CD PLAYER PROJECT
AT KUMARAGURU COLLEGE OF TECHNOLOGY

PROJECT REPORT

SUBMITTED BY

MURARI. M

RAJAGOPALAN. S

SARANYAN. U

VISHAKAN. G

GUIDED BY

L.S. JAYASHREE, B.E., M.E.,
Department of Computer Science and Engineering

1999 -2000

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE AWARD OF THE DEGREE OF

BACHELOR OF SCIENCE IN

APPLIED SCIENCE - COMPUTER TECHNOLOGY

OF THE BHARATHIAR UNIVERSITY, COIMBATORE.

Department of Computer Science and Engineering

Kumaraguru College of Technology

Coimbatore - 641 006.

Kumaraguru College of Technology

Department of Computer Science and Engineering

Coimbatore-641 006

CERTIFICATE

This is to certify that the contents of the project report entitled

CD PLAYER

has been submitted by Mr ....

In Partial Fulfillment of the Requirements for the award of the

Degree of Bachelor of Science

IN Applied Science-COMPUTER TECHNOLOGY

Branch of the Bharathiar University, Coimbatore.

During the academic year 1999-2000

....[illegible].......... 22/3/2000

Guide *Professor And Head*

Certified that the candidate with University Register Number......................

was examined by us in project Viva – Voce Examination held on ....

....

Internal Examiner *External Examiner*

Surya Superfine Exporters

P. Box No. 6701, 15, New Thillai Nagar, P. N. Pudur, Coimbatore-641041 India

Telex : 0855-323 PCOIN
0855-521 PCOIN
0855-503 PCOIN

Phone : 0422-44345
Fax : 0091-0422-25800
0091-0422-22130

To

The Head Of Department,
Computer Science & Engineering,
Kumaraguru College of Technology,
Coimbatore- 641 006.

P-454

This is to certify that the following students have undergone the project entitled 'CD Player' and have completed it successfully.

1. Murari .M
2. Rajagopalan .S
3. Saranyan .U
4. Vishakan .G

During the course of this project, their attendance and conduct has been found impeccable. We wish them the very best for a bright future.

P.V.Suriyakala
[illegible]
[Managing Director]

ACKNOWLEDGEMENT

We wish to express our sincere and heartfelt gratitude to Dr.K.K.PADMANABHAN.,BSc(Engg),M.Tech.,Ph.D, our esteemed Principal, Kumaraguru College of Technology for giving us the needed encouragement in starting this project and carrying this out successfully.

Our heartfelt thanks and a deep sense of gratitude to Prof.Dr,S.Thangaswamy.,B.E(Hons),Ph.D, the Head of Department of Computer Science and Engineering for his benevolent attitude and spurring encouragement.

We would like to thank our guide L.S.Jayashree, B.E. M.E. Department of Computer Science and Engineering without whose motivation and guidance we would not have been able to embark on a project of this magnitude.

We would also like to thank our class advisor Mr.S.Andrews,M.Sc,PDGPM, for his constant support and guidance.

without which we would not have been able to complete our project successfully.

We are also immensely indebted to our project incharge **Mr.K.R.Baskaran**,B.E,M.S.,Sr.Lecturer, Department of Computer Science and Engineering , for his able guidance and motivation ,in situations when our hearts were downtrodden.

We are deeply indebted to Mrs.Suriyakala, Managing Director for enabling us to do the project in Surya Superfine Exporters.

We reciprocate the kindness shown to us by the staff members of the department of computer science, people at home and our beloved friends who have supported and helped us in getting this project done.

SYNOPSIS

The task is to design and create software to run an Audio CD. This software will play the track selected by the user.

THE PROCESS

The user enters the application by running the application Brave Hearts CD player. This application will have three frames.

(a) The main frame with the control buttons.

(b) The play section frame and,

(c) The about dialog box.

In the main frame of the application, the user can play any one of the tracks by selecting from the combo box, which lists all the tracks in the CD. There is a pause control button to stop the track at its current position. It can be resumed by reclicking the button. The play section enables the user to play a selected part of the particular track.

The play section has the 'From' and 'To' sections with three edit boxes (Minute, Second, Frame) each. There are two control buttons ‘play’ and ‘cancel’ to play the selected portion and cancel it.

The About dialog box displays about the application. The about dialog box appears in the system menu of the main frame and while exiting the application.

CONTENTS

1.INTRODUCTION

1.1 PURPOSE

1.2 SCOPE

1.3 DEFNITIONS, ABBREVATIONS

1.1 PURPOSE

Due to the tremendous growth of the IT industry there is a great and drastic change in every aspect of life. The life style of people has been changed. Computer applications play a vital role in the day to day life of everyone.

As these computer applications are user friendly, people get converted towards them. This changes the life and is made easier to live. The telephones are changed into cellulars and video conferencing. The entertainment has changed from radios and television to multimedia.

The multimedia plays a fine part in entertaining the people with high quality of sound and movies. This is possible through the compact disks which is shortly called as CD. These CD's cannot be played directly in computers. It requires some special software to play them .The aim of the project is to deve [illegible] software in VISUAL C++ platform to satisfy the above requirements.

1.2 SCOPE

The application is restricted to a limited number of functions due to the complexity in the coding part. The coding has been done in such a way that this application allows us to add various other functions to it. The functions include skipping to the previous track, playing the previous track, displaying playing time of the track, adding slider, volume controls and equalizers.

The coding is done with some of the functions in the MCI class. With the help of the other functions in the MCI class, the above said developments can be added.

1.3 DEFINITIONS, ABBREVATIONS

MICROSOFT FOUNDATION CLASS LIBRARY : MCISAMPLE

AppWizard has created this MCISample application for you.This application not only demonstrates the basics of using the Microsoft Foundation classes but is also a starting point for writing your application.

This file contains a summary of what you will find in each of the files that make up your MCISample application.

MCISAMPLE.H

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CMCISampleApp application class.

MCISAMPLE.CPP

This is the main applicati~~on~~ ~~source~~ file that contains the application_class CMCISampleApp.

MCISAMPLE.RC

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Developer Studio.

RES\MCISAMPLE.ICO

This is an icon file, which is used as the application's icon. The main resource file MCISample.rc includes this icon.

RES\MCISAMPLE.RC2

This file contains resources that are not edited by Microsoft Developer Studio. You should place all resources not editable by the resource editor in this file.

MCISAMPLE.CLW

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

APPWIZARD CREATES ONE DIALOG CLASS:

MCISampleDlg.h,MCISampleDlg.cpp - the dialog. These files contain your CMCISampleDlg class. This class defines the behavior of your application's main dialog. The dialog's template is in MCISample.rc, which can be edited in Microsoft Developer Studio.

OTHER STANDARD FILES:

STDAFX.H, STDAFX.CPP

These files are used to build a precompiled header (PCH) file named MCISample.pch and a precompiled types file named StdAfx.obj.

RESOURCE.H

This is the standard header file, which defines new resource IDs. Microsoft Developer Studio reads and updates this file.

2. GENERAL DESCRIPTION

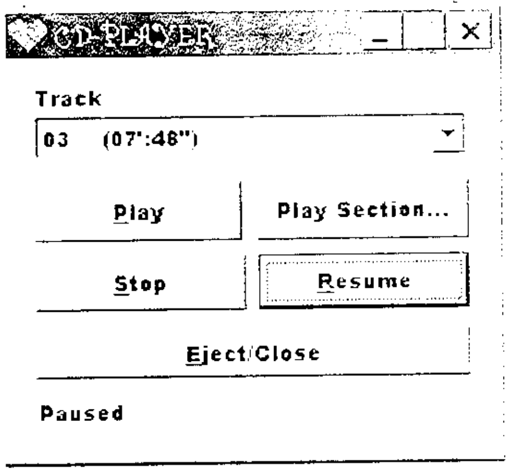
2.1 PRODUCT FEATURES

2.2 PRODUCT FUNCTIONS

2.1 PRODUCT FEATURES

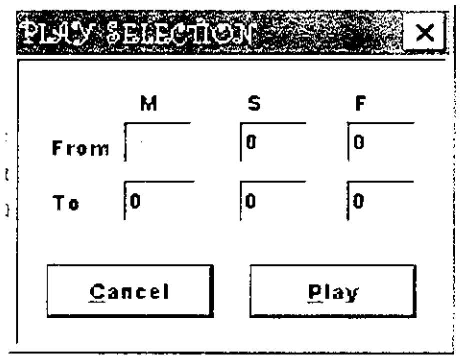
The application totally has three frames. A main frame and two subordinate frames. The main frame appears as soon as the user runs the application. It has all the main control buttons of the application.

MAIN FRAME



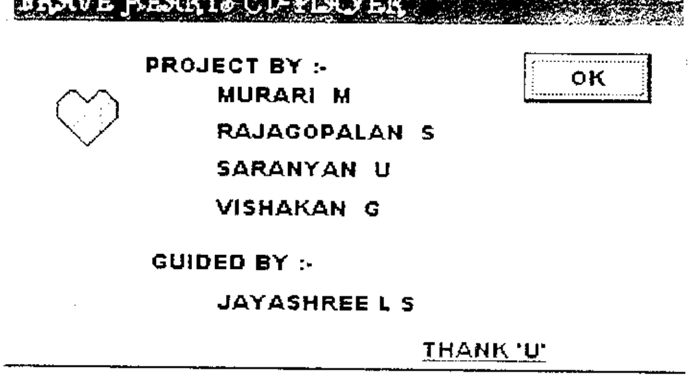
The play section frame appears as soon as the play section button in the main frame is clicked.

PLAY SELECTION FRAME



The about dialog box displays the details of the project. The dialog box appears when we exit the application.

'ABOUT DIALOGUE' FRAME



BRAVE HEARTS CD-PLAYER

PROJECT BY :-
MURARI M
RAJAGOPALAN S
SARANYAN U
VISHAKAN G

GUIDED BY :-
JAYASHREE L S

THANK 'U'

2.2 PRODUCT FUNCTIONS

Unlike the ordinary CD player, the computer requires specific software to run the audio CD's. This is software written for running an audio CD in a multimedia player. This software contains the following functions for running an audio CD...

- **Play**
- **Play section**
- **Track selection**
- **Pause & Resume**
- **Stop**
- **Eject/close**
- **Status Bar**

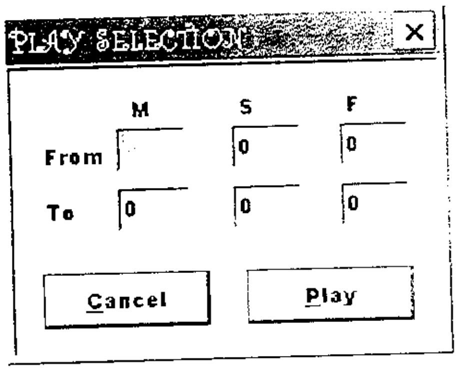
Play

The play function is used to play the selected track. The length of the track is detected and the selected track is played till the end. After completion of playing of the current track the

·next track is selected and played.

PLAY SECTION

Whenever an audio CD is played the length of the selected track is detected. Thus we are entitled to play any particular portion of the current track .We can do this by specifying the start and end of the portion of the track we want to listen to.



	M	S	F
From		0	0
To	0	0	0

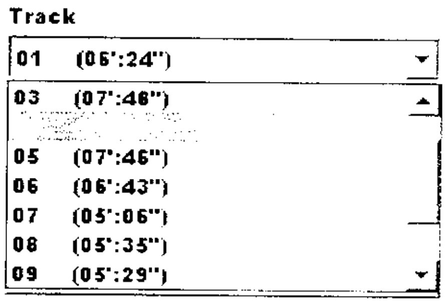
If we click on the play section button we get a window. It has 'From' and 'To' sections .In the 'from' section we can specify the start of the portion to be played and in the 'To' section we specify the track has to stop playing. The time specification done in the 'From' and 'To' sections is done in minutes, seconds and frames [denoted in the window as 'M', 'S' and 'F' respectively]. Here there are two functions 'PLAY' and 'CANCEL'.

