetSize;           // 返回信息大小
    DWORD dwNumber;            // 索引号
    UINT  wDeviceType;         // 设备类型
} MCI_SYSINFO_PARMS;
```

表 1-7 MCI_SYSINFO 命令消息标志

消 息 标 志	意 义
MCI_SYSINFO_QUANTITY	返回指定的设备类型的数目，如与 MCI_SYSINFO_OPEN 一起使用，则仅返回已打开设备的数目
MCI_SYSINFO_NAME	返回设备名，如与 MCI_SYSINFO_OPEN 一起使用，则仅返回已打开设备的名称
MCI_SYSINFO_OPEN	仅返回已打开设备的信息
MCI_SYSINFO_INSTALLNAME	指定设备的安装名称

在使用 MCI 设备时，还应注意共享、等待与通告等标志的使用。

下例说明了用命令消息接口播放数字化波形声音的方法：

```
// 其他代码
……
// 打开设备
MCI_OPEN_PARMS OpenParams;
OpenParams.lpstrDeviceType = "waveaudio";
OpenParams.lpstrElementName = "c:\\windows\\tada.wav";
MCIERROR ErrorID = mciSendCommand(NULL,
                                  MCI_OPEN,
                                  MCI_OPEN_ELEMENT|MCI_OPEN_TYPE,
                                  (DWORD)(LPMCI_OPEN_PARMS)&OpenParams);
if (ErrorID != 0)
{
    // 错误处理
    ……
}
```

```
else
{
    // 播放操作
    WORD wDeviceID = OpenParams.wDeviceID;
    MCI_PLAY_PARMS PlayParams;
    ErrorID = mciSendCommand(wDeviceID,
                             MCI_PLAY,
                             MCI_NOTIFY,
                             (DWORD)(LPMCI_PLAY_PARMS)&PlayParams);
    if (ErrorID != 0)
    {
        // 播放出错，需关闭设备
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return ErrorID;
    }
}

// 其他代码
……
```

（2）MCI 命令字符串接口方式

MCI 命令字符串方式使用 ASCII 字符串来发送驱动 MCI 设备的命令，这种方式采用的接口函数有 mciSendString、mciGetErrorString。

mciSendString 用于向 MCI 设备发送命令字符串，其函数原型如下：

```
MCIERROR mciSendString(
LPCTSTR lpszCommand,
LPTSTR lpszReturnString,
UINT cchReturn,
HANDLE hwndCallback
);
```

第一个参数是远程指针 lpszCommand，指向一个以 NULL 结尾的 MCI 命令字符串，该字符串的语法格式如下所示：

command device_name argument

其第二个参数 lpszReturnString 指向一个用于存储 MCI 命令执行后返回的字符串信息的缓冲区。第三个参数 cchReturn 用于指定该字符串缓冲区的大小。MCI 命令执行后，返回的信息将存放在 lpszReturnString 中，如果返回信息的长度大于 cchReturn 声明的长度，MCI 将返回一个错误代码。如设置 lpszReturnString 为 NULL，则将忽略返回信息。

其第四个参数 hwndCallback 指定一个接受 MM_MCINOTIFY 的窗口的句柄，除非 MCI 命令中包含了 notify 标志，否则该参数可忽略。如该函数返回非 0 值，则说明调用失败，可用 mciGetErrorString 取得包含错误信息的字符串。

MCI 的系统、通用和可选命令字符串列表于表 2-3 中，每一个命令字符串都对应着相应的命令消息。同样，这些命令也可具有其相应的扩展形式。

下例说明了用命令字符串接口播放数字化波形声音的方法：

```
// 其他代码
```

```
……
// 打开设备
char szReturn[256];
MCIERROR ErrorID =
mciSendString("open c:\\windows\\tada.wav type waveaudio alias wave",
                szReturn,
                sizeof(szReturn),
                hWnd);        // hWnd 已定义，或设为 NULL
if (ErrorID != 0)
{
    // 错误处理
    ……
}
else
{
    // 播放操作
    ErrorID = mciSendString("play wave",
                            szReturn,
                            sizeof(szReturn),
                            hWnd);      // hWnd 已定义，或设为 NULL
    if (ErrorID != 0)
    {
        // 播放出错，需关闭设备
        mciSendString("close wave", NULL, 0, NULL);
        return ErrorID;
    }
}

