IMPLIMENTATION OF MEDIA PLAYER

By

ANURAG SINGH-031228
KARAN SEHGAL-031236
ATUL UNIYAL-031422

JAYPEE UNIVERSITY OF
INFORMATION TECHNOLOGY

MAY-2007

Submitted in partial fulfillment of the Degree of Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING AND INFORMATION TECHNOLOGY
JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY-WAKNAGHAT

CERTIFICATE

This is to certify that the work entitled, "**Implementation of Media Player**" submitted by **Anurag Singh (031228), Karan Sehgal (031236) and Atul Uniyal (031422)** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

29.05.07

Mr. Ajay Kumar Singh
(Senior Lecturer)
Department of Computer Science Engineering and Information Technology,
Jaypee University of Information Technology,
Waknaghat, Solan – 173215, Himachal Pradesh,
INDIA

ACKNOWLEDGEMENT

We wish to express our earnest gratitude to **Mr. Ajay Kumar Singh**, for providing us valuable guidance and timely suggestions by the help of which we successfully completed our project – IMPLEMENTATION OF MEDIA PLAYER. We'd also like to thank him for his moral support in times when we were facing difficulties.

We would also like to the thank **Brig. (retd.) S. P. Ghrera,** (HOD, Department of Computer Science Engineering and the faculty of the Computer Science Engineering Department of Jaypee University of Information Technology, Waknaghat for their valuable suggestions that made us improve our project.

We would like to thank all the staff members of the Computing facilities of **Jaypee University of Information Technology**, Waknaghat, for providing us with support and facilities required for the completion of this project.

Anurag Singh – 031228

Karan Sehgal – 031236

Atul Uniyal – 031422

ABSTRACT

The implementation of a media player for Windows platform is presented. The player can play digital video and audio files on the machine.

CONTENTS

Chapter 3: Codec

Chapter 4: The Project

Chapter 5: Conclusion

LIST OF TABLES AND FIGURES

LIST OF ABBREVATIONS

.MPEG/.MPG	Moving Picture Experts Group
.MP3	MPEG-1, Audio Layer 3
.WAV	Windows Audio File
.WMA	Windows Media Audio
.WMV	Windows Media Video
.DAT	Digital Audio Tape
API	Application Program Interface
MCI	Multimedia Control Interface

CHAPTER 1
INTRODUCTION TO MEDIA PLAYER

INTRODUCTION

A media player is a piece of application software for playing back multimedia files. Most media players support an array of media formats, including both audio and video files. It's all about playing media with different file extensions. A media player uses codec to compress and decompress files of different formats and plays it.

We have used the VB.NET to develop our media player. The media player developed by us supports all file formats and comes with a setup file which can be executed on any windows platform. The operations supported by the media player are PLAY, PAUSE, STOP, REPEAT, OPEN and CLOSE file. In addition to these basic features we have included the feature of separately muting the left and the right speakers. The TEMPO of a media file can also be altered as desired by the user. We can play both audio and video files through this media player. The different file formats supported by the software include .mp3, .wmv, .avi, .dat and many more. The main advantage of this player as compared to other players is that it supports all the different file formats.

HISTORY OF MEDIA PLAYER'S

There is an almost endless list of media players currently available in the market, both audio and video players. The most popular among them being Windows Media Player, Real Player, QuickTime, VLC and Winamp.

List of few media players in the market:

Name	Creator	First public release date	Stable version	Cost (USD)	Software license	Proprietary format
iTunes	Apple Computer	January 9, 2001	7.0.2	Free	Proprietary	Apple Lossless (part of QuickTime)
Media Center	J. River	July 2003	11.1.090	$40.00	Proprietary	None
Media Player Classic	Gabest	May 29, 2003	6.4.9.0	Free	GPL	.mpcpl (playlists) .dsm (container)
Musicmatch Jukebox	Yahoo!	May 1998	10.1	Free (basic), $19.99 (plus)	Proprietary	None
Napster	William Christopher Gorog	October 2003	3.7.2.6	$9.95 and up	Proprietary	N.A
QuickTime	Apple Computer	December 1991	7.1.3 (September 12, 2006)	Free (basic), $29.95	Proprietary	QuickTime
RealPlayer	RealNetworks	1995	10.5	Free (basic)	Proprietary	RealAudio, RealVideo
VLC media player	VideoLAN	February 2000	0.8.5 (May 6, 2006)	Free	GPL	None
Winamp	Nullsoft	June 1997	5.31 (October'06)	Free (basic), $19.95 (Pro)	Proprietary	NSA, NSV
Windows Media Player	Microsoft	November 1992	11	Windows License	Proprietary	WMA, WMV

Table 1.1: Features of media player's available in the market.