The play button in the play section plays the portion of the track mentioned by the user in the edit boxes.

The cancel button in the play section cancels the play section and returns to the main frame of the application.

Track Selection

The tracks that are available in the CD are listed in a combo box in the main frame of the application. The play time of the tracks is also displayed within parenthesis in side of the each track. The combo box lists the tracks of the CD.A track is selected from the list box.



Pause & Resume

The 'Pause' function is to temporarily stop the current playing track. As soon as the 'Pause' button is pressed, the text in the button is changed to 'Resume'. By reclicking the button the track continues playing.

Pause
Resume

Stop

The stop function stops the current playing track. We have to select another track to play.

Eject/Close

The Eject/Close function is to open and close the door of CD drive.

Status bar

There is a status bar at the bottom of main application to indicate the status of the application. For example, if a track is currently playing then the text 'Playing....' appears in the status bar.

3. SPECIFIC REQUIREMENTS

3.1 FUNCTIONAL REQUIREMENTS

3.2 HARDWARE REQUIREMENTS

3.1 FUNCTIONAL REQUIRMENTS

Under the Functional Requirements the list of the inputs for the functions, the process under the function and the output of the function are explained under their function name.

ONPLAY

Playing a CD is written in the class 'CMCISampleDlg' and in the function 'OnPlay'. The first input for the function is the m_nTrackToPlay which has the track number obtained through getting the track number according to the format and it type converts the track number to strings through "GetAt" function and stores in the m_nTrackToPlay.

Here the track obtained from the format is stored in the variable m_strTrack. Using the control types DDX-CBString from the track selected in the Combo box of the Main frame is being stored in the variable m_strString.

The Track length is stored in the object 'msf' of the class CMsf. This is done by using the function 'GetTrackLength' defined in the class CcdAudio. This function 'GetTrackLength' in turn calls the function 'GetTrackInfo' which performs the function of retrieving the length of the track which uses the parameters dwTrack and dwItem. This function uses Microsoft Foundation Class 'MCI_STATUS_PARMS'. This in turn returns the Windows Operating System the MCI_STATUS, MCI_STATUS_ITEM, MCI_TRACK and the address of the object of the class 'MCI_STATUS_PARMS'.

The next input for the play function is the m_smsfFrom. This is obtained with the class CMsf which returns the starting position. The inputs are track number (m_nTrackToPlay), minute (0), second (0) and frame (0). The output of this class is stored in the m_tmsFrom.

The other one is m_tmsfTo which is also obtained using the class CMsf which returns the end position. The inputs are track number (m_nTrackToPlay), minute (using GetMinute()), second (using GetSecond()) and frame (using GetFrame()). The output of this class is stored in the m_tmsfTo.

With these inputs the background function Play defined in the class CcdAudio in the file CdAudio is called.The inputs for this function are dwFrom(m_tmsfFrom), dwTo(m_tmsfTo) and bAsync(0). This function uses the Microsoft Foundation Class "MCI PLAY PARMS" class. This class returns to the windows operating system to play using MCI_PLAY, dwFlag and &mci_play_parms. The above command is sent to the windows operating system only if it satisfies the condition "Async==TRUE ".

The 'Status' bar is also changed with the caption "Playing...". The buttons 'Stop' and 'Pause' are made enabled with the help of the function "EnableWindow()".

ONPLAYSECTION

Play section is just capturing a small part of the track and playing it. This is similar to the play function.

This function is performed when the "OK" button of the 'play section' frame is clicked. For the 'play section' the track number to be played is obtained in the similar manner as in the 'play' function. The starting position is obtained with the details of all the track number minutes, seconds and frame using the CTmsf class which is defined under the header file MCI.h .

The end position is obtained in the similar manner as in the 'play' function using the CTmsf class.

Then the play function is invoked to play the selected part in the track.

The 'Status' bar is also changed with the caption "Playing...". The buttons 'Stop' and 'Pause' are enabled with the help of the function "EnableWindow( )".

ONSTOP

The 'OnStop' function is invoked when the button "STOP" is clicked in the main frame. In the 'Onstop' function it calls the background function 'Stop' written in the C++ file CdAudio in the class CcdAudio.

The 'Stop' function uses the Microsoft Foundation Class MCI_GENERIC_PARMS. Using this function the dwCallback is assigned to the m_hMainWnd which activates the window to concentrate on stopping the playing track.This function in turn sends a command to the windows operating system by using send command with its parameters MCI_STOP ,dwFlag(0) and the address of the object of the class MCI_GENERIC_PARMS.

The 'Status' bar is also changed with the caption "Stopped". The buttons 'Stop' and 'Pause' are made disabled with the help of the function "EnableWindow(FALSE)".

ONPAUSE(PAUSE AND RESUME)

The 'Pause' function is made active when the Pause button of the main frame is pressed.

Inside the function to pause the playing track, the thing to be changed in the function is the Mode. Then the function carries out with Pausing the track by checking the GetMode( ) function written in the Mci.cpp. This in turn calls the function GetStatus which is also defined in the c++ file Mci.cpp. In this GetStatus function, the status of the machine is required with Microsoft Foundation Classes, MCI_STATUS_PARMS.

Using this class the function sends the Windows Operating System with the parameters MCI_STATUS, MCI_STATUS_ITEM and the address of the object of the class MCI_STATUS_PARMS.

If the condition is satisfied, then the function 'Pause' written in the c++ file CdAudio is called. Inside the function it uses the Microsoft Foundation Class, MCI_GENERIC_PARMS. The call back of the MCI_GENERIC_PARMS is assigned to the m_hMainWnd.

The function Pause sends a command to the Windows Operating System with the parameters MCI_PAUSE, the flag dwFlag as 0 and the address of the object of the class MCI_GENERIC_PARMS.

To 'Resume' the 'Paused' track once again the Mode of the track is checked, and if it is in the paused mode the Pause function calls the 'Play' function of the CdAudio file. For the Play function the starting position is obtained through the function 'GetCurrentPos' which in turn calls the 'GetStatus' function written in the c++ file Mci inside the class CMciDevice.

With those functions the Play function is called and it plays from the current position to the end of the track.

The 'Status' bar is also changed with the caption "Paused". The button 'Stop' is enabled with the help of the function "EnableWindow()". The text in the button 'Pause' is being changed to 'Resume' with the function "SetWindowText(_T("Caption"))".

Whenever the 'Resume' button is clicked, the 'Status' bar is also changed with the caption "Playing...". The buttons 'Stop' and 'Pause' are made enabled with the help of the function "EnableWindow()".

The text of the button is also changed as "Pause" with the function "SetWindowText(_T("Caption"))"

OnSelectionofTrack

Initially the unsaved data in the window were saved using the in-built function "UpdateData()". Then the track to play is obtained from the combo box which is in the format is in the variable m_strTrack. This is changed to string value using the GetAt function and is stored in the variable m_nTrackToPlay.

If the position of the track is to be obtained using the Seek function which is written under the c++ file CdAudio. This Seek function requires three parameters which are starting position, end position and status(TRUE or FALSE). Inside the seek function, the dwCallback is assigned to the m_hMainWnd. Then the SendCommand is sent to the Windows Operating System with the parameters MCI_SEEK, dwFlag with the value MCI_NOTIFY and the address of the object of the Microsoft Foundation Class, MCI_SEEK_PARMS which is used by the function Seek. That's how the track selection is made.

OnEject

This function in turn is used to open and close the door of the CD-Drive. This is done for the insertion of the CD into the CD drive. This is being performed with the help of the function OpenDoor() written in the c++ file CdAudio. Inside the function the parameter passed is also a return value of a function which returns a BOOLEAN value which is the function IsReady() which in turn calls the function GetStatus() which is already defined. In the OpenDoor function the door of the CD-Drive is opened if it is being in the closed state otherwise it is closed. This is performed using the 'Set' function which is being defined in the file CdAudio.cpp. This function uses the Microsoft Foundation Class, MCI_SET_PARMS. Using the send command the Windows Operating System is made to open and close the door of the CD-Drive.

Inside this function a separate function "EnableControls()" is called. Inside this function, the Combo box, the 'Play' button, the 'Pause' button and the 'PlaySection' are enabled and the ‘Pause’ button and the ‘Stop’ button are disabled.

OnClose

This function is to stop the running track and to close the track, which is opened. First, the CD-Drive is checked with the IsReady() function which helps in specifying that the CD-Drive is in the closed state. If it is in the closed state, then the function calls the 'Stop' function and the 'Close' function.

The 'Close' function is also defined under the c++ file Mci. In this function, the Microsoft Foundation Class. MCI_GENERIC_PARMS is used. This function in turn sends a command to the Windows Operating System using the SendCommand in which the parameters are the MCI_CLOSE, the dwFlag and the address of the object of the class MCI_GENERIC_PARMS.

3.2 HARDWARE REQUIREMENTS

Processor Speed	166 MHz
HardDisk Capacity	1 GB
RAM	16 MB
CD-ROM Drive	12x

A Multimedia kit with External Speakers

4. Design constraints

4.1 Application Modules

4.1 Application Modules

The application is developed in the Visual C++ platform.The application is not coded as a single module.The source code is divided into

- 7 C++ files with extension '.cpp'.

  CdAudio.cpp

  Hyperlink.cpp

  Mci.cpp

  MCISample.cpp

  MCISampleDlg.cpp

  PlaySectionDlg.cpp

  StdAfx.cpp

- 8 header files with extension '.h'.

  CdAudio.h

  hyperlink.h

  mci.h

  MCISample.h

  MCISampleDlg.h

  PlaySectionDlg.h

  StdAfx.h

  resource.h

- 1 resource script file with extension '.rc'.