// 其他代码
……
```

（3）示例

1）下例定义的函数 GetNumberOfDevices 利用 MCI 接口函数获取当前系统已打开的 MCI 设备的数量。

```
////////////////////////////////////////////////////////////////////////
// GetNumberOfDevices
// 获取当前系统已打开的 MCI 设备数量
// 参数：无
// 返回值: 当前系统已打开的 MCI 设备数量
////////////////////////////////////////////////////////////////////////
int GetNumberOfDevices (void)
{
    MCI_SYSINFO_PARMS sysinfo;
    DWORD dwDevices;

    sysinfo.lpstrReturn = (LPSTR)(LPDWORD)&dwDevices;
    sysinfo.dwRetSize = sizeof(DWORD);
    if (mciSendCommand(MCI_ALL_DEVICE_ID,
```

```
                        MCI_SYSINFO,
                        MCI_SYSINFO_OPEN | MCI_SYSINFO_QUANTITY,
                        (DWORD)(LPMCI_SYSINFO_PARMS)&sysinfo) != 0)
        return 0;       // 出错
    else
        return (int)dwDevices;
}
```

2）下例说明了播放 CD 音频的方法。

```
//////////////////////////////////////////////////////////////////////
// PlayCDTrack
//
// 使用 MCI_OPEN, MCI_PLAY 播放指定的 CD 音频音轨，开始播放即
// 返回，当播放结束时，将通知回调窗口
//
// 参数：hWndNotify ―― 接收通知消息的回调窗口句柄
//       bTrack ―― 指定的音轨
//
// 返回值：  0 ―― 成功
//          非 0 ―― 失败，返回值为 MCI 错误码
//////////////////////////////////////////////////////////////////////
DWORD PlayCDTrack(HWND hWndNotify, BYTE bTrack)
{
    UINT wDeviceID;
    DWORD dwReturn;
    MCI_OPEN_PARMS mciOpenParms;
    MCI_SET_PARMS mciSetParms;
    MCI_PLAY_PARMS mciPlayParms;

    // 打开 CD 音频设备
    mciOpenParms.lpstrDeviceType = "cdaudio";
    if (dwReturn = mciSendCommand(NULL,
                                  MCI_OPEN,
                                  MCI_OPEN_TYPE,
                                  (DWORD)(LPVOID) &mciOpenParms))
    {
        // 打开设备失败，返回错误码
        return (dwReturn);
    }

    // 打开设备成功，获取设备 ID
    wDeviceID = mciOpenParms.wDeviceID;

    // 设置时间格式为 TMSF
    mciSetParms.dwTimeFormat = MCI_FORMAT_TMSF;
```

```
    if (dwReturn = mciSendCommand(wDeviceID,
                                  MCI_SET,
                                  MCI_SET_TIME_FORMAT,
                                  (DWORD)(LPVOID) &mciSetParms))
    {
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (dwReturn);
    }
    // 开始播放指定音轨，当音轨播放完毕时，用 MM_MCINOTIFY 通知
    // 消息通知父窗口。如设备出错，关闭设备
    mciPlayParms.dwFrom = 0L;
    mciPlayParms.dwTo = 0L;
    mciPlayParms.dwFrom = MCI_MAKE_TMSF(bTrack, 0, 0, 0);
    mciPlayParms.dwTo = MCI_MAKE_TMSF(bTrack + 1, 0, 0, 0);
    mciPlayParms.dwCallback = (DWORD) hWndNotify;
    if (dwReturn = mciSendCommand(wDeviceID,
                                  MCI_PLAY,
                                  MCI_FROM | MCI_TO | MCI_NOTIFY,
                                  (DWORD)(LPVOID) &mciPlayParms))
    {
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (dwReturn);
    }