Comparison:

	Audio playback	Video playback	Outbound streaming	Skinnable	Media Database	Gapless Audio Decoding	Visualizer	Remote Controllable
iTunes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
jetAudio	Yes	Yes	Yes	Yes	Yes	N.A	Yes	N.A
Media Center	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Media Player Classic	Yes	Yes	No	No	No	N.A	No	Yes
MPlayer	Yes	Yes	Yes	Yes	No	No	N.A	Yes
Musicmatch Jukebox	Yes	No	No	Yes	Yes	N.A	Yes	N.A
QuickTime	Yes	Yes	Yes	Partial	No	No	No	N.A
RealPlayer	Yes	Yes	No	Yes	Yes	N.A	Yes	N.A
VLC	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Winamp	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Windows Media Player	Yes	Yes	Yes	Yes	Yes	Yes	Yes	N.A
WMV Player	Yes	Yes	No	No	No	N/A	No	N.A
	Audio playback	**Video playback**	**Outbound streaming**	**Skinnable**	**Media Database**	**Gapless MP3/AAC Decoding**	**Visualizer**	**Remote Controllable**

Table 1.2: Comparison of features

CHAPTER 2
LANGUAGE USED – VB.NET

VB.NET

Visual Basic .NET (VB.NET) is an object-oriented computer language that can be viewed as an evolution of Microsoft's Visual Basic (VB) implemented on the Microsoft .NET framework. Visual Basic .NET provides the easiest, most productive language and tool for rapidly building Windows and Web applications. It comes with enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE). It also supports creation of applications for wireless, Internet-enabled hand-held devices.

The great majority of VB.NET developers use Visual Studio .NET as their integrated development environment (IDE). SharpDevelop provides an open-source alternative IDE.

Like all .NET languages, programs written in VB.NET require the .NET framework to execute.

Visual Basic .NET 2003 was released with version 1.1 of the .NET Framework. New features included support for the .NET Compact Framework and a better VB upgrade wizard. Improvements were also made to the performance and reliability of the .NET IDE (particularly the background compiler) and runtime.

In addition, Visual Basic .NET 2003 was also available in the *Visual Studio .NET 2003 Academic Edition* (VS03AE). VS03AE is distributed to a certain number of scholars from each country for free.

Basic features of VB.NET

- Full Object-Oriented Constructs.
- Provides the easiest, most productive language and tool for rapidly building windows and web applications.
- Easy application deployment and maintenance.
- Comes with enhanced visual designers and increased application performance.
- Features for building more robust applications easily and quickly:
    - Powerful form designer.
    - In-place menu editor.
- Powerful Integrated Development Environment (IDE).

Integrated Development Environment (IDE)

- Also known as integrated design environment and integrated debugging environment.
- Computer software that assists computer programmers to develop software.
- Consist of:
    - A source code editor.
    - A compiler and/or interpreter.
    - Build-automation tools.
    - A debugger.

Some other features:

Powerful Windows-based Applications

Visual Basic .NET comes with features such as a powerful new forms designer, an in-place menu editor, and automatic control anchoring and docking. Visual Basic .NET delivers new productivity features for building more robust applications easily and quickly. With an improved integrated development environment (IDE) and a significantly reduced startup time, Visual Basic .NET offers fast, automatic formatting of code as you type, improved IntelliSense, an enhanced object browser and XML designer, and much more.

Building Web-based Applications

With Visual Basic .NET we can create Web applications using the shared Web Forms Designer and the familiar "drag and drop" feature. You can double-click and write code to respond to events. Visual Basic .NET 2003 comes with an enhanced HTML Editor for working with complex Web pages. We can also use IntelliSense technology and tag completion, or choose the WYSIWYG editor for visual authoring of interactive Web applications.

