ENEE408G Multimedia Signal Processing
Multimedia Programming Manual

Using Microsoft DirectX and Speech SDK

Department of Electrical and Computer Engineering
University of Maryland at College Park

Last Updated: January 21, 2003
# Contents

## Part I. Microsoft Visual Studio .NET
1. Introduction to .Net Framework 2

## Part II. DirectX and Component Object Model (COM)
1. Introduction to DirectX 4
2. Component Object Model (COM) 4

## Part III. DirectShow
1. Introduction to DirectShow 8
2. Architectures and Terminologies of DirectShow 8
3. Demo Project – 1: SimpleDirectShow 11
4. Demo Project – 2: MultimediaViewDS 25

## Part IV. Microsoft Speech SDK
1. Introduction to Microsoft Speech SDK 44
2. Demo Project: Text-To-Speech 45

## Part V. Further References
58
Part I. Microsoft Visual Studio .NET

1. Introduction to .Net Framework

The .NET is Microsoft’s new vision for combining the Internet and the Web in the development, engineering and use of software. The major .NET strategy is its independence from a specific language or platform. Developers can create a .Net application in any .NET-compatible language.

The Microsoft .NET Framework is at the heart of the .NET strategy. The key to the framework is common language runtime (CLR). Programs are compiled into machine-specific instructions in two steps. The first step is to compile this program into Microsoft Intermediate Language (MSIL), which defines instructions for the CLR. Code converted into MSIL from other languages and sources can be woven together by the CLR. The MSIL code cannot run directly on any computer. We need just-in-time compiler (JITer) to read and produce actual machine code. This provides .NET with a certain amount of platform independence. We can represent the whole procedures in the following figure.

![Source Code: Visual Basic, Visual C++, C#, COBOL, etc.](image)

![Microsoft Intermediate Language (MSIL)](image)

![Platform-specific code](image)

**Development Tool, such as Visual Studio .Net**

**Just-in-time (JIT) compiler**

---


Visual Studio .Net is Microsoft’s integrated development environment for creating, documenting, running and debugging programs written in a variety of .NET programming languages. Visual Studio .Net organizes programs into projects and solutions. A project is a group of related files, such as C++ code, image and documentation. A solution is a group of projects that represent a complete application, or a set of related applications. Each project in the solution may perform a different task.

Visual Studio .NET also offers editing tools for manipulating several types of files. Figure I-2 shows the IDE. The upper-left window is the Solution Explorer/Class View/Resource View. This window lists all the files, classes, and resources (including the Graph User Interface, GUI) in the solution. The upper-right window is the Toolbox window if you are currently working GUI. The Toolbox contains reusable software components (Controls) that can be used to customize applications. You can drag controls and drop on your own GUI. The lower-left window is the Properties Window, which
allow manipulation of the properties for a form or control. Properties specify information about a control, such as size, color and position. Each control has its own set of properties. The bottom of the Properties Window contains a description of the selected property. The window at the bottom is the Output Window and it will display messages that result from compiling and linking your program.

Figure I-2
Part II. DirectX and Component Object Model (COM)

1. Introduction to DirectX

Microsoft DirectX is a set of low-level application programming interfaces (APIs) for creating games and other high-performance multimedia applications. It includes support for two-dimensional (2-D) and three-dimensional (3-D) graphics, sound effects and music, input devices, and networked applications such as multiplayer games.

Microsoft DirectX 9.0 is made up of the following components.

- **DirectX Graphics** combines the Microsoft DirectDraw and Microsoft Direct3D components of previous DirectX versions into a single application programming interface (API) that you can use for all graphics programming. The component includes the Direct3D extensions (D3DX) utility library, which simplifies many graphics programming tasks.
- Microsoft **DirectInput** provides support for a variety of input devices, including full support for force-feedback technology.
- Microsoft **DirectPlay** provides support for multiplayer networked games.
- Microsoft **DirectSound** can be used in the development of high-performance audio applications that play and capture waveform audio.
- Microsoft **DirectMusic** provides a complete solution for both musical and non-musical soundtracks based on waveforms, MIDI sounds, or dynamic content authored in DirectMusic Producer.
- Microsoft **DirectShow** provides for high-quality capture and playback of multimedia streams.
- **DirectSetup** is a simple API that provides one-call installation of the DirectX components.
- **DirectX Media Objects** (DMO) provide support for writing and using data-streaming objects, including video and audio encoders, decoders, and effects.

2. Component Object Model (COM)

The Component Object Model (COM) is an object-oriented programming model used by numerous applications, including Microsoft DirectX. COM objects are basically black boxes that can be used by applications to perform one or more tasks. They are most commonly implemented as a dynamic-link library (DLL). Like a conventional DLL, COM objects expose methods that your application can call to perform any of the supported tasks. A COM object's public methods are grouped into one or more interfaces. To use a method, you must create the object and obtain the appropriate interface from the object. An interface typically contains a related set of methods that provide access to a

---

2 For further information, please refer DirectX 9.0 (C++) > Introducing DirectX 9.0 > Using the DirectX SDK > Programming DirectX with C/C++ > Using COM (in DirectX 9.0 documentation).
particular feature of the object. Here we use a simple example with DirectShow to demonstrate how to use COM.

**Step-1 Add New Project:** Open Visual Studio .NET and create a new project from *File → New → Project.*

**Step-2 Add additional dependencies:** Click *Project → Properties* from menu bar. You will see a property pages, choose *Linker - Input* in the left column and then click “*Additional Dependencies*” entry in the right column. An “*Additional Dependencies*” dialog box will show up. Key in "*Strmiids.lib*" and "*Quartz.lib*" and press *OK.*

**Step-3 Editing Codes:**

```c
#include <dshow.h>
#include <stdio.h>

void main(void)
{
    // ********** declaration **********
    IGraphBuilder *pGraph = NULL;
    IMediaControl *pControl = NULL;
    IMediaEvent   *pEvent = NULL;

    // ********** Initialize the COM library. **********
    HRESULT hr = CoInitialize(NULL);

    // ********** Check whether it is successful or not using FAILED. **********
    if (FAILED(hr))
    {
        printf("ERROR - Could not initialize COM library");
        return;
    }

    // ******** Create the filter graph manager and query for interfaces. ********
    hr = CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER, IID_IGraphBuilder, (void **)&pGraph);
    if (FAILED(hr))
    {
        printf("ERROR - Could not create the Filter Graph Manager.");
        return;
    }

    // query interface ********
    hr = pGraph->QueryInterface(IID_IMediaControl, (void **)&pControl);
    hr = pGraph->QueryInterface(IID_IMediaEvent, (void **)&pEvent);

    // ## Build the graph. IMPORTANT: Change this string to a file on your system.
    hr = pGraph->RenderFile(L"C:\DXSDK\Samples\Media\piano.mp3", NULL);
    if (SUCCEEDED(hr))
    {
        // ***** Run the graph.
        hr = pControl->Run();
        if (SUCCEEDED(hr))
        {
            // Wait for completion. ********
            long evCode;
            pEvent->WaitForCompletion(INFINITE, &evCode);
        }
    }
}
```

EEN408G Spring 2003 (Update 01/21/2003)
Multimedia Programming Manual
// Note: Do not use INFINITE in a real application, because it
// can block indefinitely.
}
}

// release the interface pointers and close the COM library
pControl->Release();
pEvent->Release();
pGraph->Release();
CoUninitialize();

Explanation:

1: Declaration of COM interfaces.

2: Call CoInitialize to initialize the COM library:

All COM methods return a 32-bit integer called an HRESULT. With most methods,
the HRESULT is essentially a structure that contains two primary pieces of
information:
- Whether the method succeeded or failed.
- More detailed information about the outcome of the operation supported by the
  method.
We may be interested only in whether the method succeeded or failed. A robust way
to test whether an HRESULT value indicates success or failure is to pass the value to
the one of the following macros, defined in Winerror.h.
- The SUCCEEDED macro returns TRUE for a success code and FALSE for a
  failure code.
- The FAILED macro returns TRUE for a failure code and FALSE for a success
code.

3: Call CoCreateInstance to create the Filter Graph Manager. As shown, the class
identifier (CLSID) is CLSID_FilterGraph. The Filter Graph Manager is provided by
an in-process DLL, so the execution context is CLSCTX_INPROC_SERVER. The
call to CoCreateInstance returns the IGraphBuilder interface, which mostly contains
methods for building the filter graph.

4: Two other interfaces are needed for this example:
- IMediaControl controls streaming. It contains methods for stopping and starting
  the graph.
- IMediaEvent has methods for getting events from the Filter Graph Manager. In
  this example, the interface is used to wait for playback to complete.
Both of these interfaces are exposed by the Filter Graph Manager. Use the returned
IGraphBuilder pointer to query for them:

5: The IGraphBuilder:RenderFile method builds a filter graph that can play the
specified file. The first parameter is the file name, represented as a wide character (2-
byte) string. The second parameter is reserved and must equal NULL.
6: This method can fail if the specified file does not exist, or the file format is not recognized. Assuming that the method succeeds, however, the filter graph is now ready for playback. To run the graph, call the `IMediaControl::Run` method:

7: When the filter graph runs, data moves through the filters and is rendered as video and audio. Playback occurs on a separate thread. You can wait for playback to complete by calling the `IMediaEvent::WaitForCompletion` method. This method blocks until the file is done playing, or until the specified time-out interval elapses. The value INFINITE means the application blocks indefinitely until the file is done playing.

8: When the application is finished, release the interface pointers and close the COM library.
Part III. DirectShow

1. Introduction to DirectShow

DirectShow is Microsoft’s latest and most powerful multimedia application tool. The major design objective of DirectShow is for used to develop dynamic multimedia applications without concerning the internals of the media streaming process or specifics of media formats. DirectShow encapsulates the issues of file format, stream synchronization, and hardware implementation. Therefore, it allows developers to concentrate on the functionality and robustness of an application.