  MCISample.rc2

BIBLIOGRAPHY

VISUAL C++ MULTIMEDIA
By
Peter Aitken
Scott Jarol

HEAVY METAL VISUAL C++ PROGRAMMING
By
Steve Holzner

VISUAL C++ 2 PROGRAMMING FOR DUMMIES
By
Michael Hyman
Bob Arnson

MICROSOFT DEVELOPERS NETWORK (MSDN)
Version 6.0

```
#include "stdafx.h"
#include "CdAudio.h"

/////////////////////////////////////////////////////////////////////////////
// CCdAudio implementation
//
const DWORD CCdAudio::FormatMilliseconds = MCI_FORMAT_MILLISECONDS;
const DWORD CCdAudio::FormatMSF                  = MCI_FORMAT_MSF;
const DWORD CCdAudio::FormatTMSF             = MCI_FORMAT_TMSF;

const DWORD CCdAudio::StatusCurrentTrack = MCI_STATUS_CURRENT_TRACK;
const DWORD CCdAudio::StatusLength        = MCI_STATUS_LENGTH;
const DWORD CCdAudio::StatusPosition      = MCI_STATUS_POSITION;
const DWORD CCdAudio::StatusStart         = MCI_STATUS_START;

const DWORD CCdAudio::TrackTypeAudio = MCI_CDA_TRACK_AUDIO;
const DWORD CCdAudio::TrackTypeOther = MCI_CDA_TRACK_OTHER;

// Open
DWORD CCdAudio::Open(BOOL bShareable /*=FALSE*/)
{
	return CMciDevice::Open(MCI_DEVTYPE_CD_AUDIO, bShareable);
}

// Set the device time format. Allowable time formats are:
// MCI_FORMAT_TMSF, MCI_FORMAT_MSF, MCI_FORMAT_MILLISECONDS
DWORD CCdAudio::SetTimeFormat(DWORD dwTimeFormat)
{
	MCI_SET_PARMS mciSetParms;

	mciSetParms.dwTimeFormat = dwTimeFormat;

	return SendCommand(MCI_SET,
		MCI_SET_TIME_FORMAT, (DWORD)(LPVOID) &mciSetParms);
}

// Gest the current time format
DWORD CCdAudio::GetTimeFormat()
{
	return GetStatus(MCI_STATUS_TIME_FORMAT);
}

// Plays a given CD audio track using MCI_OPEN, MCI_PLAY. Returns as
// soon as playback begins.
DWORD CCdAudio::PlayTrack(BYTE nTrack, BOOL bAsync /*=TRUE*/)
{
	DWORD dwReturn;
	// Save current time format
	DWORD dwOldTimeFormat = GetTimeFormat();

	// Set the time format to track/minute/second/frame (TMSF).
      if (dwReturn = SetTimeFormat(MCI_FORMAT_TMSF))
```

```
            return dwReturn;

        // Get track lenght
        CMsf msf = GetTrackLength(nTrack);

        CTmsf msfFrom = CTmsf(nTrack, 0, 0, 0);
        CTmsf msfTo = CTmsf(nTrack,
            msf.GetMinute(), msf.GetSecond(), msf.GetFrame());

        // Play the track
        if (dwReturn = Play(msfFrom, msfTo, bAsync))
            return dwReturn;

        // Restore the saved time format
        return SetTimeFormat(dwOldTimeFormat);
}

// Plays CD from a position to another. dwFrom and dwTo are interpreted
// accordingly with the current time format.
DWORD CCdAudio::Play(DWORD dwFrom /*=0L*/, DWORD dwTo /*=0L*/,
                                BOOL bAsync /*=TRUE*/)
{
        MCI_PLAY_PARMS mciPlayParms;
      mciPlayParms.dwFrom = dwFrom;
        mciPlayParms.dwTo = dwTo;

        DWORD dwFlags = MCI_FROM | MCI_TO;
        if (bAsync)
        {
              mciPlayParms.dwCallback = (DWORD) m_hMainWnd;
              dwFlags |= MCI_NOTIFY;
        }

        return SendCommand(MCI_PLAY, dwFlags, (DWORD)(LPVOID) &mciPlayParms);
}

// Stops playback
DWORD CCdAudio::Stop()
{
        MCI_GENERIC_PARMS mciGenericParms;

        mciGenericParms.dwCallback = (DWORD) m_hMainWnd;

        return SendCommand(MCI_STOP, 0, (DWORD) &mciGenericParms);
}

// Pauses playback
DWORD CCdAudio::Pause()
{
        MCI_GENERIC_PARMS mciGenericParms;

        mciGenericParms.dwCallback = (DWORD) m_hMainWnd;

        return SendCommand(MCI_PAUSE, 0, (DWORD) &mciGenericParms);
}

// Open the CD door
```

```
DWORD CCdAudio::OpenDoor(BOOL bOpen /*=TRUE*/)
{
	if (bOpen)
		return Set(MCI_SET_DOOR_OPEN);
	else
		return Set(MCI_SET_DOOR_CLOSED);
}

// Retrieves the starting position of a track
DWORD CCdAudio::GetTrackPos(DWORD dwTrack)
{
	return GetTrackInfo(dwTrack, StatusPosition);
}

// Retrieves the type of a track
DWORD CCdAudio::GetTrackType(DWORD dwTrack
{
	return GetTrackInfo(dwTrack, MCI_CDA_STATUS_TYPE_TRACK);
}

// Retrieves the length of a track
DWORD CCdAudio::GetTrackLength(DWORD dwTrack)
{
	return GetTrackInfo(dwTrack, StatusLength);
}

// Retrieves the total CD length
DWORD CCdAudio::GetMediaLength(DWORD dwTrack)
{
	return GetStatus(StatusLength);
}

// Retrieves a track information
DWORD CCdAudio::GetTrackInfo(DWORD dwTrack, DWORD dwItem)
{
	MCI_STATUS_PARMS mciStatusParms;
	mciStatusParms.dwCallback = (DWORD) m_hMainWnd;
	mciStatusParms.dwItem = dwItem;
	mciStatusParms.dwTrack = dwTrack;
	mciStatusParms.dwReturn = 0;

	SendCommand(MCI_STATUS,
		MCI_STATUS_ITEM|MCI_TRACK, (DWORD) &mciStatusParms);

	return mciStatusParms.dwReturn;
}

// Retrieves the current track
DWORD CCdAudio::GetCurrentTrack()
{
	return GetStatus(StatusCurrentTrack);
}

// Retrieves the CD current position
DWORD CCdAudio::GetCurrentPos()
{
	return GetStatus(StatusPosition);
```

```
}

// Retrieves the CD starting position
DWORD CCdAudio::GetStartPos()
{
	return GetStatus(StatusStart);
}

// Gets the number of playable tracks
DWORD CCdAudio::GetNumberOfTracks()
{
	return GetStatus(StatusNumberOfTracks);
}

// Seek to a specified position
DWORD CCdAudio::Seek(DWORD dwTo, BOOL bAsync /*=FALSE*/)
{
	return Seek(dwTo, MCI_TO, bAsync);
}

// Seek to end
DWORD CCdAudio::SeekToStart(BOOL bAsync /*=FALSE*/)
{
	return Seek(0, MCI_SEEK_TO_START, bAsync);
}

// Seek to start
DWORD CCdAudio::SeekToEnd(BOOL bAsync /*=FALSE*/)
{
	return Seek(0, MCI_SEEK_TO_END, bAsync);
}

// Generic Seek
DWORD CCdAudio::Seek(DWORD dwTo, DWORD dwFlags, BOOL bAsync)
{
	MCI_SEEK_PARMS mciSeekParms;

	if (bAsync)
	{
		mciSeekParms.dwCallback = (DWORD) m_hMainWnd;
		dwFlags |= MCI_NOTIFY;
	}

	mciSeekParms.dwTo = dwTo;

	return SendCommand(MCI_SEEK, dwFlags, (DWORD) &mciSeekParms);
}

// Checks if the device is Ready to be operated
BOOL CCdAudio::IsReady()
{
	return GetStatus(StatusReady);
}

// Generic MCI_SET command
DWORD CCdAudio::Set(DWORD dwWhat)
{
        MCI_SET_PARMS mciSetParms;

        mciSetParms.dwCallback = (DWORD) m_hMainWnd;

        return SendCommand(MCI_SET, dwWhat, (DWORD) &mciSetParms);
}

// Generic MCI_GETDEVCAPS_ITEM command
DWORD CCdAudio::GetDevCapsItem(DWORD dwItem)
{
        MCI_GETDEVCAPS_PARMS mciCapsParms;

        mciCapsParms.dwCallback = (DWORD) m_hMainWnd;
        mciCapsParms.dwItem = dwItem;
        mciCapsParms.dwReturn = 0;

        SendCommand(MCI_GETDEVCAPS, MCI_GETDEVCAPS_ITEM,
                (DWORD) &mciCapsParms);

        return mciCapsParms.dwReturn;
}
```

```
#include "stdafx.h"
#include "HyperLink.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

#define TOOLTIP_ID 1

#define SETBITS(dw, bits)       (dw |= bits)
#define CLEARBITS(dw, bits)     (dw &= ~(bits))
#define BITSET(dw, bit)         (((dw) & (bit)) != 0L)

/////////////////////////////////////////////////////////////////////////////
// CHyperLink

const DWORD CHyperLink::StyleUnderline              = 0x00000001;           //
Underline bit
const DWORD CHyperLink::StyleUseHover               0x00000002;             // Hand
over coloring bit
const DWORD CHyperLink::StyleAutoSize               0x00000004;             // Auto
size bit
const DWORD CHyperLink::StyleDownClick              0x00000008;             // Down
click mode bit
const DWORD CHyperLink::StyleGetFocusOnClick = 0x00000010;          // [illegible] focus
on click bit
const DWORD CHyperLink::StyleNoHandCursor  = 0x00000020;            // [illegible] hand
cursor bit
const DWORD CHyperLink::StyleNoActiveColor          0x00000040;             // No
active color bit

COLORREF CHyperLink::g_crLinkColor          RGB(0, 0, 255); // Blue
COLORREF CHyperLink::g_crActiveColor      = RGB(0, 128, 128);       // Dark cyan
COLORREF CHyperLink::g_crVisitedColor     = RGB(128, 0, 128);       // [illegible]
COLORREF CHyperLink::g_crHoverColor         RGB(255, 0, [illegible]);        // [illegible]
HCURSOR     CHyperLink::g_hLinkCursor               NULL;                   //
No cursor

CHyperLink::CHyperLink()
{
      m_bOverControl              = FALSE;    // Cursor not yet over [illegible]
      m_bVisited                  = FALSE;    // Link has not been visited [illegible]
      m_bLinkActive               = FALSE;    // Control doesn't own the [illegible] yet
      m_strURL.Empty();                       // Set URL to an empty [illegible]
      // Set default styles
      m_dwStyle = StyleUnderline|StyleAutoSize|[illegible]
}

CHyperLink::~CHyperLink()
{
    m_Font.DeleteObject();
}

IMPLEMENT_DYNAMIC(CHyperLink, CStatic)

BEGIN MESSAGE_MAP(CHyperLink, CStatic)
    //{{AFX_MSG_MAP(CHyperLink)
    ON_WM_CTLCOLOR_REFLECT()
    ON_WM_SETCURSOR()
    ON_WM_MOUSEMOVE()
      ON_WM_LBUTTONUP()
      ON_WM_SETFOCUS()
      ON_WM_KILLFOCUS()
      ON_WM_KEYDOWN()
      ON_WM_NCHITTEST()
      ON_WM_LBUTTONDOWN()
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CHyperLink message handlers

BOOL CHyperLink::PreTranslateMessage(MSG* pMsg)
{
    m_ToolTip.RelayEvent(pMsg);
    return CStatic::PreTranslateMessage(pMsg);
}

void CHyperLink::PreSubclassWindow()
{
      // If the URL string is empty try to set it to the window text
    if (m_strURL.IsEmpty())
        GetWindowText(m_strURL);

    // Check that the window text isn't empty.
      // If it is, set it as URL string.
    CString strWndText;
    GetWindowText(strWndText);
    if (strWndText.IsEmpty()) {
            // Set the URL string as the window text
        ASSERT(!m_strURL.IsEmpty());     // window text and URL both NULL!
            CStatic::SetWindowText(m_strURL);
    }

    // Get the current window font
    CFont* pFont = GetFont();

      if (pFont != NULL) {
            LOGFONT lf;
            pFont->GetLogFont(&lf);
            lf.lfUnderline = BITSET(m_dwStyle, StyleUnderline);
            if (m_Font.CreateFontIndirect(&lf))
                  CStatic::SetFont(&m_Font);
            // Adjust window size to fit URL if necessary
            AdjustWindow();
      }
      else {
            // if GetFont() returns NULL then probably the static
            // control is not of a text type: it's better to set
            // auto-resizing off
            CLEARBITS(m_dwStyle,StyleAutoSize);
        }

        if (!BITSET(m_dwStyle,StyleNoHandCursor))
            SetDefaultCursor();         // Try to load an "hand" cursor

    // Create the tooltip
    CRect rect;
    GetClientRect(rect);
    m_ToolTip.Create(this);

    m_ToolTip.AddTool(this, m_strURL, rect, TOOLTIP_ID);

    CStatic::PreSubclassWindow();
}

// Handler for WM_CTLCOLOR reflected message (see message map)
HBRUSH CHyperLink::CtlColor(CDC* pDC, UINT nCtlColor)
{
        ASSERT(nCtlColor == CTLCOLOR_STATIC);

        if (m_bOverControl && BITSET(m_dwStyle,StyleUseHover))
                pDC->SetTextColor(g_crHoverColor);
        else if (!BITSET(m_dwStyle,StyleNoActiveColor) && m_bLinkActive)
                pDC->SetTextColor(g_crActiveColor);
        else if (m_bVisited)
                pDC->SetTextColor(g_crVisitedColor);
        else
                pDC->SetTextColor(g_crLinkColor);

        // Set transparent drawing mode
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)GetStockObject(NULL_BRUSH);