    return (0L);
}
```

3）下例将播放指定的 MIDI 文件。

```
////////////////////////////////////////////////////////////////////////
// PlayMIDIFile
//
// 使用 MCI_OPEN 和 MCI_PLAY 播放指定的 MIDI 文件，开始播放即返回，
// 当播放结束时，将通知回调窗口
//
// 参数：hWndNotify ―― 接收通知消息的回调窗口句柄
//       lpszMIDIFileName ―― 指定的 MIDI 文件名
//
// 返回值：0 ―― 成功
//         非 0 ―― 失败，返回值为 MCI 错误码
////////////////////////////////////////////////////////////////////////
DWORD PlayMIDIFile(HWND hWndNotify, LPSTR lpszMIDIFileName)
{
    UINT wDeviceID;
    DWORD dwReturn;
    MCI_OPEN_PARMS mciOpenParms;
```

```
MCI_PLAY_PARMS mciPlayParms;
MCI_STATUS_PARMS mciStatusParms;
MCI_SEQ_SET_PARMS mciSeqSetParms;

// 用指定的文件名打开设备
// MCI 将试图选取 MIDI 映射器作为输出端口
mciOpenParms.lpstrDeviceType = "sequencer";
mciOpenParms.lpstrElementName = lpszMIDIFileName;
if (dwReturn = mciSendCommand(NULL, MCI_OPEN,
                                    MCI_OPEN_TYPE | MCI_OPEN_ELEMENT,
                                    (DWORD)(LPVOID) &mciOpenParms))
{
    // 打开设备失败，返回错误码
    return (dwReturn);
}
// 打开设备成功，获取设备 ID
wDeviceID = mciOpenParms.wDeviceID;

// 检查输出端口是否为 MIDI 映射器
mciStatusParms.dwItem = MCI_SEQ_STATUS_PORT;
if (dwReturn = mciSendCommand(wDeviceID, MCI_STATUS,
                                    MCI_STATUS_ITEM, (DWORD)(LPVOID) &mciStatusParms))
{
    mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
    return (dwReturn);
}
// 输出端口不是 MIDI 映射器
// 询问用户是否继续
if (LOWORD(mciStatusParms.dwReturn) != MIDI_MAPPER)
{
    if (MessageBox(hMainWnd, "MIDI 映射器不存在，继续吗 ?", "", MB_YESNO) == IDNO)
    {
        // 不继续，关闭设备并返回
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (0L);
    }
}

// 开始播放，当播放完毕时，用 MM_MCINOTIFY 通知
// 消息通知父窗口。如设备出错，关闭设备
mciPlayParms.dwCallback = (DWORD) hWndNotify;
if (dwReturn = mciSendCommand(wDeviceID, MCI_PLAY, MCI_NOTIFY,
                                    (DWORD)(LPVOID) &mciPlayParms))
{
    mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
    return (dwReturn);
}
```

```
    return (0L);
}
```

4）下例说明了录制并存储数字化波形声音的方法。

```
//////////////////////////////////////////////////////////////////////
// RecordWAVEFile
// 使用 MCI_OPEN，MCI_RECORD 和 MCI_SAVE 命令消息录制并存储数字
// 化波形声音
// 参数：dwMilliSeconds —— 录音的时间（ms）
//          lpszFileName —— 指定的存盘文件名
//
// 返回值： 0 —— 成功
//          非 0 —— 失败，返回值为 MCI 错误码
//////////////////////////////////////////////////////////////////////
DWORD RecordWAVEFile(DWORD dwMilliSeconds, LPSTR lpszFileName)
{
    UINT wDeviceID;
    DWORD dwReturn;
    MCI_OPEN_PARMS mciOpenParms;
    MCI_RECORD_PARMS mciRecordParms;
    MCI_SAVE_PARMS mciSaveParms;
    MCI_PLAY_PARMS mciPlayParms;

    // 用新文件打开波形声音，以进行录音
    mciOpenParms.lpstrDeviceType = "waveaudio";
    mciOpenParms.lpstrElementName = "";
    if (dwReturn = mciSendCommand(0, MCI_OPEN,
                                  MCI_OPEN_ELEMENT | MCI_OPEN_TYPE,
                                  (DWORD)(LPVOID) &mciOpenParms))
    {
        // 打开设备失败，返回错误码
        return (dwReturn);
    }