DirectShow achieves its universality by introducing a modular architecture based entirely on the Component Object Model (COM) similar to the rest of the DirectX family. This allows DirectShow or a DirectShow application to seamlessly integrate the models necessary to play back or capture a particular media type or source.

DirectShow supports file types as follows. Windows Media™ Audio (WMA), Windows Media™ Video (WMV), Advanced Systems Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), QuickTime (version 2 and lower), WAV, AIFF, AU, SND, and MIDI. The supported compression format is Windows Media Video, ISO MPEG-4 video version 1.0, Microsoft MPEG-4 version 3, Sipro Labs ACELP, Windows Media Audio, MPEG Audio Layer-3 (MP3), Digital Video (DV), MPEG-1, MJPEG, and Cinepak.

2. Architectures and Terminologies of DirectShow

The following diagram shows the relationship between an application, the DirectShow components, and some of the hardware and software components that DirectShow supports.

The DirectShow architecture. (Adopted from DirectX9.0 documentation)
(a) Filters

Filters are components that manipulate the bits of a multimedia stream. Filters can be grouped into three categories:

1. Source Filters
   A source filter introduces data into the graph. The data might come from a file, a network, a device, such as video camera or television card, or even a region on the video screen.

2. Transform Filters
   A transform filter takes an input stream, processes the data, and creates an output stream. Encoders/decoders, compressors/de-compressors, and stream splitters are examples of transform filters.

3. Renderer Filters
   Renderer filters sit at the end of the chain. They receive data and present it to the user. For example, a video renderer draws video frames on the display; an audio renderer sends audio data to the sound card; and a file-writer filter writes data to a file.

(b) Graphs

A filter graph is formed when filters are connected in succession to render a particular media source. A graph typically contains at least one of each of the three types of filters listed above.

(c) Pins

Pins are objects associated with a filter that handle the transfer of a multimedia stream between filters. There are two types of pin, input or output. The input pin of one filter connects to the output pin of the filter up the stream. The output pin of that filter connects to the input pint of the filter down the stream.

(d) Filter Graph Manager

The Filter Graph Manager is an object that managed the creation of a filter graph. The Filter Graph Manager instantiated the filters that are to be added to a graph. To initiate the connection process, it informs the filters that they are to be connected. In doing so, the graph filter notifies the pins of two connecting filters that a connection has been requested. Once a connection has been negotiated, the Filter Graph Manager requests the creation of an allocator on the downstream filter. The Filter Graph Manager is also responsible for delivering event notifications triggered by filters in a graph to other filters, the calling application, and the streaming process. The Filter Graph Manager is usually run by an application. However, pointers to its interfaces are available to filters and provide filters for most of the Filter Graph Manager functionality available to an application. This means that filters themselves can request, for example, that a stream be stopped.

(e) Allocators
After filters have negotiated a connection they establish a common allocator. An allocator is an object, usually created by an input pin, that creates a buffer suited for the data type being negotiated between pins. This buffer is used to queue up media samples and feeds one or more of them when a filter is ready to process the next batch of media samples.

In the rest of this part, we use two examples to demonstrate how to use DirectShow with Microsoft Visual Studio .Net.
3. Demo Project – 1: A Simple DirectShow Example with GUI
In this demo project, we will design a simple application using DirectShow and learn
(1) how to initialize and terminate DirectShow object and interface.
(2) how to playback, pause, and stop a streaming media

(a) Create a new project by File \(\rightarrow\) New \(\rightarrow\) Project from Visual Studio .NET menu bar.

(1) Select “Visual C++ Projects” on the Project Types column.
(2) Select “MFC Application” on the Templates column.
(3) Key in “SimpleDirectShow” on Name edit box and select a Location for this project.
(4) Click “OK” and a MFC Application Wizard window will pop up.

(b) MFC Application Wizard

(1) This tap shows the overview for this project. Since we are going to design a
dialog-based application, we need to change it. Click “Application Type” on the
left blue column.

(2) Choose “Dialog Based” on the “Application Type”, check “MFC standard” on
Project style, “MFC in a shared DLL” on the Use of MFC, and pick “English
(United States)” on the Resource language list box. Thus, we finish the decision of application type. Next, we design the user interface by clicking “User Interface Features” on the left blue column.

(3) Uncheck every item in the Main frame styles. Type “SimpleDirectShow” in the Dialog title edit box. Next, click on “Advanced Features” on the left blue column.

(4) We do not need to change the advanced features. Let’s move to next step by clicking on “Generated Classes” on the left blue column.
(5) This tap displays the generated classes and the base classes used in our application. We do not need to change the classes in this demo project. Click “Finish” to finish the initial setting.

(b) Design Dialog Box

(1) Overview: The following figure shows the IDE that we will use in this demo project. Click the Resource View tap and open the Dialog folder. You will see `IDD_SIMPLEDIRECTSHOW_DIALOG`. Click it and you will see another dialog box with text “TODO: Place dialog controls here” shown in the Edit Window. We will design our own GUI on this dialog box. The small window to the right of this dialog box is the “Controls Toolbar”, which provides control components. We will use this toolbar to add controls.
(2) Edit the Dialog Box. Delete the “TODO: Place dialog controls here” and the “OK” button by clicking them and pressing “Delete” key on the keyboard. Drag and drop four buttons from Controls toolbar to this dialog box. Rearrange those buttons and adjust the width and height of this dialog box as the following figure.

![Dialog Box with Buttons](image)

(3) Edit the properties, add member variables, and event handlers of buttons. Let’s edit Button1 first.
(i) Right click on Button1 and choose “Properties”. The Properties window will be on focus. Change “Caption” to “Open Files” and “ID” to “IDC_BUTTON_openfile”.

![Property Window](image)

(ii) Add member variables of buttons. Right click on Button1 and choose “Add Variable”. An “Add Member Variable Wizard” dialog box will show up.

![Add Member Variable Wizard](image)

Type “m_button_openfile” in the “variable name”. Press “Finish”.

EEN408G Spring 2003 (Update 01/21/2003)
Multimedia Programming Manual 14
(You will direct to Edit Window. To return to the dialog editor, you can press the “Resource View window” and click “IDD_SIMPLEDIRECSHOW_DIALOG” under SimpleDirectShow.rc → Dialog.)

(iii) Add event handler. Right click on Button1 and choose “Add Event Handler...”. An “Event Handler Wizard” dialog box will show up. Choose “BN_CLICKED” in the Message types. Press “Add and Edit”. Thus, we finish the design for Button1. Repeat the same procedures (i)–(iii) for Button2, Button3, and Button4 according to the following table.

<table>
<thead>
<tr>
<th>Button</th>
<th>Caption</th>
<th>ID</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button2</td>
<td>Play</td>
<td>IDC.BUTTON_play</td>
<td>m_button_play</td>
</tr>
<tr>
<td>Button3</td>
<td>Pause</td>
<td>IDC.BUTTON_pause</td>
<td>m_button_pause</td>
</tr>
<tr>
<td>Button4</td>
<td>Stop</td>
<td>IDC.BUTTON_stop</td>
<td>m_button_stop</td>
</tr>
</tbody>
</table>

Finally, change the caption of “Cancel” button to “Exit”. We finish the interface design. The following figure shows the GUI.

(c) Include DirectShow Libraries

Right click “SimpleDirectShow “ on the Solution Explorer. Choose “Properties”. A “SimpleDirectShow Property Pages” dialog box will pop up.
Click on “Configuration Properties” → “Linker” → “Input”, you will see an “Additional Dependencies” window.

Key in "Strmiids.lib" and "Quartz.lib". Click “OK” and back to the Property Pages window. Click “OK”. Thus, we include DirectShow libraries.

(d) Add Window Messages

We need to release the resource that we request after we terminate this application. We can add OnClose window message to terminate COM. Click on the Message icon on the properties window. Set “WM_CLOSE” as “OnClose”.