}

void CHyperLink::OnMouseMove(UINT nFlags, CPoint point)
{

        if (m_bOverControl)             // Cursor currently over control
        {
                CRect rect;
                GetClientRect(rect);

                if (!rect.PtInRect(point))
                {
                        m_bOverControl = FALSE;
                        ReleaseCapture();
                        Invalidate();
                        return;
                }
        }
        else                            // Cursor has left control area
        {
                m_bOverControl = TRUE;
```

```
		Invalidate();
		SetCapture();
	}
}

/////////////////////////////////////////////////////////////////////////////
// "Normally, a static control does not get mouse events unless it has
// SS_NOTIFY. This achieves the same effect as SS_NOTIFY, but it's fewer
// lines of code and more reliable than turning on SS_NOTIFY in OnCtlColor
// because Windows doesn't send WM_CTLCOLOR to bitmap static controls."
// (Paul DiLascia)
UINT CHyperLink::OnNcHitTest(CPoint /*point*/)
{
		return HTCLIENT;
}

void CHyperLink::OnLButtonDown(UINT /*nFlags*/, CPoint /*point*/)
{
	if (BITSET(m_dwStyle,StyleGetFocusOnClick))
		SetFocus();				// Set the focus and make the link
active
	if (BITSET(m_dwStyle,StyleDownClick))
		FollowLink();
	m_bLinkActive = TRUE;
}

void CHyperLink::OnLButtonUp(UINT /*nFlags*/, CPoint /*point*/)
{
	if (m_bLinkActive && !BITSET(m_dwStyle,StyleDownClick))
		FollowLink();
}

BOOL CHyperLink::OnSetCursor(CWnd* /*pWnd*/, UINT /*nHitTest*/, UINT
/*message*/)
{
	if (g_hLinkCursor)
	{
		::SetCursor(g_hLinkCursor);
		return TRUE;
	}
	return FALSE;
}

void CHyperLink::OnSetFocus(CWnd* /*pOldWnd*/)
{
	m_bLinkActive = TRUE;
	Invalidate();							// Repaint to set
the focus
}

void CHyperLink::OnKillFocus(CWnd* /*pNewWnd*/)
{
	// Assume that control lost focus = mouse out
	// this avoid troubles with the Hover color
	m_bOverControl = FALSE;
	m_bLinkActive = FALSE;
```

```
    Invalidate();                                                  // Repaint to show
the focus
}

void CHyperLink::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    if (nChar == VK_SPACE)
        FollowLink();
    else
        CStatic::OnKeyDown(nChar, nRepCnt, nFlags);
}

/////////////////////////////////////////////////////////////////////////////
// CHyperLink operations

void CHyperLink::SetColors(   COLORREF crLinkColor,
                                     COLORREF crActiveColor,
                                     COLORREF crVisitedColor,
                              COLORREF crHoverColor /* = -1 */)
{
    g_crLinkColor    = crLinkColor;
    g_crActiveColor    = crActiveColor;
    g_crVisitedColor = crVisitedColor;

    if (crHoverColor == -1)
        g_crHoverColor = ::GetSysColor(COLOR_HIGHLIGHT);
    else
        g_crHoverColor = crHoverColor;
}

void CHyperLink::SetColors(HYPERLINKCOLORS& linkColors) {
    g_crLinkColor       = linkColors.crLink;
    g_crActiveColor     = linkColors.crActive;
    g_crVisitedColor = linkColors.crVisited;
    g_crHoverColor      = linkColors.crHover;
}

void CHyperLink::GetColors(HYPERLINKCOLORS& linkColors) {
    linkColors.crLink = g_crLinkColor;
    linkColors.crActive = g_crActiveColor;
    linkColors.crVisited = g_crVisitedColor;
    linkColors.crHover = g_crHoverColor;
}

void CHyperLink::SetLinkCursor(HCURSOR hCursor) {
    ASSERT(hCursor != NULL);

    g_hLinkCursor = hCursor;
    if (g_hLinkCursor == NULL)
        SetDefaultCursor();
}

HCURSOR CHyperLink::GetLinkCursor() {
    return g_hLinkCursor;
}

BOOL CHyperLink:: ModifyLinkStyle(DWORD dwRemove, DWORD dwAdd,
                                                  BOOL bApply /* =TRUE */)
{
      // Check if we are adding and removing the same style
      if ((dwRemove & dwAdd )!= 0L)
            return FALSE;

      // Remove old styles and set the new ones
      CLEARBITS(m_dwStyle, dwRemove);
      SETBITS(m_dwStyle, dwAdd);

      if (bApply && ::IsWindow(GetSafeHwnd())) {
            // If possible, APPLY the new styles on the fly
            if (BITSET(dwAdd,StyleUnderline) || BITSET(dwRemove,StyleUnderline))
                  SwitchUnderline();
            if (BITSET(dwAdd,StyleAutoSize))
                  AdjustWindow();
            if (BITSET(dwRemove,StyleUseHover))
                  Invalidate();
      }
      return TRUE;
}

DWORD CHyperLink::GetLinkStyle() const {
      return m_dwStyle;
}

void CHyperLink::SetURL(CString strURL)
{
      m_strURL = strURL;

      if (::IsWindow(GetSafeHwnd())) {
            ShowWindow(SW_HIDE);
            AdjustWindow();
            m_ToolTip.UpdateTipText(strURL, this, TOOLTIP_ID);
            ShowWindow(SW_SHOW);
      }
}

CString CHyperLink::GetURL() const {
      return m_strURL;
}

void CHyperLink::SetWindowText(LPCTSTR lpszText)
{
      ASSERT(lpszText != NULL);

      if (::IsWindow(GetSafeHwnd())) {
            // Set the window text and adjust its size while the window
            // is kept hidden in order to allow dynamic modification
            ShowWindow(SW_HIDE);                          // Hide window
            // Call the base class SetWindowText()
            CStatic::SetWindowText(lpszText);
            // Resize the control if necessary
            AdjustWindow();
            ShowWindow(SW_SHOW);                          // Show window
      }
}
```

```
void CHyperLink::SetFont(CFont* pFont)
{
        ASSERT(::IsWindow(GetSafeHwnd()));
        ASSERT(pFont != NULL);

        // Set the window font and adjust its size while the window
        // is kept hidden in order to allow dynamic modification
        ShowWindow(SW_HIDE);                             // Hide window
        LOGFONT lf;
        // Create the new font
        pFont->GetLogFont(&lf);
        m_Font.DeleteObject();
        m_Font.CreateFontIndirect(&lf);
        // Call the base class SetFont()
        CStatic::SetFont(&m_Font);
        // Resize the control if necessary
        AdjustWindow();
        ShowWindow(SW_SHOW);                             // Show window
}

// Function to set underline on/off
void CHyperLink::SwitchUnderline()
{
        LOGFONT lf;
        CFont* pFont = GetFont();
        if (pFont != NULL) {
                pFont->GetLogFont(&lf);
                lf.lfUnderline = BITSET(m_dwStyle,StyleUnderline);
                m_Font.DeleteObject();
                m_Font.CreateFontIndirect(&lf);
                SetFont(&m_Font);
        }
}

// Move and resize the window so that its client area has the same size
// as the hyperlink text. This prevents the hyperlink cursor being active
// when it is not over the text.
void CHyperLink::AdjustWindow()
{
        ASSERT(::IsWindow(GetSafeHwnd()));

        if (!BITSET(m_dwStyle,StyleAutoSize))
          return;

      // Get the current window rect
      CRect rcWnd;
      GetWindowRect(rcWnd);

        // For a child CWnd object, window rect is relative to the
        // upper-left corner of the parent window's client area.
      CWnd* pParent = GetParent();
      if (pParent)
          pParent->ScreenToClient(rcWnd);

        // Get the current client rect
        CRect rcClient;
        GetClientRect(rcClient);

        // Calc border size based on window and client rects
        int borderWidth = rcWnd.Width() - rcClient.Width();
        int borderHeight = rcWnd.Height() - rcClient.Height();

    // Get the extent of window text
    CString strWndText;
    GetWindowText(strWndText);

    CDC* pDC = GetDC();
    CFont* pOldFont = pDC->SelectObject(&m_Font);
    CSize Extent = pDC->GetTextExtent(strWndText);
    pDC->SelectObject(pOldFont);
    ReleaseDC(pDC);

    // Get the text justification style
    DWORD dwStyle = GetStyle();

    // Recalc window size and position based on text justification
    if (BITSET(dwStyle, SS_CENTERIMAGE))
            rcWnd.DeflateRect(0, (rcWnd.Height() - Extent.cy) / 2);
    else
            rcWnd.bottom = rcWnd.top + Extent.cy;

    if (BITSET(dwStyle, SS_CENTER))
            rcWnd.DeflateRect((rcWnd.Width() - Extent.cx) / 2, 0);
    else if (BITSET(dwStyle,SS_RIGHT))
            rcWnd.left  = rcWnd.right - Extent.cx;
      else // SS_LEFT
            rcWnd.right = rcWnd.left + Extent.cx;

      // Move and resize the window
      MoveWindow(rcWnd.left, rcWnd.top, rcWnd.Width() + borderWidth,
            rcWnd.Height() + borderHeight);
}

void CHyperLink::SetVisited(BOOL bVisited /* = TRUE */) {
      m_bVisited = bVisited;
}

BOOL CHyperLink::IsVisited() const {
      return m_bVisited;
}

/////////////////////////////////////////////////////////////////////////////
// CHyperLink implementation


void CHyperLink::SetDefaultCursor()
{
      if (g_hLinkCursor == NULL)          // No cursor handle - load our own
      {
        // Get the windows directory
            CString strWndDir;
            GetWindowsDirectory(strWndDir.GetBuffer(MAX_PATH), MAX_PATH);
            strWndDir.ReleaseBuffer();
            strWndDir += _T("\\winhlp32.exe");
            // This retrieves cursor #106 from winhlp32.exe, which is a hand
pointer
            HMODULE hModule = LoadLibrary(strWndDir);
            if (hModule) {
                  HCURSOR hHandCursor = ::LoadCursor(hModule,
MAKEINTRESOURCE(106));
                  if (hHandCursor)
                        g_hLinkCursor = CopyCursor(hHandCursor);
            }
            FreeLibrary(hModule);
      }
}

LONG CHyperLink::GetRegKey(HKEY key, LPCTSTR subkey, LPTSTR retdata)
{
    HKEY hkey;
    LONG retval = RegOpenKeyEx(key, subkey, 0, KEY_QUERY_VALUE, &hkey);

    if (retval == ERROR_SUCCESS) {
        long datasize = MAX_PATH;
            TCHAR data[MAX_PATH];
            RegQueryValue(hkey, NULL, data, &datasize);
            lstrcpy(retdata,data);
            RegCloseKey(hkey);
    }

    return retval;
}

// Error report function
void CHyperLink::ReportError(int nError)
{
      CString str;

      switch (nError) {
        case 0:                       str = _T("The operating system is out\nof
memory or resources."); break;
        case ERROR_FILE_NOT_FOUND:    str = _T("The specified file was not
found."); break;
            case ERROR_PATH_NOT_FOUND:      str = _T("The specified path was not
found."); break;
        case ERROR_BAD_FORMAT:        str = _T("The .EXE file is invalid\n(non-
Win32 .EXE or error in .EXE image)."); break;
        case SE_ERR_ACCESSDENIED:     str = _T("The operating system
denied\naccess to the specified file."); break;
        case SE_ERR_ASSOCINCOMPLETE:  str = _T("The filename association
is\nincomplete or invalid."); break;
        case SE_ERR_DDEBUSY:          str = _T("The DDE transaction could
not\nbe completed because other DDE transactions\nwere being processed.");
break;
        case SE_ERR_DDEFAIL:          str = _T("The DDE transaction failed.");
break;
        case SE_ERR_DDETIMEOUT:       str = _T("The DDE transaction could
not\nbe completed because the request timed out."); break;
```

```