    // 打开设备成功，获取设备 ID
    wDeviceID = mciOpenParms.wDeviceID;

    // 按指定的时间（ms）长度开始录音，等待录音结束再继续执行程序
    // 设备的时间格式应设为 ms
    mciRecordParms.dwTo = dwMilliSeconds;
    if (dwReturn = mciSendCommand(wDeviceID, MCI_RECORD,
                                  MCI_TO | MCI_WAIT, (DWORD)(LPVOID) &mciRecordParms))
    {
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (dwReturn);
```

```
    }

    // 播放录音，并提示用户存盘
    mciPlayParms.dwFrom = 0L;
    if (dwReturn = mciSendCommand(wDeviceID, MCI_PLAY,
                                  MCI_FROM | MCI_WAIT, (DWORD)(LPVOID) &mciPlayParms))
    {
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (dwReturn);
    }
    if (MessageBox(hMainWnd, "将录音存盘吗 ?", "", MB_YESNO) == IDNO)
    {
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (0L);
    }

    // 将录音存为 lpszFileName 文件， 等待存盘结束再继续执行程序
    mciSaveParms.lpfilename = lpszFileName;
    if (dwReturn = mciSendCommand(wDeviceID, MCI_SAVE,
                                  MCI_SAVE_FILE | MCI_WAIT,
                                  (DWORD)(LPVOID) &mciSaveParms))
    {
        mciSendCommand(wDeviceID, MCI_CLOSE, 0, NULL);
        return (dwReturn);
    }

    return (0L);
}
```

5）下例说明了播放数字化视频 AVI 的方法。

```
////////////////////////////////////////////////////////////////////////
// PlayMovie
// 使用 MCI_OPEN 和 MCI_PLAY 命令消息打开并播放数字化视频 AVI
// 参数：lpszFileName 指定的 AVI 文件名
//       dwFrom —— 开始播放位置
//       dwTo —— 结束播放位置
//
// 返回值：0 —— 成功
//         非 0 —— 失败，返回值为 MCI 错误码
////////////////////////////////////////////////////////////////////////
DWORD PlayMovie(LPSTR lpszFileName, DWORD dwFrom, DWORD dwTo)
{
    DWORD dwReturn;

    // 打开 AVI
```

```
MCI_DGV_OPEN_PARMS      mciOpen;
mciOpen.dwCallback      = 0L;
mciOpen.wDeviceID       = 0;
mciOpen.lpstrDeviceType = AVI_VIDEO;
mciOpen.lpstrElementName = lpszFileName;
mciOpen.lpstrAlias      = NULL;
mciOpen.dwStyle         = 0;
mciOpen.hWndParent  = NULL;
if (dwReturn=mciSendCommand(0, MCI_OPEN,
                            (DWORD)(MCI_OPEN_TYPE),
                            (DWORD)(LPMCI_DGV_OPEN_PARMS)&mciOpen) )
{
    // 打开设备失败，返回错误码
    return dwReturn;
}

// 播放 AVI
MCI_DGV_PLAY_PARMS mciPlay;      // play parameters
DWORD dwFlags = 0;

// 打开设备成功，获取设备 ID
WORD wDevID = mciOpen.wDeviceID；

// 检查开始播放位置，如不等于 0，则进行设置
if (dwFrom)
{
    mciPlay.dwFrom = dwFrom;    // 设置参数
    dwFlags |= MCI_FROM;        // 设置标志
}

// 检查结束播放位置，如不等于 0，则进行设置
if (dwTo)
{
    mciPlay.dwTo = dwTo;        // 设置参数
    dwFlags |= MCI_TO;          // 设置标志
}

// 用 MCI_PLAY 命令播放并返回结果
return mciSendCommand(wDevID, MCI_PLAY, dwFlags,
                      (DWORD)(LPVOID)&mciPlay);
```