(d) Edit *SimpleDirectShowDlg.h*

```c
/* SimpleDirectShowDlg.h : header file
A very simple application using DirectShow
Modified from AudioBox in DirectX samples.

Author : Guan-Ming Su
Date : 01/02/2003
*/

#pragma once
#include "afxwin.h"
#include "dshow.h"

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

    // Add the public member method
    HRESULT InitDirectShow(void);     // Initialize DirectShow
    HRESULT FreeDirectShow(void);   // Free DirectShow
    void ResetDirectShow(void);            // Reset DirectShow
    HRESULT PrepareMedia(LPTSTR lpszMovie);  // Prepare Media
    HRESULT RunMedia(void);                                // Run Media
    HRESULT StopMedia(void);                               // Stop Media
    HRESULT PauseMedia(void);                             // Pause Media

    // Dialog Data
    enum { IDD = IDD_SIMPLEDIRECTSHOW_DIALOG };

protected:
    virtual void DoDataExchange(CDataExchange*pDX); // DDX/DDV support

    // Implementation
    protected:
    HICON m_hIcon;

    // add your own protected variable
    TCHAR filenamein[255];
    IGraphBuilder *m_pGB;
    IMediaControl *m_pMC;
    IMediaEventEx *m_pME;

    // Add the DirectShow header
    #include "dshow.h"

    // Add your own macro
    #define DEFAULT_MEDIA_PATH TEXT("\0")
    #define SAFE_RELEASE(i) {if (i) i->Release(); i = NULL;}

    // Constructor
    CSimpleDirectShowDlg(CWnd* pParent = NULL); // standard constructor

    // Standard destructor
    ~CSimpleDirectShowDlg();

    // Public member functions
    STDMETHODIMP InitDirectShow(HWND hwnd);
    STDMETHODIMP FreeDirectShow(HWND hwnd);
    STDMETHODIMP ResetDirectShow(HWND hwnd);
    STDMETHODIMP PrepareMedia(HWND hwnd, LPTSTR lpszMovie);
    STDMETHODIMP RunMedia(HWND hwnd);
    STDMETHODIMP StopMedia(HWND hwnd);
    STDMETHODIMP PauseMedia(HWND hwnd);

    // Dialog Data
    enum { IDD = IDD_SIMPLEDIRECTSHOW_DIALOG };`
// Generated message map functions
virtual BOOL OnInitDialog();
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
declare_message_map()
public:
CButton m_button_openfile;
CButton m_button_play;
CButton m_button_pause;
CButton m_button_stop;
afx_msg void OnBnClickedButtonopenfile();
afx_msg void OnBnClickedButtonplay();
afx_msg void OnBnClickedButtonpause();
afx_msg void OnBnClickedButtonstop();
afx_msg void OnClose();
};

(c) Edit SimpleDirectShowDlg.cpp

/* SimpleDirectShowDlg.cpp: implementation file
A very simple application using DirectShow.
Modified from AudioBox in DirectX samples.
Author: Guan-Ming Su
Date: 01/02/2003
*/
#include "stdafx.h"
#include "SimpleDirectShow.h"
#include "SimpleDirectShowDlg.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// Add Global variable ➔
// FILTER_STATE: Specifies a filter's state or the state of the filter graph
FILTER_STATE g_psCurrent=State_Stopped;
// ←

// CSimpleDirectShowDlg dialog
CSimpleDirectShowDlg::CSimpleDirectShowDlg(CWnd* pParent /*=NULL*/)
: CDialog(CSimpleDirectShowDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CSimpleDirectShowDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_BUTTON_openfile, m_button_openfile);
    DDX_Control(pDX, IDC_BUTTON_play, m_button_play);
    DDX_Control(pDX, IDC_BUTTON_pause, m_button_pause);
    DDX_Control(pDX, IDC_BUTTON_stop, m_button_stop);
}
BEGIN_MESSAGE_MAP(CSimpleDirectShowDlg, CDialog)
ON_WM_PAINT() 
ON_WM_QUERYDRAGICON() 
//}}AFX_MSG_MAP
ON_BN_CLICKED(IDC_BUTTON_openfile, OnBnClickedButtonopenfile)
ON_BN_CLICKED(IDC_BUTTON_play, OnBnClickedButtonplay)
ON_BN_CLICKED(IDC_BUTTON_pause, OnBnClickedButtonpause)
ON_BN_CLICKED(IDC_BUTTON_stop, OnBnClickedButtonstop)
END_MESSAGE_MAP()}
ON_WM_CLOSE ()
END_MESSAGE_MAP()

// CSimpleDirectShowDlg message handlers

BOOL CSimpleDirectShowDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    // 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

    // TODO: Add extra initialization here
    // add your own code here ➔

    // ** DirectShow-specific initialization code **
    CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);
    m_pGB = NULL;
    m_pMC = NULL;
    m_pME = NULL;

    // Initialize DirectShow interfaces
    HRESULT hr = InitDirectShow();
    if(FAILED(hr)){
        FreeDirectShow();
        return FALSE;
    }

    g_psCurrent=State_Stopped;
    m_button_play.EnableWindow(FALSE);
    m_button_pause.EnableWindow(FALSE);
    m_button_stop.EnableWindow(FALSE);

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

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

            SendMessage(WM_ICONERASEBKGND, reinterpret_cast<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 function to obtain the cursor to display while the user drags the minimized window.

```cpp
HCURSOR CSimpleDirectShowDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}
```

```cpp
void CSimpleDirectShowDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default

    // Add your own code ➔
    // Release DirectShow interfaces
    StopMedia();
    FreeDirectShow();

    // Release COM
    CoUninitialize();
    //
    CDiallog::OnClose();
}
```

```
// Add the following DirectShow functions ➔ (DS BEGIN)
// ##################################################
// ################## DirectShow ##################
// ##################################################

HRESULT CSimpleDirectShowDlg::InitDirectShow(void)
{
    HRESULT hr = S_OK;
    // Create the filter graph manager
    hr = CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC, 
    IID_IGraphBuilder, (void **)&m_pGB);
    if (FAILED(hr))
        return hr;
    // Query for interfaces
    if (FAILED(hr = m_pGB->QueryInterface(IID_IMediaControl, (void **)&m_pMC))
        return hr;
    if (FAILED(hr = m_pGB->QueryInterface(IID_IMediaEventEx, (void **)&m_pME))
        return hr;
    return S_OK;
}
```