        case SE_ERR_DLLNOTFOUND:        str = _T("The specified dynamic-link
library was not found."); break;
        //case SE_ERR_FNF:                    str = _T("Windows 95 only: The
specified file was not found."); break;
            case SE_ERR_NOASSOC:            str = _T("There is no application
associated\nwith the given filename extension."); break;
        case SE_ERR_OOM:                str = _T("There was not enough memory to
complete the operation."); break;
      //case SE_ERR_PNF:                str = _T("The specified path was not
found."); break;
            case SE_ERR_SHARE:              str = _T("A sharing violation
occurred. "); break;
        default:                        str.Format(_T("Unknown Error (%d)
occurred."), nError); break;
    }

      str = "Can't open link:\n\n" + str;
      AfxMessageBox(str, MB_ICONEXCLAMATION | MB_OK);
}


HINSTANCE CHyperLink::GotoURL(LPCTSTR url, int showcmd)
{
    TCHAR key[MAX_PATH + MAX_PATH];

    // First try ShellExecute()
    HINSTANCE result = ShellExecute(NULL, _T("open"), url, NULL,NULL, showcmd);

    // If it failed, get the .htm regkey and lookup the program
    if ((UINT)result <= HINSTANCE_ERROR) {

        if (GetRegKey(HKEY_CLASSES_ROOT, _T(".htm"), key) == ERROR_SUCCESS) {
            lstrcat(key, _T("\\shell\\open\\command"));

            if (GetRegKey(HKEY_CLASSES_ROOT,key,key) == ERROR_SUCCESS) {
                TCHAR *pos;
                pos = _tcsstr(key, _T("\"%1\""));
                if (pos == NULL) {                      // No quotes found
                    pos = strstr(key, _T("%1"));        // Check for %1, without
quotes
                    if (pos == NULL)                    // No parameter at all...
                        pos = key+lstrlen(key)-1;
                    else
                        *pos = '\0';                    // Remove the parameter
                }
                else
                    *pos = '\0';                        // Remove the parameter

                lstrcat(pos, _T(" "));
                lstrcat(pos, url);
                result = (HINSTANCE) WinExec(key,showcmd);
            }
        }
      }

    return result;
}
```

```
// Activate the link
void CHyperLink::FollowLink()
{
      int result = (int) GotoURL(m_strURL, SW_SHOW);
    if (result <= HINSTANCE_ERROR) {
        MessageBeep(MB_ICONEXCLAMATION);  // Unable to follow link
        ReportError(result);
    } else {
            // Mark link as visited and repaint window
            m_bVisited = TRUE;
            Invalidate();
      }
}
```

```
#include "stdafx.h"
#include "mci.h"

//////////////////////////////////////////////////////////////////////////////
// CMciDevice implementation
//

// Common Modes
const DWORD CMciDevice::ModeNotReady = MCI_MODE_NOT_READY;
const DWORD CMciDevice::ModePause    = MCI_MODE_PAUSE;
const DWORD CMciDevice::ModePlay     = MCI_MODE_PLAY;
const DWORD CMciDevice::ModeStop     = MCI_MODE_STOP;
const DWORD CMciDevice::ModeOpen     = MCI_MODE_OPEN;
const DWORD CMciDevice::ModeRecord   = MCI_MODE_RECORD;
const DWORD CMciDevice::ModeSeek     = MCI_MODE_SEEK;
// Common status
const DWORD CMciDevice::StatusReady              = MCI_STATUS_READY;
const DWORD CMciDevice::StatusMediaPresent       = MCI_STATUS_MEDIA_PRESENT;
const DWORD CMciDevice::StatusMode               = MCI_STATUS_MODE;
const DWORD CMciDevice::StatusNumberOfTracks = MCI_STATUS_NUMBER_OF_TRACKS;
// Common capabilites
const DWORD CMciDevice::GetdevcapsCanEject         = MCI_GETDEVCAPS_CAN_EJECT;
const DWORD CMciDevice::GetdevcapsCanPlay  = MCI_GETDEVCAPS_CAN_PLAY;
const DWORD CMciDevice::GetdevcapsCanRecord        = MCI_GETDEVCAPS_CAN_RECORD;
const DWORD CMciDevice::GetdevcapsCanSave  = MCI_GETDEVCAPS_CAN_SAVE;
const DWORD CMciDevice::GetdevcapsCompound         =
MCI_GETDEVCAPS_COMPOUND_DEVICE;
const DWORD CMciDevice::GetdevcapsDeviceType = MCI_GETDEVCAPS_DEVICE_TYPE;
const DWORD CMciDevice::GetdevcapsHasAudio         = MCI_GETDEVCAPS_HAS_AUDIO;
const DWORD CMciDevice::GetdevcapsHasVideo         = MCI_GETDEVCAPS_HAS_VIDEO;
const DWORD CMciDevice::GetdevcapsUsesFiles        = MCI_GETDEVCAPS_USES_FILES;

const DWORD CMciDevice::InfoProduct = MCI_INFO_PRODUCT;

const DWORD CMciDevice::DevtypeAnimation  = MCI_DEVTYPE_ANIMATION;
const DWORD CMciDevice::DevtypeCdaudio           = MCI_DEVTYPE_CD_AUDIO;
const DWORD CMciDevice::DevtypeDat               = MCI_DEVTYPE_DAT;
const DWORD CMciDevice::DevtypeDigitalvideo = MCI_DEVTYPE_DIGITAL_VIDEO;
const DWORD CMciDevice::DevtypeOther             = MCI_DEVTYPE_OTHER;
const DWORD CMciDevice::DevtypeOverlay           = MCI_DEVTYPE_OVERLAY;
const DWORD CMciDevice::DevtypeScanner           = MCI_DEVTYPE_SCANNER;
const DWORD CMciDevice::DevtypeSequencer  = MCI_DEVTYPE_SEQUENCER;
const DWORD CMciDevice::DevtypeVcr               = MCI_DEVTYPE_VCR;
const DWORD CMciDevice::DevtypeVideodisc  = MCI_DEVTYPE_VIDEODISC;
const DWORD CMciDevice::DevtypeWaveaudio  = MCI_DEVTYPE_WAVEFORM_AUDIO;

CMciDevice::CMciDevice() {
      m_wDeviceID = NULL;
      m_hMainWnd = NULL;
      m_bReportErrors = FALSE;
}
```

```
CMciDevice::~CMciDevice() {
}

// Open by device name (obtained by the registry or system.ini)
DWORD CMciDevice::Open(LPCSTR lpstrName, BOOL bShareable /*=FALSE*/)
{
        ASSERT(lpstrName != NULL);

        DWORD dwReturn;
        MCI_OPEN_PARMS mciOpenParms;

        // Open a device by specifying the device name.
        mciOpenParms.lpstrDeviceType = lpstrName;

        DWORD dwFlags = MCI_OPEN_TYPE|MCI_WAIT;
        if (bShareable) dwFlags |= MCI_OPEN_SHAREABLE;

        dwReturn = SendCommand(MCI_OPEN, dwFlags, (DWORD)(LPVOID) &mciOpenParms);

        if (dwReturn == 0) {
                // The device opened successfully; get the device ID.
                m_wDeviceID = mciOpenParms.wDeviceID;
        }

        return dwReturn;
}

// Open by device type
DWORD CMciDevice::Open(DWORD dwDeviceType, BOOL bShareable /*=FALSE*/)
{
        DWORD dwReturn;
        MCI_OPEN_PARMS mciOpenParms;

        // Opens a device by specifying a device-type constant.
        mciOpenParms.lpstrDeviceType = (LPCSTR) dwDeviceType;

        DWORD dwFlags = MCI_OPEN_TYPE|MCI_OPEN_TYPE_ID|MCI_WAIT;
        if (bShareable) dwFlags |= MCI_OPEN_SHAREABLE;

        dwReturn = SendCommand(MCI_OPEN, dwFlags,
                        (DWORD)(LPVOID) &mciOpenParms);

        if (dwReturn == 0) {
                // The device opened successfully; get the device ID.
                m_wDeviceID = mciOpenParms.wDeviceID;
        }

        return dwReturn;
}

// Closes the device
DWORD CMciDevice::Close() {
        MCI_GENERIC_PARMS mciGenericParms;
        mciGenericParms.dwCallback = (DWORD) m_hMainWnd;
        return SendCommand(MCI_CLOSE, 0, (DWORD) &mciGenericParms);
}
```

```
// Gets the current callback window
HWND CMciDevice::GetCallbackHwnd() const {
      return m_hMainWnd;
}

// Set the current callback window
void CMciDevice::SetCallbackWnd(CWnd *pWnd) {
      ASSERT(pWnd != NULL);
      m_hMainWnd = pWnd->GetSafeHwnd();
}

// Set the current callback window
void CMciDevice::SetCallbackWnd(HWND hWnd) {
      ASSERT(hWnd != NULL);
      m_hMainWnd = hWnd;
}

// Attaches the MCI device to a device already opened
void CMciDevice::Attach(UINT wDeviceID) {
      m_wDeviceID = wDeviceID;
}

// Gets the device ID
MCIDEVICEID CMciDevice::GetDeviceID() const {
      return m_wDeviceID;
}

// mciSendCommand with error handling
DWORD CMciDevice::SendCommand(UINT uMsg, DWORD fdwCommand, DWORD dwParam) {
      DWORD dwReturn;
      if (dwReturn = mciSendCommand(m_wDeviceID, uMsg, fdwCommand, dwParam)) {
            m_dwLastError = dwReturn;
            if (m_bReportErrors)
                  ShowError(dwReturn);
      }
      return dwReturn;
}

MCIERROR CMciDevice::GetLastError() const {
      return m_dwLastError;
}

// Generic MCI_GETDEVCAPS_ITEM command: good for all devices
DWORD CMciDevice::GetDevCaps(DWORD dwDevcaps, BOOL bItem /*=FALSE*/) {
      MCI_GETDEVCAPS_PARMS mciCapsParms;

      mciCapsParms.dwCallback = (DWORD) m_hMainWnd;
    mciCapsParms.dwReturn = 0;

      if (bItem) {
            mciCapsParms.dwItem = dwDevcaps;
            dwDevcaps |= MCI_GETDEVCAPS_ITEM;
      }

      SendCommand(MCI_GETDEVCAPS, dwDevcaps, (DWORD) &mciCapsParms);

      return mciCapsParms.dwReturn;
```

```
}

// Generic MCI_INFO command
DWORD CMciDevice::GetInfo(DWORD dwInfo, LPSTR lpstrReturn, DWORD dwRetSize) {
      MCI_INFO_PARMS mciInfoParms;

      mciInfoParms.dwCallback = (DWORD) m_hMainWnd;
    mciInfoParms.lpstrReturn = lpstrReturn;
    mciInfoParms.dwRetSize = dwRetSize;

      return SendCommand(MCI_INFO, dwInfo, (DWORD) &mciInfoParms);
}

// Set error report on/off
void CMciDevice::ReportErrors(BOOL bReport /*=TRUE*/) {
      m_bReportErrors = bReport;
}

// Uses mciGetErrorString to get a textual description of an MCI error.
// Displays the error description using MessageBox.
void CMciDevice::ShowError(DWORD dwError)
{
    char szErrorBuf[MAXERRORLENGTH];
    MessageBeep(MB_ICONEXCLAMATION);

      CWnd* pMainWnd = AfxGetApp()->m_pMainWnd;
      ASSERT(pMainWnd != NULL);
      HWND hMainWnd = pMainWnd->GetSafeHwnd();

    if(mciGetErrorString(dwError, (LPSTR) szErrorBuf, MAXERRORLENGTH)) {
            MessageBox(hMainWnd, szErrorBuf, _T("MCI Error"),
                          MB_ICONEXCLAMATION);
      }
    else
        MessageBox(hMainWnd, _T("Unknown Error"), _T("MCI Error"),
                          MB_ICONEXCLAMATION);
}

// Closes all MCI devices opened by the application.
// Waits until devices are closed before returning.
DWORD CMciDevice::CloseAll()
{
      DWORD dwReturn;

      if (dwReturn = mciSendCommand(MCI_ALL_DEVICE_ID, MCI_CLOSE, MCI_WAIT,
NULL))
            ShowError(dwReturn);
      return dwReturn;
}

DWORD CMciDevice::GetMode() {
      return GetStatus(StatusMode);
}

// Generic MCI_STATUS command
DWORD CMciDevice::GetStatus(DWORD dwItem) {
      MCI_STATUS_PARMS mciStatusParms;
```

```
        mciStatusParms.dwCallback = (DWORD) m_hMainWnd;
        mciStatusParms.dwItem = dwItem;
        mciStatusParms.dwReturn = 0;

        SendCommand(MCI_STATUS, MCI_STATUS_ITEM, (DWORD) &mciStatusParms);

        return mciStatusParms.dwReturn;
}
```

```
                                        // MCISample.cpp


#include "stdafx.h"
#include "MCISample.h"
#include "MCISampleDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CMCISampleApp

BEGIN_MESSAGE_MAP(CMCISampleApp, CWinApp)
	//{{AFX_MSG_MAP(CMCISampleApp)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG
	ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMCISampleApp construction

CMCISampleApp::CMCISampleApp()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CMCISampleApp object

CMCISampleApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CMCISampleApp initialization

BOOL CMCISampleApp::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a
shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	CMCISampleDlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