Simplified Deployment

With Visual Basic .NET we can build applications more rapidly and deploy and maintain them with efficiency. Visual Basic .NET 2003 and .NET Framework 1.1 makes "DLL Hell" a thing of the past. Side-by-side versioning enables multiple versions of the same component to live safely on the same machine so that applications can use a specific version of a component. XCOPY-deployment and Web auto-download of Windows-based applications combine the simplicity of Web page deployment and maintenance with the power of rich, responsive Windows-based applications.

Powerful, Flexible, Simplified Data Access

You can tackle any data access scenario easily with ADO.NET and ADO data access. The flexibility of ADO.NET enables data binding to any database, as well as classes, collections, and arrays, and provides true XML representation of data. Seamless access to ADO enables simple data access for connected data binding scenarios. Using ADO.NET, Visual Basic .NET can gain high-speed access to MS SQL Server, Oracle, DB2, Microsoft Access, and more.

Improved Coding

You can code faster and more effectively. A multitude of enhancements to the code editor, including enhanced IntelliSense, smart listing of code for greater readability and a background compiler for real-time notification of syntax errors transforms into a rapid application development (RAD) coding machine.

Direct Access to the Platform

Visual Basic developers can have full access to the capabilities available in .NET Framework 1.1. Developers can easily program system services including the event log, performance counters and file system. The new Windows Service project template enables to build real Microsoft Windows NT Services. Programming against Windows Services and creating new Windows Services is not available in Visual Basic .NET Standard, it requires Visual Studio 2003 Professional, or higher.

Full Object-Oriented Constructs

You can create reusable, enterprise-class code using full object-oriented constructs. Language features include full implementation inheritance, encapsulation, and polymorphism. Structured exception handling provides a global error handler and eliminates spaghetti code.

COM Interoperability

You can maintain your existing code without the need to recode. COM interoperability enables you to leverage your existing code assets and offers seamless bi-directional communication between Visual Basic 6.0 and Visual Basic .NET applications.

Reuse Existing Investments

You can reuse all your existing ActiveX Controls. Windows Forms in Visual Basic .NET 2003 provide a robust container for existing ActiveX controls. In addition, full support for existing ADO code and data binding enable a smooth transition to Visual Basic .NET 2003.

Upgrade Wizard

You upgrade your code to receive all of the benefits of Visual Basic .NET 2003. The Visual Basic .NET Upgrade Wizard, available in Visual Basic .NET 2003 Standard Edition, and higher, upgrades up to 95 percent of existing Visual Basic code and forms to Visual Basic .NET with new support for Web classes and UserControls.

.NET Framework consists of two main components:

- Common Language Runtime (CLR).
- Class Libraries.

Common Language Runtime (CLR):

- It is the "execution engine" of .NET.
- It manages the execution of programs.
- Provides the environment within the program run.
- When the program is compiled the output of the compiler is not an executable file but a file that contains a special type of code: the Microsoft Intermediate Language (MSIL). This defines a set of portable instructions that are independent of any specific CPU.
- It's the job of the CLR to translate the intermediate code into an executable code when the program is executed making the program to run in any environment for which the CLR is implemented.
- That's how the .NET framework achieves portability.
- This MSIL is turned into executable code using the just in time (JIT) compiler.
- This makes the program to run fast.

Class Libraries:

- It is the second major entity of the .NET framework.
- It gives the program access to runtime environment.
- It consists of lots of pre-written code.
- Code for all the elements like forms and controls.

VB.NET vs. VB:

The main and the most important difference is that VB.NET is *object oriented* whereas VB is *object based.*

- The biggest change from VB to VB .NET is that VB .NET is Object-Oriented. VB .NET now supports all the key OOP features like Inheritance, Polymorphism, Data Abstraction and Encapsulation. We can now create classes and objects, derive classes from other classes and so on. The major advantage of OOP is code reusability. VB does not support polymorphism.
- Many new controls have been added to the toolbar to make application development more efficient
- VB .NET now adds Console Applications to it apart from Windows and Web Applications.
- New keywords are added and old one's are either removed or renamed
- VB.NET now supports structured exception handling using Try...Catch...
- VB .NET now supports Multithreading. A threaded application allows to do number of different things at once, running different execution threads allowing to use system resources.