```
HRESULT CSimpleDirectShowDlg::FreeDirectShow(void)
{
    HRESULT hr=S_OK;
    // Disable event callbacks
    if (m_pME)  {
        hr = m_pME->SetNotifyWindow((OAHWND)NULL, 0, 0);
        SAFE_RELEASE(m_pME);
    }
    StopMedia();
    SAFE_RELEASE(m_pMC);
    SAFE_RELEASE(m_pGB);
    return hr;
}
```
void CSimpleDirectShowDlg::ResetDirectShow()
{
    // Destroy the current filter graph its filters.
    FreeDirectShow();
    // Reinitialize graph builder and query for interfaces
    InitDirectShow();
}

// (DS END) 

// Add the following Media functions → (MF BEGIN)
HRESULT CSimpleDirectShowDlg::PrepareMedia(LPTSTR lpszMovie)
{
    USES_CONVERSION;
    HRESULT hr = S_OK;
    if (!m_pGB || (!m_pME))
        return E_NOINTERFACE;
    // Allow DirectShow to create the FilterGraph for this media file
    hr = m_pGB->RenderFile(T2W(lpszMovie), NULL);
    if (FAILED(hr)) {
        MessageBox("Fail on Render file");
        return hr;
    }
    return hr;
}

HRESULT CSimpleDirectShowDlg::RunMedia()
{
    HRESULT hr = S_OK;
    if (!m_pMC)
        return E_NOINTERFACE;
    // Start playback
    hr = m_pMC->Run();
    if (FAILED(hr)) {
        MessageBox("Fail on Run");
        return hr;
    }
    // Remember play state
    g_psCurrent = State_Running;
    return hr;
}

HRESULT CSimpleDirectShowDlg::StopMedia()
{
    HRESULT hr = S_OK;
    if (!m_pMC)
        return E_NOINTERFACE;
    // Stop playback
    hr = m_pMC->Stop();
    if (FAILED(hr)) {
        MessageBox("Fail on Stop");
        return hr;
    }
HRESULT CSimpleDirectShowDlg::PauseMedia(void)
{
    HRESULT hr=S_OK;
    if (!m_pMC)
        return E_NOINTERFACE;
    hr = m_pMC->Pause();
    if (FAILED(hr)) {
        MessageBox("Fail on Pause");
        return hr;    }
    // Remember play state
    g_psCurrent = State_Paused;
    return hr;
}

// (MF END)

// ##################################################
// ########### response for buttons ################
// ##################################################
void CSimpleDirectShowDlg::OnBnClickedButtonopenfile()
{
    // TODO: Add your control notification handler code here

    // add your own code here ➔
    static OPENFILENAME ofn={0};
    static BOOL bSetInitialDir = FALSE;

    // Reset filename
    *filenamein = 0;

    // Fill in standard structure fields
    ZeroMemory(&ofn, sizeof(OPENFILENAME));
    ofn.lStructSize       = sizeof(OPENFILENAME);
    ofn.hwndOwner         = GetSafeHwnd();
    ofn.lpstrFilter       = "Audio Files:*.mp3;*.wav;*.mid;
                          *.mp3;*.wav;*.mid
                          Video Files:*.mpg;*.avi;*.wmv;
                          *.mpg;*.avi;*.wmv
                          Image Files:*.bmp;*.jpg;*.tga;
                          *.bmp;*.jpg;*.tga
                          
                          *

    ofn.lpstrCustomFilter = NULL;
    ofn.nFilterIndex      = 1;
    ofn.lpstrFile         = filenamein;
    ofn.nMaxFile          = MAX_PATH;
    ofn.lpstrFileTitle    = TEXT("Open Multimedia File...");
    ofn.lpstrFileTitle    = TEXT("Open Multimedia File...");
    ofn.lpstrFileTitle    = TEXT("Open Multimedia File...");
    ofn.lpstrFileTitle    = TEXT("Open Multimedia File...");
    ofn.Flags             = OFN_FILEMUSTEXIST | OFN_READONLY | OFN_PATHMUSTEXIST;

    // Remember the path of the first selected file
    if (bSetInitialDir == FALSE) {
        ofn.lpstrInitialDir = DEFAULT_MEDIA_PATH;
        bSetInitialDir = TRUE; }
    else
        ofn.lpstrInitialDir = NULL;

    // Create the standard file open dialog
    GetOpenFileName((LPOPENFILENAME)&ofn);
    ResetDirectShow();
    m_button_play.EnableWindow(TRUE);
    m_button_pause.EnableWindow(TRUE);
    m_button_stop.EnableWindow(TRUE);
void CSimpleDirectShowDlg::OnBnClickedButtonplay()
{
    // TODO: Add your control notification handler code here
    // Add your own code here →
    HRESULT hr = S_OK;
    ResetDirectShow();
    hr = PrepareMedia(filenamein);
    if (SUCCEEDED(RunMedia())) {
        // Set button states
        m_button_play.EnableWindow(FALSE);
        m_button_pause.EnableWindow(TRUE);
        m_button_stop.EnableWindow(TRUE);
    }
    // ←
}

void CSimpleDirectShowDlg::OnBnClickedButtonpause()
{
    // TODO: Add your control notification handler code here
    // add your own code here
    if (g_psCurrent == State_Paused) {
        RunMedia();
        m_button_play.EnableWindow(FALSE);
        m_button_pause.SetText("Pause");
        m_button_pause.EnableWindow(TRUE);
        m_button_stop.EnableWindow(TRUE);
    } else if (g_psCurrent == State_Running) {
        PauseMedia();
        m_button_play.EnableWindow(FALSE);
        m_button_pause.SetText("resume");
        m_button_pause.EnableWindow(TRUE);
        m_button_stop.EnableWindow(TRUE);
    }
    // ←
}

void CSimpleDirectShowDlg::OnBnClickedButtonstop()
{
    // TODO: Add your control notification handler code here
    // add your own code here
    StopMedia();
    m_button_play.EnableWindow(TRUE);
    m_button_pause.EnableWindow(FALSE);
    m_button_pause.SetText("Pause");
    m_button_stop.EnableWindow(FALSE);
    // ←
}

(f) Compile and Run

We can build this program by clicking Build → Build Solution (or press F7) and execute it using Debug → Start without Debugging (Ctrl+F5). You will see the run time GUI.
By clicking “Open Files” button, an “Open Multimedia File” window will show up.

You can change the directory by clicking on “Look in”. If you install DirectX SDK media samples, you can find those samples under “c:\DXSDK\Samples\Media\”.

Choose one files. You can click on “Play”, “Pause”, and “Stop” to control these multimedia files. If you select Files of type as Image or Video, an “ActiveMovie Window” will show up and display the file.
4. Demo Project 2 – Multimedia Player
In this demo project, we inherit and enhance previous example and learn
(1) how to display video and image in the picture control,
(2) how to obtain the status of streaming media, and
(3) how to control the volume.

(a) Create a new project by File → New → Project from Visual Studio .NET menu bar.

(5) Select “Visual C++ Projects” on the Project Types column.
(6) Select “MFC Application” on the Templates column.
(7) Type “MultiMediaViewDS” on Name edit box. Select Locations for this project.
(8) Click “OK” and a MFC Application Wizard window will pop up.

(b) MFC Application Wizard
(1) This tap shows the overview for this project. Since we are going to design a
dialog-based application, we need to change it. Click “Application Type” on the
left blue column.

(2) Choose “Dialog Based” on the “Application Type”, check “MFC standard” on
Project style , “MFC in a shared DLL” on the Use of MFC, and pick “English
(United States)” on the Resource language list box. Thus, we finish the decision
of application type. Next, we design the user interface by clicking “User Interface
Features” on the left blue column.
(3) Uncheck every item in the **Main frame styles**. Type “**MultiMediaViewDS**” in the **Dialog title** edit box. Next, click on “**Advanced Features**” on the left blue column.

(4) We do not need to change the advanced features. Let’s move to next step by clicking on “**Generated Classes**” on the left blue column.

(5) This tap displays the generated classes and the base classed used in our application. We do not need to change the classes in this demo project. Click “**Finish**” to finish the initial setting.
(b) Design Dialog Box

(1) Overview: Click the Resource View tap and open the Dialog folder. You will see IDD_MULTIMEDIAVIEWDS_DIALOG. Click it and you will see another dialog box with text “TODO: Place dialog controls here” shown in the Edit Window. We will design our own GUI on this dialog box. The small window to the right of this dialog box is the “Controls Toolbar”, which provides control components. We will use this toolbar to add controls.

![Diagram of Dialog Box]

(2) Edit the Dialog Box. Delete the “TODO: Place dialog controls here” and the “OK” button by clicking them and pressing “Delete” key on the keyboard. Drag and drop five buttons, two slider controls, two static texts, and one picture control from Controls toolbar to this dialog box. Rearrange those controls and adjust the width and height of this dialog box as the following figure.

![Diagram of Modified Dialog Box]
(3) Edit the properties and add member variables of slider controls. Let’s edit the horizontal slider first.

(i) Right click on horizontal slider and choose “Properties”. The Properties window will be on focus. Set “Auto Ticks” as “True”, “Enable Selection Range” as “True”, “Tick Marks” as “True”, and “ID” as “IDC_SLIDER_POS”.

(ii) Add member variables of the slider. Right click on this slider again. Choose “Add Variable”. An “Add Member Variable Wizard” dialog box will show up.

Type “m_SeekBar” in the “Variable name” and click “Finish”.
(You will direct to Edit Window. To return to the dialog editor, you can press the “Resource View window” and click “IDD_SIMPLEDIRECSHOW_DIALOG” under SimpleDirectShow.rc ➔Dialog.)

(iii) Let’s work on the vertical slider now. Repeat step (i) to change the properties for the vertical slider as follows. Change its “Orientation” as “Vertical”, “Client Edge” as “True”, and “ID” as “IDC_SLIDER_VOLUME”. Repeat step (ii) by right clicking on this volume slider and choose “Add Variable”. Set its variable name as “m_VolumeBar”.

(4) Edit the properties and add member variables of other controls. Repeat step (i) and (ii) in previous section according to the following table.

<table>
<thead>
<tr>
<th>Control</th>
<th>Caption</th>
<th>ID</th>
<th>Add Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button1</td>
<td>Open File</td>
<td>IDC_BUTTON_openfile</td>
<td>m_button_openfile</td>
</tr>
<tr>
<td>Button2</td>
<td>Play</td>
<td>IDC_BUTTON_play</td>
<td>m_button_play</td>
</tr>
<tr>
<td>Button3</td>
<td>Pause</td>
<td>IDC_BUTTON_pause</td>
<td>m_button_pause</td>
</tr>
<tr>
<td>Button4</td>
<td>Stop</td>
<td>IDC_BUTTON_stop</td>
<td>m_button_stop</td>
</tr>
<tr>
<td>Button5</td>
<td>Step</td>
<td>IDC_BUTTON_step</td>
<td>m_button_step</td>
</tr>
<tr>
<td>Cancel</td>
<td>Exit</td>
<td>IDCANCEL</td>
<td></td>
</tr>
<tr>
<td>Static</td>
<td>Position</td>
<td>IDC_STATIC_POSITION</td>
<td>m_StrPosition</td>
</tr>
<tr>
<td>Static2</td>
<td>Volume</td>
<td>IDC_STATIC_Volume</td>
<td>m_static_volume</td>
</tr>
<tr>
<td>PictureControl</td>
<td>N/A</td>
<td>IDC_MOVIE_SCREEN</td>
<td>m_screen</td>
</tr>
</tbody>
</table>

For button1 to button 5, we need to add 5 event handlers using Event Handler Wizard, which can be active by right clicking each control and choose “Add Event Handler”. Choose “BN_CLICKED” on “Message type” and “CMultiMediaViewDSDlg” on “Class List”.

Thus, we finish the interface design. The following figure shows the GUI.
(c) Include DirectShow Libraries

Right click “MultiMediaViewDS” on the Solution Explorer. Choose “Properties”. A “MultiMediaViewDS Property Pages” dialog box will pop up.

Click on “Additional Dependences” under Configuration Properties→Linker→Input. An Additional Dependences window will show up. Type “strmiids.lib” and “msvcrtd.lib” in the edit box.
(d) Add Windows Messages
We need to add two windows messages, **WM_CLOSE** and **OnTimer**. The former is for termination of COM and the latter is for timer setting. Click on the dialog box and go to the Properties window. Click to add messages.

Set “**WM_CLOSE**” as “**OnClose**”, “**WM_TIMER**” as “**OnTimer**”, “**WM_VSCROLL**” as “**ONVScroll**”.

(e) Edit *MultiMediaViewDSDlg.h*

// Add the DirectShow header →
#include "dshow.h"

// Add your own macro →
#define DEFAULT_MEDIA_PATH TEXT\\(\"\\0\"
#define SAFE_RELEASE(i) {if (i) i->Release(); i = NULL;}

// Add const variables →
const int TICKLEN=100, TIMERID=55;
const int MINIMUM_VOLUME=3000;

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

    // Add the public member method →
    HRESULT InitDirectShow(void); // Initialize DirectShow
    HRESULT FreeDirectShow(void); // Free DirectShow

---

Notice that “// Add your own code →” and “// ←” indicates that you may put some codes in these areas. In this example, we use italic red font for the added codes (to distinguish from the codes already automatically generated by eVC).
void ResetDirectShow(void); // Reset DirectShow
HRESULT PrepareMedia(LPTSTR lpszMovie); // Prepare Media
HRESULT RunMedia(void); // Run Media
HRESULT StopMedia(void); // Stop Media
HRESULT PauseMedia(void); // Pause Media
HRESULT StepFrame(void);
void CenterVideo(void);
void StartSeekTimer(void);
void StopSeekTimer();
void ConfigureSeekbar(void);
BOOL CanStep(void);
void UpdatePosition(REference_TIME rtNow);
void ReadMediaPosition(void);

// Dialog Data
c enum { IDD = IDD_MULTIMEDIAVIEWDS_DIALOG };
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

// Implementation
protected:
HICON m_hIcon;

// add your own protected variable
TCHAR filenamein[255];
IGraphBuilder *m_pGB;
IMediaSeeking *m_pMS;
IMediaControl *m_pMC;
IMediaEventEx *m_pME;
IBasicVideo *m_pBV;
IBasicAudio *m_pBA;
IVideoWindow *m_pVW;
UINT_PTR g_wTimerID;
REFERENCE_TIME g_rtTotalTime;

// Generated message map functions
virtual BOOL OnInitDialog();
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
DECLARE_MESSAGE_MAP()
public:
CSliderCtrl m_SeekBar;
CSliderCtrl m_VolumeBar;
CButton m_button_openfile;
CButton m_button_play;
CButton m_button_pause;
CButton m_button_stop;
CButton m_button_step;
CStatic m_StrPosition;
CStatic m_static_volume;
CStatic m_screen;
afx_msg void OnClose();
afx_msg void OnTimer(UINT nIDEvent);
afx_msg void OnPaint(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
afx_msg void OnBtnClickedButtonopenfile();
afx_msg void OnBtnClickedButtonplay();
afx_msg void OnBtnClickedButtonpause();
afx_msg void OnBtnClickedButtonstop();
afx_msg void OnBtnClickedButtonstep();
(f) Edit `MultiMediaViewDSDlg.cpp`

```cpp
/* MultiMediaViewDSDlg.cpp : implementation file
A simple multimedia viewer for image, video, and audio.
It is modified from JukeBox, AudioBox, and StillView in DirectShow sample codes.