```
                                                        // MCISampleDlg.cpp


#include "stdafx.h"
#include "MCISample.h"
#include "MCISampleDlg.h"
#include "PlaySectionDlg.h"
#include "hyperlink.h"
#include <dbt.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	CHyperLink  m_mailLink;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
	virtual BOOL OnInitDialog();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	DDX_Control(pDX, IDC_MAIL_LINK, m_mailLink);
	//}}AFX_DATA_MAP
}
```

```
BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
      //{{AFX_MSG_MAP(CAboutDlg)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CAboutDlg::OnInitDialog()
{
      CDialog::OnInitDialog();

      m_mailLink.SetURL(_T("GOOD BYE"));

      return TRUE;  // return TRUE unless you set the focus to a control
                    // EXCEPTION: OCX Property Pages should return FALSE
}

/////////////////////////////////////////////////////////////////////////////
// CMCISampleDlg dialog

CMCISampleDlg::CMCISampleDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CMCISampleDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CMCISampleDlg)
      m_strTrack = _T("");
      //}}AFX_DATA_INIT
      // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
      m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMCISampleDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CMCISampleDlg)
      DDX_Control(pDX, IDC_EJECT, m_btnEject);
      DDX_Control(pDX, IDC_PLAY_SECTION, m_btnPlaySection);
      DDX_Control(pDX, IDC_PLAY, m_btnPlay);
      DDX_Control(pDX, IDC_STOP, m_btnStop);
      DDX_Control(pDX, IDC_PAUSE, m_btnPause);
      DDX_Control(pDX, IDC_STATUS, m_wndStatus);
      DDX_Control(pDX, IDC_TRACKSELECT, m_comboTrack);
      DDX_CBString(pDX, IDC_TRACKSELECT, m_strTrack);
      //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMCISampleDlg, CDialog)
      //{{AFX_MSG_MAP(CMCISampleDlg)
      ON_WM_SYSCOMMAND()
      ON_WM_PAINT()
      ON_WM_QUERYDRAGICON()
      ON_BN_CLICKED(IDC_PLAY, OnPlay)
      ON_BN_CLICKED(IDC_STOP, OnStop)
      ON_CBN_SELCHANGE(IDC_TRACKSELECT, OnSelchangeTrackselect)
      ON_WM_CLOSE()
      ON_BN_CLICKED(IDC_PAUSE, OnPause)
      ON_BN_CLICKED(IDC_EJECT, OnEject)
      ON_BN_CLICKED(IDC_PLAY_SECTION, OnPlaySection)
      //}}AFX_MSG_MAP
```

```
        ON_WM_DEVICECHANGE()
        ON_MESSAGE(MM_MCINOTIFY, OnMciNotify)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CMCISampleDlg message handlers

BOOL CMCISampleDlg::OnInitDialog()
{
        CDialog::OnInitDialog();


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

        CMenu* pSysMenu = GetSystemMenu(FALSE);
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }

        // 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

        // Set MCI error report mode
        //m_CdAudio.ReportErrors(TRUE);

        // Disable the Stop/Pause buttons
        m_btnStop.EnableWindow(FALSE);
        m_btnPause.EnableWindow(FALSE);

        // Open the CD audio  device
        if (m_CdAudio.Open()) {
                m_wndStatus.SetWindowText(_T("Device not ready!"));

                m_btnPlay.EnableWindow(FALSE);
                m_btnPlaySection.EnableWindow(FALSE);
                m_btnEject.EnableWindow(FALSE);
        }
        else {
                // Set the callback window for asynchronous operations
                m_CdAudio.SetCallbackWnd(this);
                m_wndStatus.SetWindowText(_T("Brave hearts CD player ready"));
        }

        if (!m_CdAudio.IsReady()) {
                m_btnPlay.EnableWindow(FALSE);
                m_btnPlaySection.EnableWindow(FALSE);
        }

        // Set time format for CD audio
```

```
	m_CdAudio.SetTimeFormat(CCdAudio::FormatTMSF);

	// Update dialog controls
	UpdateControls();

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

void CMCISampleDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
	if ((nID & 0xFFF0) == IDM_ABOUTBOX)
	{
		CAboutDlg dlgAbout;
		dlgAbout.DoModal();
	}
	else
	{
		CDialog::OnSysCommand(nID, lParam);
	}
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CMCISampleDlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CMCISampleDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
```

```
void CMCISampleDlg::OnPlay()
{
	UpdateData();
	// Convert track to number
	m_nTrackToPlay =
		(m_strTrack.GetAt(0)-'0')*10 +
		(m_strTrack.GetAt(1)-'0');

	// Get track lenght
	CMsf msf = m_CdAudio.GetTrackLength(m_nTrackToPlay);

	m_tmsfFrom = CTmsf(m_nTrackToPlay, 0, 0, 0);
	m_tmsfTo = CTmsf(m_nTrackToPlay,
		msf.GetMinute(), msf.GetSecond(), msf.GetFrame());

	// Play selected track
	if (!m_CdAudio.Play(m_tmsfFrom, m_tmsfTo)) {
		m_wndStatus.SetWindowText(_T("Playing..."));
		m_btnStop.EnableWindow();
		m_btnPause.EnableWindow();
	}

}

void CMCISampleDlg::OnPlaySection()
{
	CPlaySectionDlg dlg;

	if (dlg.DoModal() == IDOK) {
			// Convert track to number
		m_nTrackToPlay =
			(m_strTrack.GetAt(0)-'0')*10 +
			(m_strTrack.GetAt(1)-'0');

		m_tmsfFrom = CTmsf(m_nTrackToPlay,
			dlg.m_nMinuteFrom, dlg.m_nSecondFrom, dlg.m_nFrameFrom);
		m_tmsfTo = CTmsf(m_nTrackToPlay,
			dlg.m_nMinuteTo, dlg.m_nSecondTo, dlg.m_nFrameTo);

		if (!m_CdAudio.Play(m_tmsfFrom, m_tmsfTo)) {
			m_wndStatus.SetWindowText(_T("Playing..."));
			m_btnStop.EnableWindow();
			m_btnPause.EnableWindow();
		}
	}
}

void CMCISampleDlg::OnStop()
{
	if (!m_CdAudio.Stop()) {
		m_wndStatus.SetWindowText(_T("Stopped"));
		m_btnStop.EnableWindow(FALSE);
		m_btnPause.EnableWindow(FALSE);
		m_btnPause.SetWindowText(_T("&Pause"));
	}
}
```

```
void CMCISampleDlg::OnSelchangeTrackselect()
{
	UpdateData();
	// Convert track to number
	m_nTrackToPlay =
		(m_strTrack.GetAt(0)-'0')*10 +
		(m_strTrack.GetAt(1)-'0');
	// Seek to the selected track
	if (m_nTrackToPlay > 0)
		m_CdAudio.Seek(CTmsf(m_nTrackToPlay, 0, 0, 0));
}

void CMCISampleDlg::OnClose()
{
	// Show the About dialog
	CAboutDlg dlg;
	dlg.DoModal();

	// Stop and close the CD-Audio
	if (m_CdAudio.IsReady()) {
		m_CdAudio.Stop();
		m_CdAudio.Close();
	}

	CDialog::OnClose();
}

void CMCISampleDlg::OnPause()
{
	if (m_CdAudio.GetMode() == CCdAudio::ModePlay) {
		// Pause playback
		if (!m_CdAudio.Pause()) {
			m_wndStatus.SetWindowText(_T("Paused"));
			m_btnPause.SetWindowText(_T("&Resume"));
		}
	}
	else {
		// Resume playback from current position
		if (!m_CdAudio.Play(m_CdAudio.GetCurrentPos(), m_tmsfTo
			m_btnPause.SetWindowText(_T("&Pause"));
			m_wndStatus.SetWindowText(_T("Playing..."));

	}
}

void CMCISampleDlg::OnEject()
{
	// Open/Close the CD door. IsReady() returns FALSE if the
	// CD door is open
	m_CdAudio.OpenDoor(m_CdAudio.IsReady());
	m_btnPause.SetWindowText(_T("&Pause"));
	BOOL bReady = m_CdAudio.IsReady();
	EnableControls(bReady, bReady);
}

// Intercept a new CD-Audio insertion
BOOL CMCISampleDlg::OnDeviceChange( UINT nEventType, DWORD dwData
        DEV_BROADCAST_HDR *pdbch = (DEV_BROADCAST_HDR *) dwData;

        switch(nEventType) {
        case DBT_DEVICEARRIVAL:                                    // CD drawer was
closed
                if (pdbch->dbch_devicetype == DBT_DEVTYP_VOLUME) {
                        PDEV_BROADCAST_VOLUME pdbcv =
                                (PDEV_BROADCAST_VOLUME) dwData;
                        if (pdbcv->dbcv_flags == DBTF_MEDIA)
                        {
                                TCHAR szDrive[4];
                                // pdbcv->unitmask contains the drive bits
                                for (UINT i=0; !(pdbcv->dbcv_unitmask & (1<<i)); i++);
                                wsprintf(szDrive, _T("%c:\\"), 'A'+i);
                                if (GetDriveType(szDrive) == DRIVE_CDROM) {

                                        UpdateControls();
                                }
                        }
                        else {
                                // It is a network drive
                        }
                        break;
                }
        case DBT_DEVICEREMOVECOMPLETE:                             // CD drawer was
open
                EnableControls(FALSE, FALSE);
        break;
        }

        return TRUE;
}

void CMCISampleDlg::EnableControls(BOOL bEnablePlay,
                                   BOOL bEnableStop /* = TRUE*/)
{
        m_comboTrack.EnableWindow(bEnablePlay);
        m_btnPlay.EnableWindow(bEnablePlay);
        m_btnPlaySection.EnableWindow(bEnablePlay);
        m_btnStop.EnableWindow(bEnableStop);
        m_btnPause.EnableWindow(bEnableStop);
}

// Update the dialog controls
void CMCISampleDlg::UpdateControls() {
        // Get the number of playable tracks on the CD
        DWORD n = m_CdAudio.GetNumberOfTracks();

        m_comboTrack.ResetContent();
        CString s;
        CMsf msf;
        // Update the trackselect combobox
        for (UINT i=1; i<=n; i++) {
              msf = m_CdAudio.GetTrackLength(i);
              if (m_CdAudio.GetTrackType(i) == CCdAudio::TrackTypeAudio)
                      s.Format(_T("%02d     (%02d\':%02d\")"), i,
                              msf.GetMinute(), msf.GetSecond());
                        m_comboTrack.AddString(s);
                }

        }
        // Select first track
        m_comboTrack.SetCurSel(0);
        // if no playable track, disable Playback buttons
        BOOL bCanPlay = m_comboTrack.GetCount() != 0;