Relation to Visual Basic

Whether Visual Basic .NET should be considered as just another version of Visual Basic or a completely different language is a topic of debate. This is not obvious, as once the methods that have been moved around and which can be automatically converted are accounted for, the basic syntax of the language has not seen many "breaking" changes, just additions to support new features like structured exception handling and short circuited expressions. One simple change that can be confusing to previous users is that of Integer and Long data types, which have each doubled in length; a 16-bit integer is known as a Short in VB.NET, while Integer and Long are 32 and 64 bits respectively. Similarly, the Windows Forms GUI editor is very similar in style and function to the Visual Basic form editor.

The things that *have* changed significantly are the semantics — from those of an object based programming language running on a deterministic, reference-counted engine based on COM to a fully object-oriented language backed by the .NET Framework, which consists of a combination of the Common Language Runtime (a virtual machine using generational garbage collection and a just-in-time compilation engine) and a far larger class library. The increased breadth of the latter is also a problem that VB developers have to deal with when coming to the language, although this is somewhat addressed by the *My* feature in Visual Studio 2005.

The changes have altered many underlying assumptions about the "right" thing to do with respect to performance and maintainability. Some functions and libraries no longer exist; others are available, but not as efficient as the "native" .NET alternatives. Even if they compile, most converted VB6 applications will require some level of refactoring to take full advantage of the new language. Extensive documentation is available to cover changes in the syntax, debugging applications, deployment and terminology.

Comparative samples

The following simple example demonstrates similarity in syntax between VB and VB.NET. Both examples pops a message box saying "Hello, World" with an OK button.

Classic VB example:

```
Private Sub Command1_Click()
    MsgBox "Hello, World"
End Sub
```

A VB.NET example:

```
Private Sub Button1_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Button1.Click
    MessageBox.Show("Hello, World")
End Sub
```

- Note that all procedure calls must be made with parentheses in VB.NET, whereas these were only required for function calls (however in VB6 they could be used in procedure calls as well by using the `Call` keyword).
- Also note that the names `Command1` and `Button1` are not obligatory. However, these are default names for a command button in VB6 and VB.NET respectively.
- Actually, there is a function called `MsgBox` in the `Microsoft.VisualBasic` namespace, but the `System.Windows.Forms.MessageBox` class is a preferred way of displaying message boxes since it has more features and is less language-specific.

The following example demonstrates a difference between VB6 and VB.NET. Both examples unload the active window.

Classic VB Example:

```
Private Sub cmdClose_Click()
    Unload Me
End Sub
```

A VB.NET example:

```
Private Sub cmdClose_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles cmdClose.Click
    Me.Close()
End Sub
```

Another useful example of the 'Me' namespace; this example shows the differences between changing opacity in VB6 and VB.net over time.

VB6 Example:

```
Private Sub Timer1_Tick()
    Form1.Opacity = Form1.opacity - 0.01
End Sub
```

VB.NET example:

```
Private Sub Timer1_Tick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Timer1.Tick
    Me.Opacity -= 0.01
End Sub
```

This gives another example of the numerous uses the 'Me' namespace has, as well as how it can increase workflow by having a lot of useful methods available so quickly.

ADVANTAGES OF OOP:

- OOP provides a clear modular structure for programs which make it good for defining abstract data types where implementation details are hidden and the unit has a clearly defined interface.

- OOP makes it easy to maintain and modify existing code as new objects can be created with small differences to existing ones.

- One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

- OOP provides a good framework for code libraries where supplied software components can be easily adapted and modified by the programmer. This is particularly useful for developing graphical user interface.

Concepts of OOP:

- Objects
- Classes
- Data Abstraction and Encapsulation
- Inheritance
- Polymorphism

Objects

Objects are the basic run-time entities in an object-oriented system. Programming problem is analyzed in terms of objects and nature of communication between them. When a program is executed, objects interact with each other by sending messages. Different objects can also interact with each other without knowing the details of their data or code.

Classes

A class is a collection of objects of similar type. Once a class is defined, any number of objects can be created which belong to that class.

Data Abstraction and Encapsulation

Abstraction refers to the act of representing essential features without including the background details or explanations. Classes use the concept of abstraction and are defined as a list of abstract attributes. Storing data and functions in a single unit (class) is encapsulation. Data cannot be accessible to the outside world and only those functions which are stored in the class can access it.