Author: Guan-Ming Su
Date: 01/03/03
*/

#include "stdafx.h"
#include "MultiMediaViewDS.h"
#include "MultiMediaViewDSDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
define new DEBUG_NEW
#endif

#define WM_GRAPHNOTIFY WM_APP + 1

// Add Global variable
// FILTER_STATE: Specifies a filter's state or the state of the filter graph
FILTER_STATE g_psCurrent=State_Stopped;
BOOL g_bAudioOnly;

CMultiMediaViewDSDlg::CMultiMediaViewDSDlg(CWnd* pParent /*=NULL*/)
: CDialog(CMultiMediaViewDSDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMultiMediaViewDSDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_SLIDER_Pos, m_SeekBar);
    DDX_Control(pDX, IDC_BUTTON_openfile, m_button_openfile);
    DDX_Control(pDX, IDC_BUTTON_play, m_button_play);
    DDX_Control(pDX, IDC_BUTTON_pause, m_button_pause);
    DDX_Control(pDX, IDC_BUTTON_stop, m_button_stop);
    DDX_Control(pDX, IDC_SLIDER_VOLUME, m_VolumeBar);
    DDX_Control(pDX, IDC_STATIC_POSITION, m_StrPosition);
    DDX_Control(pDX, IDC_MOVIE_SCREEN, m_screen);
}

BEGIN_MESSAGE_MAP(CMultiMediaViewDSDlg, CDialog)
    ON_WM_PAINT()  
    ON_WM_QUERYDRAGICON()  
    ON_WM_CLOSE()  
    ON_WM_TIMER()  
    ON_WM_VSCROLL()  
    ON_BN_CLICKED(IDC_BUTTON_openfile, OnBnClickedButtonopenfile)
    ON_BN_CLICKED(IDC_BUTTON_play, OnBnClickedButtonplay)
    ON_BN_CLICKED(IDC_BUTTON_pause, OnBnClickedButtonpause)
    ON_BN_CLICKED(IDC_BUTTON_stop, OnBnClickedButtonstop)
END_MESSAGE_MAP()