        EnableControls(bCanPlay, FALSE);
        UpdateData();
}

LONG CMCISampleDlg::OnMciNotify(UINT wFlags, LONG lDevId) {
        switch(wFlags) {
        case MCI_NOTIFY_SUCCESSFUL:
                m_wndStatus.SetWindowText(_T("Playback completed"));
                break;
        case MCI_NOTIFY_FAILURE:
                m_wndStatus.SetWindowText(_T("Playback interrupted!"));
                break;
        case MCI_NOTIFY_SUPERSEDED:
                m_wndStatus.SetWindowText(_T("Playback stopped!"));
                break;
        }
        return 0L;
}
```

```
                              // PlaySectionDlg.cpp


#include "stdafx.h"
#include "MCISample.h"
#include "PlaySectionDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CPlaySectionDlg dialog


CPlaySectionDlg::CPlaySectionDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CPlaySectionDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CPlaySectionDlg)
      m_nFrameFrom = 0;
      m_nFrameTo = 0;
      m_nMinuteFrom = 0;
      m_nMinuteTo = 0;
      m_nSecondFrom = 0;
      m_nSecondTo = 0;
      //}}AFX_DATA_INIT
}


void CPlaySectionDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CPlaySectionDlg)
      DDX_Text(pDX, IDC_FRAME_FROM, m_nFrameFrom);
      DDV_MinMaxByte(pDX, m_nFrameFrom, 0, 255);
      DDX_Text(pDX, IDC_FRAME_TO, m_nFrameTo);
      DDV_MinMaxByte(pDX, m_nFrameTo, 0, 255);
      DDX_Text(pDX, IDC_MINUTE_FROM, m_nMinuteFrom);
      DDV_MinMaxByte(pDX, m_nMinuteFrom, 0, 255);
      DDX_Text(pDX, IDC_MINUTE_TO, m_nMinuteTo);
      DDV_MinMaxByte(pDX, m_nMinuteTo, 0, 255);
      DDX_Text(pDX, IDC_SECOND_FROM, m_nSecondFrom);
      DDV_MinMaxByte(pDX, m_nSecondFrom, 0, 255);
      DDX_Text(pDX, IDC_SECOND_TO, m_nSecondTo);
      DDV_MinMaxByte(pDX, m_nSecondTo, 0, 255);
      //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CPlaySectionDlg, CDialog)
      //{{AFX_MSG_MAP(CPlaySectionDlg)
      //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CPlaySectionDlg message handlers

void CPlaySectionDlg::OnOK()
{
	// TODO: Add extra validation here

	CDialog::OnOK();
}
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

//#define VC_EXTRALEAN        // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>                 // MFC support for Windows 95 Common
Controls
#endif // _AFX_NO_AFXCMN_SUPPORT
```

//resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by MCISample.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_MCISAMPLE_DIALOG            102
#define IDR_MAINFRAME                   128
#define IDD_PLAYSECTION_DIALOG          129
#define IDC_PLAY                        1000
#define IDC_STOP                        1001
#define IDC_PAUSE                       1002
#define IDC_TRACKSELECT                 1003
#define IDC_EJECT                       1005
#define IDC_PLAY_SECTION                1006
#define IDC_MINUTE_FROM                 1007
#define IDC_SECOND_FROM                 1008
#define IDC_FRAME_FROM                  1009
#define IDC_MINUTE_TO                   1010
#define IDC_SECOND_TO                   1011
#define IDC_FRAME_TO                    1012
#define IDC_STATUS                      1013
#define IDC_MAIL_LINK                   1014

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        132
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1018
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

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

#define APSTUDIO_READONLY_SYMBOLS
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

/////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

/////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

/////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_ABOUTBOX DIALOGEX 0, 0, 168, 93
STYLE DS_SYSMODAL | DS_MODALFRAME | DS_SETFOREGROUND | DS_3DLOOK | WS_POPUP |
    WS_VISIBLE | WS_CAPTION
EXSTYLE WS_EX_STATICEDGE
CAPTION "BRAVE HEARTS CD-PLAYER"
FONT 8, "Arial Black"
BEGIN
    CONTROL         "PROJECT BY :-",IDC_STATIC,"Static",SS_SIMPLE|
                    SS_NOPREFIX | WS_GROUP,33,7,48,8
    LTEXT           "MURARI  M",IDC_STATIC,51,15,50,12
    DEFPUSHBUTTON   "OK",IDOK,128,7,32,14,WS_GROUP
    LTEXT           "THANK 'U'",IDC_MAIL_LINK,103,84,57,8
    ICON            IDR_MAINFRAME,IDC_STATIC,11,17,20,20
    LTEXT           "RAJAGOPALAN  S",IDC_STATIC,51,25,72,10
    LTEXT           "SARANYAN  U",IDC_STATIC,51,35,73,10
    LTEXT           "VISHAKAN  G",IDC_STATIC,51,46,74,10
    LTEXT           "GUIDED BY :-",IDC_STATIC,35,61,58,10
    LTEXT           "JAYASHREE L S",IDC_STATIC,52,72,81,10
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
END

#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 160
        TOPMARGIN, 7
        BOTTOMMARGIN, 92
    END
END
#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDS_ABOUTBOX            "&About Brave Hearts CD Player"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////


/////////////////////////////////////////////////////////////////////////////
// Italian (Italy) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ITA)
#ifdef _WIN32
LANGUAGE LANG_ITALIAN, SUBLANG_ITALIAN
#pragma code_page(1252)
#endif //_WIN32
```

```
/////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_MCISAMPLE_DIALOG DIALOGEX 0, 0, 123, 107
STYLE DS_MODALFRAME | DS_SETFOREGROUND | DS_3DLOOK | DS_CONTEXTHELP |
    WS_MINIMIZEBOX | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "CD-PLAYER"
FONT 8, "Arial Black"
BEGIN
    COMBOBOX        IDC_TRACKSELECT,7,15,109,72,CBS_DROPDOWNLIST | CBS_SORT |
                    WS_VSCROLL | WS_TABSTOP
    PUSHBUTTON      "&Play",IDC_PLAY,7,33,52,16
    PUSHBUTTON      "Play Section...",IDC_PLAY_SECTION,63,33,53,16
    PUSHBUTTON      "&Stop",IDC_STOP,7,53,53,15
    PUSHBUTTON      "P&ause",IDC_PAUSE,63,53,53,15
    PUSHBUTTON      "&Eject/Close",IDC_EJECT,7,73,109,13
    LTEXT           "Static",IDC_STATUS,7,90,109,10,SS_SUNKEN
    LTEXT           "Track",IDC_STATIC,7,6,34,8
END

IDD_PLAYSECTION_DIALOG DIALOGEX 100, 15, 107, 77
STYLE DS_SYSMODAL | DS_MODALFRAME | DS_3DLOOK | WS_POPUP | WS_VISIBLE |
    WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_CLIENTEDGE
CAPTION "PLAY SELECTION"
FONT 8, "Arial Black"
BEGIN
    EDITTEXT        IDC_MINUTE_FROM,26,16,18,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_SECOND_FROM,55,16,18,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_FRAME_FROM,82,16,18,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_MINUTE_TO,26,32,19,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_SECOND_TO,55,32,18,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_FRAME_TO,82,32,18,12,ES_AUTOHSCROLL
    DEFPUSHBUTTON   "&Cancel",IDCANCEL,7,55,42,15
    DEFPUSHBUTTON   "&Play",IDOK,58,55,42,15
    LTEXT           "M",IDC_STATIC,30,7,8,8,0,WS_EX_TRANSPARENT
    LTEXT           "S",IDC_STATIC,57,7,8,8
    LTEXT           "F",IDC_STATIC,84,7,8,8
    LTEXT           "From",IDC_STATIC,8,19,16,8
    LTEXT           "To",IDC_STATIC,8,34,13,8
END


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_MCISAMPLE_DIALOG, DIALOG
    BEGIN
```

```
        LEFTMARGIN, 7
        RIGHTMARGIN, 116
        TOPMARGIN, 7
        BOTTOMMARGIN, 100
    END

    IDD_PLAYSECTION_DIALOG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 100
        TOPMARGIN, 9
        BOTTOMMARGIN, 70
    END
END
#endif    // APSTUDIO_INVOKED


#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ITA)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 16, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif\r\n"
    "#include ""res\\MCISample.rc2""  // non-Microsoft Visual C++ edited
resources\r\n"
    "#include ""l.ita\\afxres.rc""          // Standard components\r\n"
    "#endif\0"
END

#endif    // APSTUDIO_INVOKED

/////////////////////////////////////////////////////////////////////////////
//
```

```
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME             ICON    DISCARDABLE     "idr_main.ico"
#endif    // Italian (Italy) resources
/////////////////////////////////////////////////////////////////////////////


#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ITA)
#ifdef _WIN32
LANGUAGE 16, 1
#pragma code_page(1252)
#endif
#include "res\MCISample.rc2"  // non-Microsoft Visual C++ edited resources
#include "l.ita\afxres.rc"          // Standard components
#endif
/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

[illegible]

```
                    // PlaySectionDlg.h


/////////////////////////////////////////////////////////////////////////////
// CPlaySectionDlg dialog

class CPlaySectionDlg : public CDialog
{
// Construction
public:
      CPlaySectionDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
      //{{AFX_DATA(CPlaySectionDlg)
      enum { IDD = IDD_PLAYSECTION_DIALOG };
      BYTE  m_nFrameFrom;
      BYTE  m_nFrameTo;
      BYTE  m_nMinuteFrom;
      BYTE  m_nMinuteTo;
      BYTE  m_nSecondFrom;
      BYTE  m_nSecondTo;
      //}}AFX_DATA


// Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CPlaySectionDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:

      // Generated message map functions
      //{{AFX_MSG(CPlaySectionDlg)
      virtual void OnOK();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};
```

```
        CCdAudio m_CdAudio;
        BYTE m_nTrackToPlay;

        void UpdateControls();
        void EnableControls(BOOL bEnablePlay, BOOL bEnableStop = TRUE);
};
```

```
                    // MCISampleDlg.h


// CMCISampleDlg dialog

#include "CdAudio.h"

class CMCISampleDlg : public CDialog
{
// Construction
public:
      CMCISampleDlg(CWnd* pParent = NULL);      // standard constructor

// Dialog Data
      //{{AFX_DATA(CMCISampleDlg)
      enum { IDD = IDD_MCISAMPLE_DIALOG };
      CButton     m_btnEject;
      CButton     m_btnPlaySection;
      CButton     m_btnPlay;
      CButton     m_btnStop;
      CButton     m_btnPause;
      CStatic     m_wndStatus;
      CComboBox   m_comboTrack;
      CString     m_strTrack;
      //}}AFX_DATA

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CMCISampleDlg)
      protected:
      virtual void DoDataExchange(CDataExchange* pDX);      // DDX/DDV support
      //}}AFX_VIRTUAL

// Implementation
protected:
      HICON m_hIcon;

      // Generated message map functions

      virtual BOOL OnInitDialog();
      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
      afx_msg void OnPaint();
      afx_msg HCURSOR OnQueryDragIcon();
      afx_msg void OnPlay();
      afx_msg void OnStop();
      afx_msg void OnSelchangeTrackselect();
      afx_msg void OnClose();
      afx_msg void OnPause();
      afx_msg void OnEject();
      afx_msg void OnPlaySection();
      //}}AFX_MSG
      afx_msg BOOL OnDeviceChange( UINT nEventType, DWORD dwData );
      afx_msg LONG OnMciNotify(UINT wParam, LONG lDevId);
      DECLARE_MESSAGE_MAP()

      CTmsf m_tmsfFrom;
      CTmsf m_tmsfTo;
```

```
                    // MCISample.h


#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
// CMCISampleApp:
// See MCISample.cpp for the implementation of this class
//

class CMCISampleApp : public CWinApp
{
public:
	CMCISampleApp();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CMCISampleApp)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CMCISampleApp)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////
```

```
        // Attaches the device to a MCI device already opened
        inline void  Attach(UINT wDeviceID);

        // Gets device ID
        MCIDEVICEID GetDeviceID() const;

        // Closes all open MCI devices: wait until devices are closed
        static  DWORD CloseAll();

        // Retrieves a device static information
        DWORD GetDevCaps(DWORD dwDevcaps, BOOL bItem = FALSE);