Inheritance

Inheritance is the process by which objects can acquire the properties of objects of other class. In OOP, inheritance provides reusability, like, adding additional features to an existing class without modifying it. This is achieved by deriving a new class from the existing one. The new class will have combined features of both the classes.

Polymorphism

Polymorphism means the ability to take more than one form. An operation may exhibit different behaviors in different instances. The behavior depends on the data types used in the operation. Polymorphism is extensively used in implementing Inheritance.

VB.NET vs. JAVA:

- Java does not have a visual interface.

- Writing of heaps of code to develop applications and on the other hand in VB.NET has rich visual interface and supports drag and drop.

- You can code faster and more effectively. A multitude of enhancements to the code editor, including enhanced IntelliSense, smart listing of code for greater readability and a background compiler for real-time notification of syntax errors transforms into a rapid application development (RAD) coding machine.

- The .NET Framework makes it easy to deploy applications. In the most common form, to install an application, all you need to do is copy the application along with the components it requires into a directory on the target computer.

- The .NET Framework handles the details of locating and loading the components an application needs, even if several versions of the same application exist on the target computer. The .NET Framework ensures that all the components the application depends on are available on the computer before the application begins to execute.

- VB.NET supports **Cross language integration, Cross language Debugging** and **Cross language inheritance.**

CHAPTER 3
CODEC

CODEC

Codec is a device or program capable of performing encoding and decoding on a digital data stream or signal.

It's software that is used to compress or decompress a digital media file, such as an audio or a video. Content creators use codec because a compressed file takes up less storage space on your computer and can be transferred across the Internet more quickly and smoothly. When you play a digital media file, Media Player uses a codec to decompress the file. Codec are used to create and play nearly all audio or video files on a computer or on Web sites.

COMPONENTS OF CODEC

- **Encoder**
- **Decoder**

The **encoder** compresses a file during creation, and the **decoder** decompresses the file so that it can be played. Some codec include both components, while other codec only include one.

Example: If you install a DVD playback program on your computer, the program will likely install a codec that only includes an MPEG-2 decoder, which allows you to play the DVD on your computer. You would not be able to use the decoder to create your own DVD, which allows you to play the DVD on your computer.

HOW DO CODEC WORK:

The media player is the software on the client computer that decompresses the streaming video or audio using a codec and plays it back on the computer screen.

- Codec encode a stream or signal for transmission, storage or encryption and decode it for viewing or editing.

- The raw encoded form of audio and video data is often called essence, to distinguish it from the metadata information that together make up the information content of the stream and any "wrapper" data that is then added to aid access to or improve the robustness of the stream.

An audio compressor converts analog audio signals into digital signals for transmission or storage. A receiving device then converts the digital signals back to analog using an audio decompressor, for playback.

DIFFERENCE BETWEEN CODEC AND FILE FORMATS:

Codec and file formats are not the same although it can be confusing because they sometimes have the same name. You can think of a file format as a type of container. Inside the container is data that has been compressed by using a particular codec.

Example: A file format such as Media Audio contains data that is compressed by using the Media Audio codec. However, a file format such as Audio Video Interleaved (AVI) can contain data that is compressed by any number of different codec, including the MPEG-2, DivX, or XVid codec. AVI files can also contain data that is not compressed by any codec. Consequently, you might be able to play some AVI files and not others, depending on which codec were used to compress the file and which codec you have installed on your computer. For the same reason, you also might be able to play the audio portion of an AVI file, but not the video portion.

USES OF CODEC:

- Many codec's are designed to emphasize certain aspects of the media to be encoded. For example, a digital video (using a DV codec) of a sports event, such as baseball or soccer, needs to encode motion well but not necessarily exact colors, while a video of an art exhibit needs to perform well encoding color and surface texture.

- A file can be compressed by more than one codec. For example, one codec might be used to compress the audio portion of a file and another codec might be used to compress the video portion of a file. If you have the right audio codec installed on your computer but not the right video codec, when you play the file you'll probably be able to hear the sound but you won't be able to see the picture.

VIDEO CODEC:

A video codec is a device or software that enables video compression and or decompression for digital video. The compression usually employs lossy data compression. Historically, video was stored as an analog signal on magnetic tape. Around the time when the compact disc entered the market as a digital-format replacement for analog audio, it became feasible to also begin storing and using video in digital form and a variety of such technologies began to emerge.