// CMultiMediaViewDSDlg dialog
CMultiMediaViewDSDlg::CMultiMediaViewDSDlg(CWnd* pParent /*=NULL*/)
: CDialog(CMultiMediaViewDSDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMultiMediaViewDSDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_SLIDER_Pos, m_SeekBar);
    DDX_Control(pDX, IDC_BUTTON_openfile, m_button_openfile);
    DDX_Control(pDX, IDC_BUTTON_play, m_button_play);
    DDX_Control(pDX, IDC_BUTTON_pause, m_button_pause);
    DDX_Control(pDX, IDC_BUTTON_stop, m_button_stop);
    DDX_Control(pDX, IDC_SLIDER_VOLUME, m_VolumeBar);
    DDX_Control(pDX, IDC_STATIC_POSITION, m_StrPosition);
    DDX_Control(pDX, IDC_MOVIE_SCREEN, m_screen);
}
```

EEN408G Spring 2003 (Update 01/21/2003)
Multimedia Programming Manual
ON_BN_CLICKED(IDC_BUTTON_step, OnBnClickedButtonstep)
END_MESSAGE_MAP()

// CMultiMediaViewDSDlg message handlers
BOOL CMultiMediaViewDSDlg::OnInitDialog()
{
    CDIalog::OnInitDialog();

    // 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

    // TODO: Add extra initialization here

    // Add your own initialization→
    // DirectShow-specific initialization code

    // Initialize COM
    CoInitializeEx(NULL, COINIT_APARTMENTTHREADED);

    // Initialize DirectShow and query for needed interfaces
    HRESULT hr = InitDirectShow();
    if (FAILED(hr))
    {
        MessageBox("Failed to initialize DirectShow!");
        return FALSE;
    }

    // IMPORTANT
    // Since we're embedding video in a child window of a dialog,
    // we must set the WS_CLIPCHILDREN style to prevent the bounding
    // rectangle from drawing over our video frames.
    //
    // Neglecting to set this style can lead to situations when the video
    // is erased and replaced with black (or the default color of the
    // bounding rectangle).
    m_screen.ModifyStyle(0, WS_CLIPCHILDREN);

    // Initialize seeking trackbar range
    m_SeekBar.SetRange(0, 100, TRUE);
    m_SeekBar.SetToFreq(10);
    m_VolumeBar.SetRange(0, MINIMUM_VOLUME, TRUE);
    g_rtTotalTime=1;

    m_pBA = NULL;

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

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

        SendMessage(WM_ICONERASEBKGND, reinterpret_cast<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 function to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CMultiMediaViewDSDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CMultiMediaViewDSDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default
    // Add your own code
    // Release DirectShow interfaces
    StopMedia();
    FreeDirectShow();
    // Release COM
    CoUninitialize();
    //
    CDialog::OnClose();
}

// #####################################################################
// ####### response to window event #######
// #####################################################################

void CMultiMediaViewDSDlg::OnTimer(UINT nIDEvent)
{
    // TODO: Add your message handler code here and/or call default
    // Add your own code
    ReadMediaPosition();
    //
    CDialog::OnTimer(nIDEvent);
}

void CMultiMediaViewDSDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar)
{
    // TODO: Add your message handler code here and/or call default
    // add your own code
    HRESULT hr= S_OK;
    long lVolume;
    // Disregard ENDSROLL messages, which are redundant
    if ((!m_pBA) /*|| (nPos == SB_ENDSROLL)*/) return;
    // Since the IBasicAudio interface adjusts volume on a logarithmic
    // scale from -10000 to 0, volumes below -4000 sound like silence.
    // Therefore, the slider covers a smaller range.
    int nPosition = m_VolumeBar.GetPos();
    lVolume = -1 * nPosition;
    // Since slider goes from MINIMUM_VOLUME to 0, use the negative value
    // Volume = -1 * nPosition;
TCHAR szPosition[24];
wsprintf(szPosition, _T("%06d\0"), lVolume);
m_static_volume.SetWindowText(szPosition);

// Set new volume
hr = m_pBA->put_Volume(lVolume);
if(FAILED(hr)){
    MessageBox("Fail on setting new volume"); }

// CDialog::OnVScroll(nSBCode, nPos, pScrollBar);

void CMultiMediaViewDSDlg::OnBnClickedButtonopenfile()
{

    // TODO: Add your control notification handler code here

    // add your own code here

    // Reset filename
    *filenamein = 0;

    // Fill in standard structure fields
    ZeroMemory(&ofn, sizeof(OPENFILENAME));
    ofn.lStructSize = sizeof(OPENFILENAME);
    ofn.hwndOwner = GetSafeHwnd();
    ofn.lpstrFilter = "Audio Files:*.mp3;*.wav;*.mid;
                         *.mp3;*.wav;*.mid;0
                         Video Files:*.mpg;*.avi;*.wmv;
                         *.mpg;*.avi;*.wmv;0
                         Image Files:*.bmp;*.jpg;*.tga;
                        *
                        *
                         *.bmp;*.jpg;*.tga;0
                         *
                         *
                         *

    ofn.lpstrCustomFilter = NULL;
    ofn.nFilterIndex = 1;
    ofn.lpstrFile = filenamein;
    ofn.nMaxFile = MAX_PATH;
    ofn.lpstrTitle = TEXT("Open Multimedia File...");
    ofn.lpstrFileTitle = NULL;
    ofn.lpstrDefExt = TEXT("*");
    ofn.Flags = OFN_FILEMUSTEXIST | OFN_READONLY | OFN_PATHMUSTEXIST;

    // Remember the path of the first selected file
    if (bSetInitialDir == FALSE) {
        ofn.lpstrInitialDir = DEFAULT_MEDIA_PATH;
        bSetInitialDir = TRUE; }
    else
        ofn.lpstrInitialDir = NULL;

    // Create the standard file open dialog
    GetOpenFileName((LPOPENFILENAME)&ofn);
    ResetDirectShow();
    m_button_play.EnableWindow(TRUE);
    m_button_pause.EnableWindow(TRUE);
    m_button_stop.EnableWindow(TRUE);
    ResetDirectShow();

    // Set up the seeking trackbar and read capabilities
    HRESULT hr = S_OK;
    hr = PrepareMedia(filenamein);
    ConfigureSeekbar();

    // See if the renderer supports frame stepping on this file.
    // Enable/disable frame stepping button accordingly
    m_button_step.EnableWindow(CanStep());
    m_VolumeBar.EnableWindow(TRUE);

}
void CMultiMediaViewDSDlg::OnBnClickedButtonplay()
{
    // TODO: Add your control notification handler code here
    // add your own code here →
    if (SUCCEEDED(RunMedia())) {
        StartSeekTimer();
        m_button_play.EnableWindow(FALSE);
        m_button_pause.EnableWindow(TRUE);
        m_button_stop.EnableWindow(TRUE);
    }
    // ←
}

void CMultiMediaViewDSDlg::OnBnClickedButtonpause()
{
    // TODO: Add your control notification handler code here
    // add your own code here →
    if (g_psCurrent == State_Paused) {
        RunMedia();
        StartSeekTimer();
        m_button_play.EnableWindow(FALSE);
        m_button_pause.SetWindowText("Pause");
        m_button_pause.EnableWindow(TRUE);
        m_button_stop.EnableWindow(TRUE);
    } else if (g_psCurrent == State_Running) {
        StopSeekTimer();
        PauseMedia();
        m_button_play.EnableWindow(FALSE);
        m_button_pause.SetWindowText("resume");
        m_button_pause.EnableWindow(TRUE);
        m_button_stop.EnableWindow(TRUE);
    }
    // ←
}

void CMultiMediaViewDSDlg::OnBnClickedButtonstop()
{
    // TODO: Add your control notification handler code here
    // add your own code →
    HRESULT hr;
    if (!m_pMC || !m_pMS)
        return;
    // Stop playback immediately with IMediaControl::Stop().
    StopSeekTimer();
    StopMedia();
    // Wait for the stop to propagate to all filters
    OAFilterState fs;
    hr = m_pMC->GetState(500, &fs);
    if (FAILED(hr)) {
        MessageBox("Failed to read graph state!");
    } // Reset to beginning of media clip
    LONGLONG pos = 0;
    hr = m_pMS->SetPositions(&pos, AM_SEEKING_AbsolutePositioning, NULL, AM_SEEKING_NoPositioning);
    if (FAILED(hr)) {
        MessageBox("Failed to seek to beginning of media!");
    }
    // Display the first frame of the media clip, if it contains video.
    // StopWhenReady() pauses all filters internally (which allows the video
    // renderer to queue and display the first video frame), after which
    // it sets the filters to the stopped state. This enables easy preview
    // of the video’s poster frame.
}
hr = m_pMC->StopWhenReady();
if (FAILED(hr)){
    MessageBox("Failed in StopWhenReady!");
}

// Reset slider bar and position label back to zero
m_button_play.EnableWindow(TRUE);
m_button_pause.SetWindowText("Pause");
m_button_pause.EnableWindow(TRUE);
m_button_stop.EnableWindow(TRUE);

ReadMediaPosition();
//
}

void CMultiMediaViewDSDlg::OnBnClickedButtonstep()
{
    // TODO: Add your control notification handler code here
    // add your own code ➔
    StepFrame();
    //
}

// Add the following DirectShow functions ➔ (DS BEGIN)
// ******************************************************************************
// ################## DirectShow ##################
// ******************************************************************************
HRESULT CMultiMediaViewDSDlg::InitDirectShow(void)
{
    HRESULT hr = S_OK;
    g_bAudioOnly = FALSE;

    // Zero interfaces (sanity check)
    m_pVW = NULL;
    m_pBV = NULL;

    // Create the filter graph manager
    hr = CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC,
        IID_IGraphBuilder, (void **)&m_pGB);
    if (FAILED(hr))
        return hr;

    // Query for interfaces
    if (FAILED(hr = m_pGB->QueryInterface(IID_IMediaControl, (void **)&m_pMC))){
        MessageBox("Fail on query interface for Media Control");
        return hr;
    }
    if (FAILED(hr = m_pGB->QueryInterface(IID_IMediaEventEx, (void **)&m_pME))){
        MessageBox("Fail on query interface for Media EventEx");
        return hr;
    }
    if (FAILED(hr = m_pGB->QueryInterface(IID_IBasicVideo, (void **)&m_pBV))){
        MessageBox("Fail on query interface for Basic Video");
        return hr;
    }
    if (FAILED(hr = m_pGB->QueryInterface(IID_IVideoWindow, (void **)&m_pVW))){
        MessageBox("Fail on query interface for Media Seeking");
        return hr;
    }
    if (FAILED(hr = m_pGB->QueryInterface(IID_IMediaSeeking, (void **)&m_pMS))){
        MessageBox("Fail on query interface for Media EventEx");
        return hr;
    }
    if (FAILED(hr = m_pGB->QueryInterface(IID_IBasicAudio, (void **)&m_pBA))){
        MessageBox("Fail on query interface for Basic Audio");
        return hr;
    }
}
HRESULT CMultiMediaViewDSDlg::FreeDirectShow(void)
{
    HRESULT hr=S_OK;
    // Disable event callbacks
    if (m_pME)
    {
        hr = m_pME->SetNotifyWindow((OAHWND)NULL, 0, 0);
        SAFE_RELEASE(m_pME);
    }
    StopMedia();
    SAFE_RELEASE(m_pMC);
    SAFE_RELEASE(m_pGB);
    SAFE_RELEASE(m_pMS);
    SAFE_RELEASE(m_pVW);
    SAFE_RELEASE(m_pBV);
    SAFE_RELEASE(m_pBA);
    SAFE_RELEASE(m_pME);
    return hr;
}

void CMultiMediaViewDSDlg::ResetDirectShow(void)
{
    // Destroy the current filter graph its filters.
    FreeDirectShow();
    // Reinitialize graph builder and query for interfaces
    InitDirectShow();
}

void CMultiMediaViewDSDlg::CenterVideo(void)
{
    LONG width, height;
    HRESULT hr;
    if ((g_bAudioOnly) || (!m_pVW))
    return;
    // Read coordinates of video container window
    RECT rc;
    m_screen.GetClientRect(&rc);
    width  =  rc.right - rc.left;
    height = rc.bottom - rc.top;
    // Ignore the video's original size and stretch to fit bounding rectangle
    hr =  m_pVW->SetWindowPosition(rc.left, rc.top, width, height);
    if (FAILED(hr))
    {
        MessageBox("Failed to set window position!");
        return;
    }
}

// (DS END) <--

// Add the following Media functions          (MF BEGIN)

// *****************************************************
// ******************** Media function ********************
// *****************************************************

HRESULT CMultiMediaViewDSDlg::PrepareMedia(LPTSTR lpszMovie)
{
    USES_CONVERSION;
return hr;
}
HRESULT hr = S_OK;
if (!m_pGB) || (!m_pME)
    return E_NOINTERFACE;

// Allow DirectShow to create the FilterGraph for this media file
hr = m_pGB->RenderFile(T2W(lpszMovie), NULL);
if (FAILED(hr)) {
    MessageBox("Fail on Render file");
    return hr;
}

// Set the message drain of the video window to point to our main
// application window. If this is an audio-only or MIDI file,
// then put_MessageDrain will fail.
hr = m_pVW->put_MessageDrain((OAHWND) m_hWnd);
if (FAILED(hr))
    g_bAudioOnly = TRUE;

// Have the graph signal event via window callbacks
hr = m_pME->SetNotifyWindow((OAHWND)m_hWnd, WM_GRAPHNOTIFY, 0);

// Configure the video window
if (!g_bAudioOnly)
    {
    // We'll manually set the video to be visible
    hr = m_pVW->put_Visible(OAFALSE);
    hr = m_pVW->put_WindowStyle(WS_CHILD);
    hr = m_pVW->put_Owner((OAHWND) m_screen.GetSafeHwnd());

    // Place video window within the bounding rectangle