        // Retrieves a device status item
        virtual DWORD GetStatus(DWORD dwStatusItem);

        // Retrieves the device mode
        virtual DWORD GetMode();

        // Retrieves a string information from the device
        virtual DWORD GetInfo(DWORD dwInfo, LPSTR lpstrReturn, DWORD dwRetSize);

        // Get/Set Callback Window
        HWND GetCallbackHwnd() const;
        void SetCallbackWnd(CWnd* pWnd);
        void SetCallbackWnd(HWND hWnd);

        // Sets MCI error report on/off
        void ReportErrors(BOOL bReport = TRUE);

protected:
        // mciSendCommand with error handling
        DWORD SendCommand(UINT uMsg, DWORD fdwCommand, DWORD dwParam);

        // MCI error report
        static void ShowError(DWORD dwError);
        MCIERROR GetLastError() const;

// Data members
protected:
        MCIDEVICEID m_wDeviceID;                                  // The device ID
        HWND        m_hMainWnd;                                   // The callback
window handle

private:
        MCIERROR    m_dwLastError;                                // The last error
code
        BOOL        m_bReportErrors;                          // Report MCI errors?
};

#endif // !defined(AFX_MCI_H_098R22G3234 23453 235124 123A44HC134.. INCLUDED )
```

```
};

//////////////////////////////////////////////////////////////////////////////
// The CMciDevice class is the base class for all MCI devices
//
class CMciDevice {
public:
	// Common modes
	static const DWORD ModeNotReady;
	static const DWORD ModePause;
	static const DWORD ModePlay;
	static const DWORD ModeStop;
	static const DWORD ModeOpen;
	static const DWORD ModeRecord;
	static const DWORD ModeSeek;
	// Common status items
	static const DWORD StatusReady;
	static const DWORD StatusMediaPresent;
	static const DWORD StatusMode;
	static const DWORD StatusNumberOfTracks;
	// Common capabilites
	static const DWORD GetdevcapsCanEject;
	static const DWORD GetdevcapsCanPlay;
	static const DWORD GetdevcapsCanRecord;
	static const DWORD GetdevcapsCanSave;
	static const DWORD GetdevcapsCompound;
	static const DWORD GetdevcapsDeviceType;
	static const DWORD GetdevcapsHasAudio;
	static const DWORD GetdevcapsHasVideo;
	static const DWORD GetdevcapsUsesFiles;

	// Device Info
	static const DWORD InfoProduct;

	// Device types
	static const DWORD DevtypeAnimation;
	static const DWORD DevtypeCdaudio;
	static const DWORD DevtypeDat;
	static const DWORD DevtypeDigitalvideo;
	static const DWORD DevtypeOther;
	static const DWORD DevtypeOverlay;
	static const DWORD DevtypeScanner;
	static const DWORD DevtypeSequencer;
	static const DWORD DevtypeVcr;
	static const DWORD DevtypeVideodisc;
	static const DWORD DevtypeWaveaudio;



	// Construction/Destruction
	CMciDevice();
	~CMciDevice();

	// Open/Close: All derived classes must implement them
	virtual DWORD Open(DWORD dwDeviceType, BOOL bShareable = FALSE);
	virtual DWORD Open(LPCSTR lpstrName, BOOL bShareable = FALSE);
	virtual DWORD Close();
```

//mci.h

```
#if !defined(AFX_MCI_H_098R22G3234_23453_235124_123A44HQ13451_INCLUDED
#define AFX_MCI_H_098R22G3234_23453_235124_123A44HQ13451_INCLUDED

#include <mmsystem.h>

class CMsf {
public:
	CMsf() {
		m_dwMsf = 0;
	}

	CMsf(DWORD dwMsf) {
		m_dwMsf = dwMsf;
	}

	CMsf(BYTE minute, BYTE second, BYTE frame) {
		m_dwMsf = MCI_MAKE_MSF(minute, second, frame);
	}

	operator DWORD() const {return m_dwMsf;}

	BYTE GetMinute() const { return MCI_MSF_MINUTE(m_dwMsf); }
	BYTE GetSecond() const { return MCI_MSF_SECOND(m_dwMsf); }
	BYTE GetFrame() const { return MCI_MSF_FRAME(m_dwMsf); }

protected:
	DWORD m_dwMsf;
};

class CTmsf {
public:
	CTmsf() {
		m_dwTmsf = 0;
	}

	CTmsf(DWORD dwTmsf) {
		m_dwTmsf = dwTmsf;
	}

	CTmsf(BYTE track, BYTE minute, BYTE second, BYTE frame) {
		m_dwTmsf = MCI_MAKE_TMSF(track, minute, second, frame);
	}

	operator DWORD() const {return m_dwTmsf;}

	BYTE GetTrack() const { return MCI_TMSF_TRACK(m_dwTmsf); }
	BYTE GetMinute() const { return MCI_TMSF_MINUTE(m_dwTmsf); }
	BYTE GetSecond() const { return MCI_TMSF_SECOND(m_dwTmsf); }
	BYTE GetFrame() const { return MCI_TMSF_FRAME(m_dwTmsf); }

protected:
	DWORD m_dwTmsf;
	afx_msg HBRUSH CtlColor(CDC* pDC, UINT nCtlColor);
	afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
	afx_msg void OnMouseMove(UINT nFlags, CPoint point);
	afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
	afx_msg void OnSetFocus(CWnd* pOldWnd);
	afx_msg void OnKillFocus(CWnd* pNewWnd);
	afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
	afx_msg UINT OnNcHitTest(CPoint point);
	afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}

#endif // !defined(AFX_HYPERLINK_H_04F7323B01 023500 0204251998_ENG_INCLUDED_
    CString GetURL() const;

        DWORD GetLinkStyle() const;
        BOOL ModifyLinkStyle(DWORD dwRemove, DWORD dwAdd, BOOL bApply=TRUE);

        void SetWindowText(LPCTSTR lpszText);
        void SetFont(CFont *pFont);

        BOOL IsVisited() const;
        void SetVisited(BOOL bVisited = TRUE);

        // Use this if you want to subclass and also set different URL
        BOOL SubclassDlgItem(UINT nID, CWnd* pParent, LPCTSTR lpszURL=NULL) {
                m_strURL = lpszURL;
                return CStatic::SubclassDlgItem(nID, pParent);
        }

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CHyperLink)
        public:
        virtual BOOL PreTranslateMessage(MSG* pMsg);
        protected:
        virtual void PreSubclassWindow();
        //}}AFX_VIRTUAL

// Implementation
protected:
        static void SetDefaultCursor();
        static LONG GetRegKey(HKEY key, LPCTSTR subkey, LPTSTR retdata);
        static void ReportError(int nError);
        static HINSTANCE GotoURL(LPCTSTR url, int showcmd);

        void AdjustWindow();
        void FollowLink();
        inline void SwitchUnderline();

// Protected attributes
protected:
        static COLORREF g_crLinkColor;                // Link normal color
        static COLORREF g_crActiveColor;      // Link active color
        static COLORREF g_crVisitedColor;     // Link visited color
        static COLORREF g_crHoverColor;               // Hover color
        static HCURSOR  g_hLinkCursor;                // Hyperlink mouse cursor

        BOOL    m_bLinkActive;                        // Is the link active?
        BOOL       m_bOverControl;                    // Is cursor over control?
        BOOL    m_bVisited;                           // Has link been visited?
        DWORD   m_dwStyle;                            // Link styles
        CString    m_strURL;                                  // Hyperlink URL string
        CFont      m_Font;                            // Underlined font (if required)

        CToolTipCtrl m_ToolTip;                       // The link tooltip

        // Generated message map functions
protected:
        //{{AFX_MSG(CHyperLink)
```

//hyperlink.h

```
#if !defined(AFX_HYPERLINK_H_C4ET323B01_023500_0204251998_ENG_INCLUDED_)
#define AFX_HYPERLINK_H_04ET323B01_023500_0204251998_ENG_INCLUDED

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

// Structure used to get/set hyperlink colors
typedef struct tagHYPERLINKCOLORS {
      COLORREF    crLink;
      COLORREF    crActive;
      COLORREF    crVisited;
      COLORREF    crHover;
} HYPERLINKCOLORS;


/////////////////////////////////////////////////////////////////////////////
// CHyperLink window

class CHyperLink : public CStatic
{
      DECLARE_DYNAMIC(CHyperLink)

public:
// Link styles
      static const DWORD StyleUnderline;
      static const DWORD StyleUseHover;
      static const DWORD StyleAutoSize;
      static const DWORD StyleDownClick;
      static const DWORD StyleGetFocusOnClick;
      static const DWORD StyleNoHandCursor;
      static const DWORD StyleNoActiveColor;

// Construction/destruction
      CHyperLink();
      virtual ~CHyperLink();

// Attributes
public:

// Operations
public:
      static void GetColors(HYPERLINKCOLORS& linkColors);

      static HCURSOR GetLinkCursor();
      static void SetLinkCursor(HCURSOR hCursor);

    static void SetColors(COLORREF crLinkColor, COLORREF crActiveColor,
                          COLORREF crVisitedColor, COLORREF crHoverCo[illegible] -1);
    static void SetColors(HYPERLINKCOLORS& colors);

      void SetURL(CString strURL);
      DWORD GetStartPos();
      BOOL IsReady();

      // Track info
      DWORD GetTrackPos(DWORD dwTrack);
      DWORD GetTrackLength(DWORD dwTrack);
      DWORD GetTrackType(DWORD dwTrack);

      // Get/Set the time format
      DWORD GetTimeFormat();
      DWORD SetTimeFormat(DWORD dwTimeFormat);

protected:
      DWORD Seek(DWORD dwTo, DWORD dwFlags, BOOL bAsync);
      DWORD Set(DWORD dwWhat);
      DWORD GetDevCapsItem(DWORD dwItem);
      DWORD GetTrackInfo(DWORD dwTrack, DWORD dwItem);
};

#endif // !defined(AFX_MCI_H_0S1R45G3534_142DS_215344_123444DGG345__INCLUDED_)
```

//CdAudio.h

```
#if !defined(AFX_MCI_H_0S1R45G3534_142DS_215344_123444DGG3451_INCLUDED_
#define AFX_MCI_H_0S1R45G3534_142DS_215344_123444DGG3451_INCLUDED_

#include "mci.h"

/////////////////////////////////////////////////////////////////////////////
// CCdAudio
//

class CCdAudio : public CMciDevice {

public:
        // Specific time formats
        static const DWORD FormatMilliseconds;
        static const DWORD FormatMSF;
        static const DWORD FormatTMSF;

        // Specific status
        static const DWORD StatusCurrentTrack;
        static const DWORD StatusLength;
        static const DWORD StatusPosition;
        static const DWORD StatusStart;
        static const DWORD TrackTypeAudio;
        static const DWORD TrackTypeOther;

        // Specific info
        static const DWORD InfoProduct;
        static const DWORD InfoMediaIdentity;
        static const DWORD InfoMediaUPC;

        // Open
        DWORD Open(BOOL bShareable = FALSE);

        // Playback/Stop/Pause
        DWORD PlayTrack(BYTE bTrack, BOOL bAsync = TRUE);
        DWORD Play(DWORD dwFrom = 0L, DWORD dwTo = 0L, BOOL bAsync = TRUE);
        DWORD Stop();
        DWORD Pause();

        // Open/Close CD door
        DWORD OpenDoor(BOOL bOpenDoor /*=TRUE*/);

        // Seek
        DWORD Seek(DWORD dwTo, BOOL bAsync = FALSE);
        DWORD SeekToStart(BOOL bAsync = FALSE);
        DWORD SeekToEnd(BOOL bAsync = FALSE);

        // Device status important items
        DWORD GetMediaLength(DWORD dwTrack);
        DWORD GetNumberOfTracks();
        DWORD GetCurrentTrack();
        DWORD GetCurrentPos();
```

```
// stdafx.cpp : source file that includes just the standard includes
//     MCISample.pch will be the pre-compiled header
//     stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```