There is a complex balance between the video quality, the quantity of the data needed to represent it, also known as the bit rate, the complexity of the encoding and decoding algorithms, robustness to data losses and errors, ease of editing, random access, the state of the art of compression algorithm design, end-to-end delay, and a number of other factors.

VIDEO CODEC DESIGN:

A typical digital video codec design starts with conversion of camera-input video from RGB color format to YCbCr color format, and often also chroma sub sampling to produce a 4:2:0 (or sometimes 4:2:2 in the case of interlaced video) sampling grid pattern. The conversion to YCbCr provides two benefits: first, it improves compressibility by providing decorrelation of the color signals; and second, it separates the luma signal, which is perceptually much more important, from the chroma signal, which is less perceptually important and which can be represented at lower resolution

Some amount of spatial and temporal down sampling may also be used to reduce the raw data rate before the basic encoding process. The most popular such transform is the 8x8 discrete cosine transform (DCT). The output of the transform is first quantized, and then entropy encoding is applied to the quantized values. When a DCT has been used, the coefficients are typically scanned using a zigzag scan order, and the entropy coding typically combines a number of consecutive zero-valued quantized coefficients with the value of the next non-zero quantized coefficient into a single symbol, and also has special ways of indicating when all of the remaining quantized coefficient values are equal to zero. The entropy coding method typically uses variable-length coding tables. Some encoders can compress the video in a multiple step process called n-pass encoding (e.g. 2-pass), which performs a slower but potentially better quality compression.

The decoding process consists of performing, to the extent possible, an inversion of each stage of the encoding process. The one stage that cannot be exactly inverted is the quantization stage. There, a best-effort approximation of inversion is performed. This part of the process is often called "inverse quantization" or "dequantization", although quantization is an inherently non-invertible process.

Video codec designs are often standardized or will be in the future. However, only the decoding process needs to be standardized to enable interoperability. The encoding process is typically not specified at all in a standard, and implementers are free to design their encoder however they want, as long as the video can be decoded in the specified manner. For this reason, the quality of the video produced by decoding the results of different encoders that use the same video codec standard can vary dramatically from one encoder implementation to another.

Commonly used standards and codec

A variety of codec can be implemented with relative ease on PCs and in consumer electronics equipment. It is therefore possible for multiple codec to be available in the same product, avoiding the need to choose a single dominant codec for compatibility reasons. In the end it seems unlikely that one codec will replace them all. Some widely-used video codec are listed below, starting with a chronological-order list of the ones specified in international standards.

H.261: Used primarily in older videoconferencing and video telephony products. H.261, developed by the ITU-T, was the first practical digital video compression standard. Essentially all subsequent standard video codec designs are based on it. It included such well-established concepts as YCbCr color representation, the 4:2:0 sampling format, 8-bit sample precision, 16x16 macro blocks, block-wise motion compensation, 8x8 block-wise discrete cosine transformation, zigzag coefficient scanning, scalar quantization, run+value symbol mapping, and variable-length coding. H.261 supported only progressive scan video.

MPEG-1 Part 2: Used for Video CDs, and also sometimes for online video. The quality is roughly comparable to that of VHS. If the source video quality is good and the bit rate is high enough, VCD can look better than VHS, and all in all very good, but VCD requires high bit rates for this. However, to get a fully compliant VCD file, bit rates higher than 1150 kbit/s and resolutions higher than 352 x 288 should not be used (includes the *.mp3 standard). When it comes to compatibility, VCD has the highest compatibility of any digital video/audio system. Almost every computer in the world can play this codec, and very few DVD players do not support it. In terms of technical design, the most significant enhancements in MPEG-1 relative to H.261 were half-pel and bi-predictive motion compensation support. MPEG-1 supported only progressive scan video.

MPEG-2 Part 2 (a common-text standard with **H.262**): Used on DVD, SVCD, and in most digital video broadcasting and cable distribution systems. When used on a standard DVD, it offers good picture quality and supports widescreen. When used on SVCD, it is not as good as DVD but is certainly better than VCD. In terms of technical design, the most significant enhancement in MPEG-2 relative to MPEG-1 was the addition of support for interlaced video. MPEG-2 is now considered an aged codec, but has tremendous market acceptance and a very large installed base.