    CenterVideo();

    // Make the video window visible within the screen window.
    // If this is an audio-only file, then there won't be a video interface.
    hr = m_pVW->put_Visible(OATRUE);
    hr = m_pVW->SetWindowForeground(-1);
    }

    return hr;
}

HRESULT CMultiMediaViewDSDlg::RunMedia()
{
    HRESULT hr=S_OK;
    if (!m_pMC)
        return E_NOINTERFACE;

    // Start playback
    hr = m_pMC->Run();
    if (FAILED(hr)) {
        MessageBox("Fail on Run");
        return hr;
    }

    // Remember play state
    g_psCurrent = State_Running;
    return hr;
}

HRESULT CMultiMediaViewDSDlg::StopMedia()
{
    HRESULT hr=S_OK;
    if (!m_pMC)
        return E_NOINTERFACE;

    // Stop playback
hr = m_pMC->Stop();
if (FAILED(hr)) {
    MessageBox("Fail on Stop");
    return hr;
}

// Remember play state
    g_psCurrent = State_Stopped;
    return hr;
}

HRESULT CMultiMediaViewDSDlg::PauseMedia(void)
{
    HRESULT hr=S_OK;
    if (!m_pMC)
        return E_NOINTERFACE;
    hr = m_pMC->Pause();
    if (FAILED(hr)) {
        MessageBox("Fail on Pause");
        return hr;
    }
    // Remember play state
    g_psCurrent = StatePaused;
    return hr;
}

// (MF END) <--

// Add the Timer and SeekBar codes here →

//###########################################
//########### Timer & SeekBar ###############
//###########################################

void CMultiMediaViewDSDlg::StartSeekTimer()
{
    // Cancel any pending timer event
    StopSeekTimer();

    // Create a new timer
    g_wTimerID = SetTimer(TIMERID, TICKLEN, NULL);
}

void CMultiMediaViewDSDlg::StopSeekTimer()
{
    // Cancel the timer
    if (g_wTimerID)
        KillTimer(g_wTimerID);
    g_wTimerID = 0;
}

void CMultiMediaViewDSDlg::ReadMediaPosition()
{
    HRESULT hr;
    REFERENCE_TIME rtNow;
    if (!m_pMS)
        return;
    // Read the current stream position
    hr = m_pMS->GetCurrentPosition(&rtNow);
    if (FAILED(hr))
        return;
    // Convert position into a percentage value and update slider position
    if (g_rtTotalTime != 0)
    {
        long lTick = (long)((rtNow * 100) / g_rtTotalTime);
    }
m_SeekBar.SetPos(lTick);
} else
m_SeekBar.SetPos(0);

// Update the 'current position' string on the main dialog
UpdatePosition(rtNow);

void CMultiMediaViewDSDlg::UpdatePosition(REFERENCE_TIME rtNow)
{
HRESULT hr;

if (!m_pMS)
return;

// If no reference time was passed in, read the current position
if (rtNow == 0)
{
// Read the current stream position
hr = m_pMS->GetCurrentPosition(&rtNow);
if (FAILED(hr))
return;
}

// Convert the LONGLONG duration into human-readable format
unsigned long nTotalMS = (unsigned long) rtNow / 10000; // 100ns -> ms
int nSeconds = nTotalMS / 1000;
int nMinutes = nSeconds / 60;
nSeconds %= 60;

// Update the display
TCHAR szPosition[24], szCurrentString[24];
wsprintf(szPosition, _T("Position: %02dm:%02ds\0"), nMinutes, nSeconds);

// Read current string and compare to the new string. To prevent flicker,
// don't update this label unless the string has changed.
m_StrPosition.GetText(szCurrentString, 24);
if (_tcscmp(szCurrentString, szPosition))
m_StrPosition.SetWindowText(szPosition);

}

void CMultiMediaViewDSDlg::ConfigureSeekbar()
{
// Disable seekbar for new file and reset tracker/position label
m_SeekBar.SetPos(0);
m_StrPosition.SetWindowText(TEXT("Position: 00m:00s\0"));
g_rtTotalTime = 0;

// If we can't read the file's duration, disable the seek bar
if (m_pMS && SUCCEEDED(m_pMS->GetDuration(&g_rtTotalTime)))
m_SeekBar.EnableWindow(TRUE);
else
m_SeekBar.EnableWindow(FALSE);

}

// Code for Stepping frames ➔
// Some hardware decoders and video renderers support stepping media
// frame by frame with the IVideoFrameStep interface. See the interface
// documentation for more details on frame stepping.

BOOL CMultiMediaViewDSDlg::CanStep(VOID)
{
HRESULT hr;
IVideoFrameStep* m_pFS;

hr = m_pGB->QueryInterface(__uuidof(IVideoFrameStep), (PVOID *)&m_pFS);
if (FAILED(hr))
    return FALSE;

// Check if this decoder can step
hr = m_pFS->CanStep(0L, NULL);
m_pFS->Release();

if (hr == S_OK)
    return TRUE;
else
    return FALSE;
}

HRESULT CMultiMediaViewDSDlg::StepFrame(void)
{
    // Get the Frame Stepping Interface
    HRESULT hr;
    IVideoFrameStep* m_pFS;
    hr = m_pGB->QueryInterface(__uuidof(IVideoFrameStep), (PVOID *)&m_pFS);
    if (FAILED(hr))
        return hr;

    // The graph must be paused for frame stepping to work
    if (g_psCurrent != State_Paused)
        OnBnClickedButtonpause();

    // Step one frame
    hr = m_pFS->Step(1, NULL);
m_pFS->Release();

    // Since the media position has changed slightly, update the
    // slider bar and position label.
    ReadMediaPosition();
    return hr;
}

(g) Compile and Run

We can build this program by clicking Build → Build Solution (or press F7) and
execute it using Debug → Start without Debugging (Ctrl+F5). The following figures
show the results.
Part IV. Microsoft Speech SDK

1. Introduction to Microsoft Speech SDK

Microsoft Speech SDK is a software development kit for building speech engines and applications for Microsoft Windows. Designed primarily for the desktop speech developer, the SDK contains the Microsoft Win32-compatible speech application programming interface (SAPI), the Microsoft continuous speech recognition engine and Microsoft concatenated speech synthesis (or text-to-speech) engine, a collection of speech-oriented development tools for compiling source code and executing commands, sample application and tutorials that demonstrate the use of Speech with other engine technologies, sample speech recognition and speech synthesis engines for testing with speech-enabled applications, and documentation on the most important SDK features.

The SAPI API is built as Component Object Model (COM), which is already introduced in Part II, provides a high-level interface between an application and speech engines. SAPI implements all the low-level details needed to control and manage the real-time operations of various speech engines.

The two basic types of SAPI engines are text-to-speech (TTS) systems and speech recognizers. TTS systems synthesize text strings and files into spoken audio using synthetic voices. Speech recognizers convert human spoken audio into readable text strings and files.

---

5 Microsoft Speech SDK 5.1 can be downloaded at http://www.microsoft.com/speech/download/sdk51/
2. Demo Project: Text-to-Speech
In this demo project, we design a Text-To-Speech (TTS) program using Speech SDK and learn
(1) how to initialize and terminate Speech object and interface,
(2) how to playback, pause, and stop a TTS system, and
(3) how to control the volume and speaking rate.

(a) Create a new project by File $\rightarrow$ New $\rightarrow$ Project from Visual Studio .NET menu bar.

(9) Select “Visual C++ Projects” on the Project Types column.
(10) Select “MFC Application” on the Templates column.
(11) Key in “SimpleTTS” on Name edit box and select a Location for this project.
(12) Click “OK” and a MFC Application Wizard window will pop up.

(b) MFC Application Wizard
(1) This tap shows the overview for this project. Since we are going to design a
dialog-based application, we need to change it. Click “Application Type” on the left blue column.

(2) Choose “Dialog Based” on the “Application Type”, check “MFC standard” on
Project style, “MFC in a shared DLL” on the Use of MFC, and pick “English
(United States)” on the Resource language list box. Thus, we finish the decision of 
application type. Next, we design the user interface by clicking “User Interface 
Features” on the left blue column.

(3) Uncheck every item in the Main frame styles. Type “Simple Text to Speech” in the 
Dialog title edit box. Next, click on “Advanced Features” on the left blue column.

(4) We do not need to change the advanced features. Let’s move to next step by 
clicking on “Generated Classes” on the left blue column.
(5) This tap displays the generated classes and the base classed used in our application. We do not need to change the classes in this demo project. Click “Finish” to finish the initial setting.

(b) Design Dialog Box

(1) Overview: Click the Resource View tap and open the Dialog folder. You will see IDD_SIMPLETTS_DIALOG. Click it and you will see another dialog box with text “TODO: Place dialog controls here” shown in the Edit Window. We will design our own GUI on this dialog box. The small window to the right of this dialog box is the “Controls Toolbar”, which provides control components. We will use this toolbar to add controls.

(2) Edit the Dialog Box. Delete the “TODO: Place dialog controls here” and the “OK” button by clicking them and pressing “Delete” key on the keyboard. Drag and drop three buttons, two sliders, two static texts, and one edit box from Controls toolbar to this dialog box. Rearrange those controls and adjust the width and height of this dialog box as the following figure.

(3) Edit the properties, add member variables, and event handlers of controls. Let’s edit Button1 first.
(i) Right click on Button1 and choose “Properties”. The Properties window will be on focus. Change “Caption” to “Speak” and “ID” to “IDC_BUTTON_speak”.

(ii) Add member variables of buttons. Right click on Button1 and choose “Add Variable”. An “Add Member Variable Wizard” dialog box will show up. Make sure the “Category” is “Control”. Type “m_button_speak” in “Variable name”.

(iii) Add event handler. Right click on Button1 and choose “Add Event Handler...”. An “Event Handler Wizard” dialog box will show up.
Add “OnBnClickedButtonspeak” (which is already done by Event Handler Wizard).

(iv). Repeat step (i) ~(iii) for other controls according to the following table.

<table>
<thead>
<tr>
<th>Controls</th>
<th>ID</th>
<th>Caption</th>
<th>Variable Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Button1</td>
<td>IDC_BUTTON_speak</td>
<td>Speak</td>
<td>m_button_speak</td>
</tr>
<tr>
<td>Button2</td>
<td>IDC_BUTTON_pause</td>
<td>Pause</td>
<td>m_button_pause</td>
</tr>
<tr>
<td>Button3</td>
<td>IDC_BUTTON_stop</td>
<td>Stop</td>
<td>m_button_stop</td>
</tr>
<tr>
<td>Slider1</td>
<td>IDC_SLIDER_Volume</td>
<td>N/A</td>
<td>m_slider_volume</td>
</tr>
<tr>
<td>Slider2</td>
<td>IDC_SLIDER_rate</td>
<td>N/A</td>
<td>m_slider_rate</td>
</tr>
<tr>
<td>Static Text1</td>
<td>IDC_STATIC_volume</td>
<td>Volume</td>
<td>m_static_volume</td>
</tr>
<tr>
<td>Static Text2</td>
<td>IDC_STATIC_rate</td>
<td>Rate</td>
<td>m_static_rate</td>
</tr>
<tr>
<td>Group Box1</td>
<td>IDC_STATIC_options</td>
<td>Options</td>
<td>N/A</td>
</tr>
</tbody>
</table>

(v). Change the “Caption” of the “Cancel” button to “Exit”.

(vi) For the Edit Control, change its properties as follows, “ID” as “IDC_EDIT_TTSInput”, “Auto HScroll” as “False”, “Multiline” as “True”, and ”Vertical Scroll” as “True”. Add one variable for edit control by right clicking on this control and choosing “Add Variable”. Change the “Category” to “Value” and set “Variable name” as “m_sInputText”.

We finish the interface design. The following figure shows the GUI.

(c) Include additional directories and dependences
(1) Right click “SimpleTTS“ on the Solution Explorer. Choose “Properties”. A “SimpleTTS Property Pages” dialog box will pop up.

Choose “Additional Include Directories” under Configuration Properties → C/C++ → General. Click \(\text{Add}\). An Additional Include Directories window will show up.

To add new additional include directory, click \(\text{Add}\). You will see a new blank line in the edit box. Click \(\text{Add}\) to specify the path, “C:\Program Files\Microsoft Speech SDK5.1\Include”. Click “OK” to include this path.

(2) We also need to include other libraries. Choose “Additional Dependences” under Configuration Properties → Linker → Input.
Type “sapi.lib” in the edit box. Click “OK”. Back to Property Pages window and press “OK” to finish these settings.

(d) Override Window member functions

We need to override member function for this dialog box. Double click on “SimpleTTSDlg.cpp” in the Solution Explorer. The Properties Window will display the CSimpleTTSDlg class, which is shown in the left figure below. We can override functions by clicking . Use scroll bar to find “WindowProc” and override it.

(e) Add Window Messages

Click “IDD_SIMPLETTS_DIALOG” on Resource View window. Right click the dialog box and go to the Properties window. Click Messages icon, , and add “WM_CLOSE”.

(f) Edit stdafx.h
To use COM (Component Object Model), we need to add codes in “stdafx.h”. You can double click on this file under SimpleTTS → Header Files → stdafx in Solution Explorer. Add the following codes at the end of this file.

```c
#include <atibase.h>
extern CComModule _Module;
#include <aticom.h>
```

(g) Edit SimpleTTSDlg.h

```c
/* SimpleTTSDlg.h : header file
A simple Text to Speech example using Microsoft Speech SDK 5.1.
which can be downloaded from
Speech SDK 5.1 http://www.microsoft.com/speech/download/sdk51/

Several sample codes:
1. under Speech SDK folder, you can find several codes.
2. A very simple example
   http://www.codeproject.com/audio/speech.asp
3. A talking head with mouth motion example
   http://www.codeproject.com/audio/speechmouth.asp

Author: Guan-Ming Su
Date: 01/04/2003
*/

#pragma once
#include "afxwin.h"
#include "afxcmn.h"

// include Speech API header ⇒
#include <sapi.h>
#include "afxcmn.h"

// CSimpleTTSDlg dialog
class CSimpleTTSDlg : public CDialog
{
    // Construction
    public:
        CSimpleTTSDlg(CWnd* pParent = NULL); // standard constructor
    
        // Add your own public function here ⇒
        void HandleHorizontalTrackbar(WPARAM wParam, LPARAM lParam);
        
    // Dialog Data
    enum { IDD = IDD_SIMPLETTS_DIALOG };

    protected:
        virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

    // Implementation
    protected:
        HICON m_hIcon;
}
```

Notice that “// Add your own code ⇒” and “//⇐” indicates that you may put some codes in these areas. In this example, we use italic red font for the added codes (to distinguish from the codes already automatically generated by eVC).
// add your own protected code here

ISpVoice * pVoice; // Text to Speech interface
int g_svoice; // a state for button pause

// Generated message map functions
virtual BOOL OnInitDialog();
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
DECLARE_MESSAGE_MAP()

public:
    CButton m_button_speak;
    CButton m_button_pause;
    CButton m_button_stop;
    CStatic m_static_volume;
    CStatic m_static_rate;
    CSliderCtrl m_slider_volume;
    CSliderCtrl m_slider_rate;
    CString m_sInputText;
protected:
    virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);

public:
    afx_msg void OnBnClickedButtonspeak();
    afx_msg void OnBnClickedButtonpause();
    afx_msg void OnBnClickedButtonstop();
    afx_msg void OnClose();

};

(h) Edit SimpleTTSDlg.cpp

/* SimpleTTSDlg.cpp : implementation file
A simple Text to Speech example using Microsoft Speech SDK 5.1.
which can be downloaded from
Speech SDK 5.1 http://www.microsoft.com/speech/download/sdk51/

Several sample codes:
1. under Speech SDK folder, you can find several codes.
2. A very simple example
   http://www.codeproject.com/audio/speech.asp
3. A talking head with mouth motion example
   http://www.codeproject.com/audio/speechmouth.asp

Author: Guan-Ming Su
Date: 01/04/2003
*/

#include "stdafx.h"
#include "SimpleTTS.h"
#include "SimpleTTSDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

#include "stdafx.h"
#include "SimpleTTS.h"
#include "SimpleTTSDlg.h"

if dEFAULT
#define new DEFAULT_NEW
#endif

// CSimpleTTSDlg dialog

CSimpleTTSDlg::CSimpleTTSDlg(CWnd* pParent /*=NULL*/) :
    CDialog(CSimpleTTSDlg::IDD, pParent)
    , m_sInputText(T(""))
{    
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}
void CSimpleTTSDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_BUTTON_speak, m_button_speak);
    DDX_Control(pDX, IDC_BUTTON_pause, m_button_pause);
    DDX_Control(pDX, IDC_BUTTON_stop, m_button_stop);
    DDX_Control(pDX, IDC_STATIC_volume, m_static_volume);
    DDX_Control(pDX, IDC_STATIC_rate, m_static_rate);
    DDX_Control(pDX, IDC_SLIDER_Volume, m_slider_volume);
    DDX_Control(pDX, IDC_SLIDER_Rate, m_slider_rate);
    DDX_Text(pDX, IDC_EDIT_TTSinput, m_sInputText);
}

BEGIN_MESSAGE_MAP(CSimpleTTSDlg, CDialog)
    ON_WM_PAINT()    
    ON_WM_QUERYDRAGICON()    
    ON_BN_CLICKED(IDC_BUTTON_speak, OnBnClickedButtonspeak)
    ON_BN_CLICKED(IDC_BUTTON_pause, OnBnClickedButtonpause)
    ON_BN_CLICKED(IDC_BUTTON_stop, OnBnClickedButtonstop)
    ON_WM_CLOSE()    
END_MESSAGE_MAP()

// CSimpleTTSDlg message handlers

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

    // 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

    // TODO: Add extra initialization here
    // Add your own initialization here →
    m_slider_volume.SetRange(0, 100, TRUE);
    m_slider_volume.SetTicFreq(10);
    m_slider_volume.SetPos(50);
    m_static_volume.SetWindowText("Volume: 050\0");
    m_slider_rate.SetRange(-10, 10, TRUE);
    m_slider_rate.SetTicFreq(10);
    m_slider_rate.SetPos(0);
    m_static_rate.SetWindowText("Rate: 0\0");

    g_svoice = 0;
    pVoice = NULL;
    if (FAILED(CoInitialize(NULL))){
        AfxMessageBox("Error to intilize COM");
        return TRUE;  // return TRUE unless you set the focus to a control
    }

    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL, IID_ISpVoice, (void **)&pVoice);
    return TRUE;  // return TRUE unless you set the focus to a control
}

// 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 CSimpleTTSDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting
        SendMessage(WM_ICONERASEBKGND, reinterpret_cast<LPARAM>(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 function to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CSimpleTTSDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CSimpleTTSDlg::OnClose()
{
    // TODO: Add your message handler code here and/or call default

    // Add your own code here →
    pVoice->Release();
pVoice = NULL;
CoUninitialize();
    // ←
    CDIalog::OnClose();
}

// #####################################################################
// #### Response to control ######
// #####################################################################

void CSimpleTTSDlg::OnBnClickedButtonspeak()
{
    // TODO: Add your control notification handler code here

    // Add your own code here →
    UpdateData();

    g_svoice = 0;
    HRESULT hr;
    hr = pVoice->Speak(m_sInputText.AllocSysString(), SPF_ASYNC, NULL);
    if (FAILED(hr))
    {
        MessageBox("Fail to speak!");
    }

    m_button_speak.EnableWindow(FALSE);
m_button_pause.EnableWindow(TRUE);
m_button_stop.EnableWindow(TRUE);
    // ←
}

void CSimpleTTSDlg::OnBnClickedButtonpause()
{
    // TODO: Add your control notification handler code here

    // add your own code here

    HRESULT hr;
    if(g_svoice == 0) // not resume
hr = pVoice->Pause();
m_button_pause.SetWindowText("Resume");
g_svoice = 1;
else{
hr = pVoice->Resume();
m_button_pause.SetWindowText("Pause");
g_svoice = 0;
}

m_button_speak.EnableWindow(FALSE);
m_button_pause.EnableWindow(TRUE);
m_button_stop.EnableWindow(TRUE);
if( FAILED( hr ) ) {
    MessageBox("Fail to pause/resume"); }
//
}

void CSimpleTTSDlg::OnBnClickedButtonstop()
{
    // TODO: Add your control notification handler code here
    // Add your own code here →
    UpdateData(FALSE);
g_svoice = 0;

    HRESULT hr;
    hr = pVoice->Speak(NULL, SPF_PURGEBEFORESpeak, 0);
    if( FAILED( hr ) ) {
        MessageBox("Fail to stop"); }
    m_button_speak.EnableWindow(TRUE);
m_button_pause.EnableWindow(TRUE);
m_button_stop.EnableWindow(FALSE);
    //
}

// #############################
// ##### slider control #######
// #############################

LRESULT CSimpleTTSDlg::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
    // TODO: Add your specialized code here and/or call the base class
    // add your own code here →
    // We override the WindowProc to deal with the message switch (message)
    // add Horizontal Scroll bar (e.g. volume and rate) here →
    // First determine which slider was adjusted
    HANDLEHorizontalTrackbar(wParam, lParam);
    break;
    //
    return CDialog::WindowProc(message, wParam, lParam);
}

// add Horizontal Scroll bar (e.g. volume and rate) here →
void CSimpleTTSDlg::HandleHorizontalTrackbar(WPARAM wParam, LPARAM lParam)
{
    if (wParam != SB_ENDSCROLL)
    {
        // First determine which slider was adjusted
        HWND hwnd = (HWND) lParam;
        //
int nID = ::GetWindowLong(hwnd, GWL_ID);
CSliderCtrl *pSlider = (CSliderCtrl *)GetDlgItem(nID);

// Read the current value of the adjusted slider
DWORD dwPosition = pSlider->GetPos();
HRESULT hr;
TCHAR szPosition[24];

switch (nID) {
    case IDC_SLIDER_Volume:
        hr = pVoice->SetVolume(dwPosition);
        wsprintf(szPosition, _T("Volume: %03d\0"), dwPosition);
        m_static_volume.SetWindowText(szPosition);
        break;
    case IDC_SLIDER_rate:
        hr = pVoice->SetRate(dwPosition);
        wsprintf(szPosition, _T("Rate: %03d\0"), dwPosition);
        m_static_rate.SetWindowText(szPosition);
        break;
}

(i) Compile and Run

We can build this program by clicking Build → Build Solution (or press F7) and execute it using Debug → Start without Debugging (Ctrl+F5). You will see the run time GUI. Type some sentences in the edit box. You can click Play button to let this application speak. You can also change the volume and speaking rate by changing the corresponding sliders.
Part V. Further Reference

1. DirectX and Speech SDK Documentation
   Both DirectX and Speech SDK have well-written document and sample codes accessible from Start ➔ Programs ➔ DirectX (or Speech SDK).

2. Books:

3. Web Sites
   (a) Microsoft MSDN: http://msdn.microsoft.com
   (b) CodeGuru: http://www.codeguru.com/
   (c) Code Project: http://www.codeproject.com/