H.263: Used primarily for videoconferencing, video telephony, and internet video. H.263 represented a significant step forward in standardized compression capability for progressive scan video. Especially at low bit rates, it could provide a substantial improvement in the bit rate needed to reach a given level of fidelity.

MPEG-4 Part 2: An MPEG standard that can be used for internet, broadcast, and on storage media. It offers improved quality relative to MPEG-2 and the first version of H.263. Its major technical features beyond prior codec standards consisted of *object-oriented* coding features and a variety of other such features not necessarily intended for improvement of ordinary video coding compression capability. It also included some enhancements of compression capability, both by embracing capabilities developed in H.263 and by adding new ones such as quarter-pel motion compensation. Like MPEG-2, it supports both progressive scan and interlaced video.

MPEG-4 Part 10: (A technically aligned standard with the ITU-T's **H.264** and often also referred to as **AVC**). This emerging new standard is the current state of the art of ITU-T and MPEG standardized compression technology, and is rapidly gaining adoption into a wide variety of applications. It contains a number of significant advances in compression capability, and it has recently been adopted into a number of company products, including for example the PlayStation Portable, iPod, the Nero Digital product suite, Mac OS X v10.4, as well as HD DVD/Blue-ray Disc.

DivX, Xvid, FFmpeg MPEG-4 and 3ivx: Different implementations of MPEG-4 Part 2.

VP6: A proprietary video codec developed by On2 Technologies.

Sorenson 3: A codec that is popularly used by Apple's QuickTime, basically the ancestor of H.264. Many of the QuickTime movie trailers found on the web use this codec.

Theora: Developed by the Xiph.org Foundation as part of their Ogg project, based upon On2 Technologies' VP3 codec, and christened by On2 as the successor in VP3's lineage, Theora is targeted at competing with MPEG-4 video and similar lower-bit rate video compression schemes.

WMV (Windows Media Video): Microsoft's family of video codec designs including WMV 7, WMV 8, and WMV 9. It can do anything from low resolution video for dial up internet users to HDTV. WMV can be viewed as a version of the MPEG-4 codec design. The latest generation of WMV is standardized by SMPTE as the VC-1 standard.

Real Video: Developed by Real Networks. A popular codec technology a few years ago, it is now fading in importance for a variety of reasons.

Cinepak: A very early codec used by Apple's QuickTime.

X264: A GPL-licensed implementation of H.264 encoding standard.

Huffyuv: Huffyuv (or HuffYUV) is a very fast, lossless Win32 video codec written by Ben Rudiak-Gould and published under the terms of the GPL as free software, meant to replace uncompressed YCbCr as a video capture format. See Lagarith as a more up-to-date codec.

Lagarith: A more up-to-date fork of Huffyyuv is available as Lagarith.

SheerVideo: A family of ultra fast lossless QuickTime and AVI codec, developed by BitJazz Inc., for RGB[A], Y'CbCr[A] 4:4:4[:4], Y'CbCr[A] and 4:2:2[:4] formats; for both 10-bit and 8-bit channels; for both progressive and interlaced data; for both Mac and PC.

All of the codec above have their qualities and drawbacks. Comparisons are frequently published. The tradeoff between compression power, speed, and fidelity (including artifacts) is usually considered the most important figure of technical merit.

Missing Codec and Video File Issues

A common problem when an end user wants to watch a video stream encoded with a specific codec is that if the exact codec is not present and properly installed on the user's machine, the video won't play (or won't play optimally).

Windows XP SP2 itself only has a very limited number of video and audio codec installed; other than Microsoft formats, Intel Indeo is the only available .avi Codec that is installed per default. All other codec, such as DivX, Xvid or Theora, must be installed manually.

Some video files and codec analysis tools have been made available to provide a user-friendly way to solve this common problem:

VideoInspector: Analyzes most containers (AVI, Matroska, MPEG, etc.) and gives direct download links for missing codec.

GSpot: A pioneer in troubleshooting video applications, GSpot remains a useful tool despite missing some features present in other software.

MediaInfo: Open-source alternative to GSpot.

AVICodec: Another useful application.

AVI2Clipboard: An extension for the Explorer context menu to easily view and save information about videos with an AVI container.

Many people find that VLC media player resolves many of these issues because it contains many popular codec in a portable standalone library, available for many operating systems, including Windows, Linux, and Mac OS X. This also resolves many issues within Windows in conflicting and poorly installed codec.

QUICKTIME:

QuickTime is a multimedia framework developed by Apple Computer, capable of handling various formats of digital video, media clips, sound, text, animation, music, and several types of interactive panoramic images.

QUICKTIME FILE FORMATS:

A QuickTime file (*.mov) functions as a multimedia container file that contains one or more *tracks*, each of which store a particular type of data, such as *audio*, *video*, *effects*, or *text*.

WORKING:

Each track in turn contains *track media*, either the digitally-encoded media stream (using a specific codec such as Cinepak, Sorenson codec, MP3, JPEG, DivX, or PNG) or a data reference to the media stored in another file or elsewhere on a network. It also has an "edit list" that indicates what parts of the media to use.

Internally, QuickTime files maintain this format as a tree-structure of "atoms", each of which uses a 4-byte OSType identifier to determine its structure. An atom can be a parent to other atoms or it can contain data, but it cannot do both.

The ability to contain abstract data references for the media data, and the separation of the media data from the media offsets and the track edit lists means that QuickTime is particularly suited for editing, as it is capable of importing and editing in place (without data copying) other formats such as AIFF, DV, MP3, MPEG-1, and AVI. Other later-developed media container formats such as Microsoft's Advanced Streaming Format or the open source Ogg and Matroska containers lack this abstraction, and require all media data to be rewritten after editing.

CONTAINER BENIFITS:

Both the MOV and MP4 containers can utilize the same MPEG-4 codec; therefore, they are mostly interchangeable in a QuickTime-only environment. However, MP4, being an international standard, has more support. This is especially true on hardware devices, such as the Sony PSP and various DVD players; on the software side, most DirectShow / Video for Windows codec packs [4] [5] include an MP4 parser, but not one for MOV.

In QuickTime Pro's MPEG-4 Export dialog, an option called "Passthrough" allows a clean export to MP4 without affecting the audio or video streams. One recent discrepancy ushered in by QuickTime 7 is that the MOV file format now supports multichannel audio (used, for example, in the high-definition trailers on Apple's site[6]), while MP4 is limited to stereo. Therefore multichannel audio must be re-encoded during MP4 export.

CHAPTER 4

THE PROJECT

METHODOLOGY:

The methodology that would be followed during the completion of this project would be in accordance with the **WATERFALL MODEL**. It is also known as the **SOFTWARE DEVELOPMENT LIFECYCLE (SDLC)**. This section deals with the various phases in the SDLC and the way in which this project will proceed, or the phases that would be encountered in the completion of this project.

The waterfall model derives its name due to the cascading effect from one phase to the other. In this model each phase has well defined starting and ending point, with identifiable deliveries to the next phase.

The model consists of six distinct stages, namely:

- In the **Requirements Analysis** phase
  - The problem is specified along with the desired service objectives (goals).
  - The constraints are identified.
- In the **Specification phase** the system specification is produced from the detailed definitions of (a) and (b) above. This document should clearly define the product function.
- In the system and software **Design phase**, the system specifications are translated into a software representation. The software engineer at this stage is concerned with:
  - Data structure
  - Software architecture
  - Algorithmic detail and
  - Interface representations

The hardware requirements are also determined at this stage along with a picture of the overall system architecture. By the end of this stage the software engineer should be able to identify the relationship between the hardware, software and the associated interfaces. Any faults in the specification should ideally not be passed 'down stream'.

- In the **Implementation and testing phase** stage the designs are translated into the software domain
    - Detailed documentation from the design phase can significantly reduce the coding effort.
    - Testing at this stage focuses on making sure that any errors are identified and that the software meets its required specification.
- In the *integration and system testing* phase all the program units are integrated and tested to ensure that the complete system meets the software requirements. After this stage the software is delivered to the customer.
- The *maintenance* phase the usually the longest stage of the software. In this phase the software is updated to:
    - Meet the changing customer needs.
    - Adapted to accommodate changes in the external environment.
    - Correct errors and oversights previously undetected in the testing phases.
    - Enhancing the efficiency of the software.

Road Map

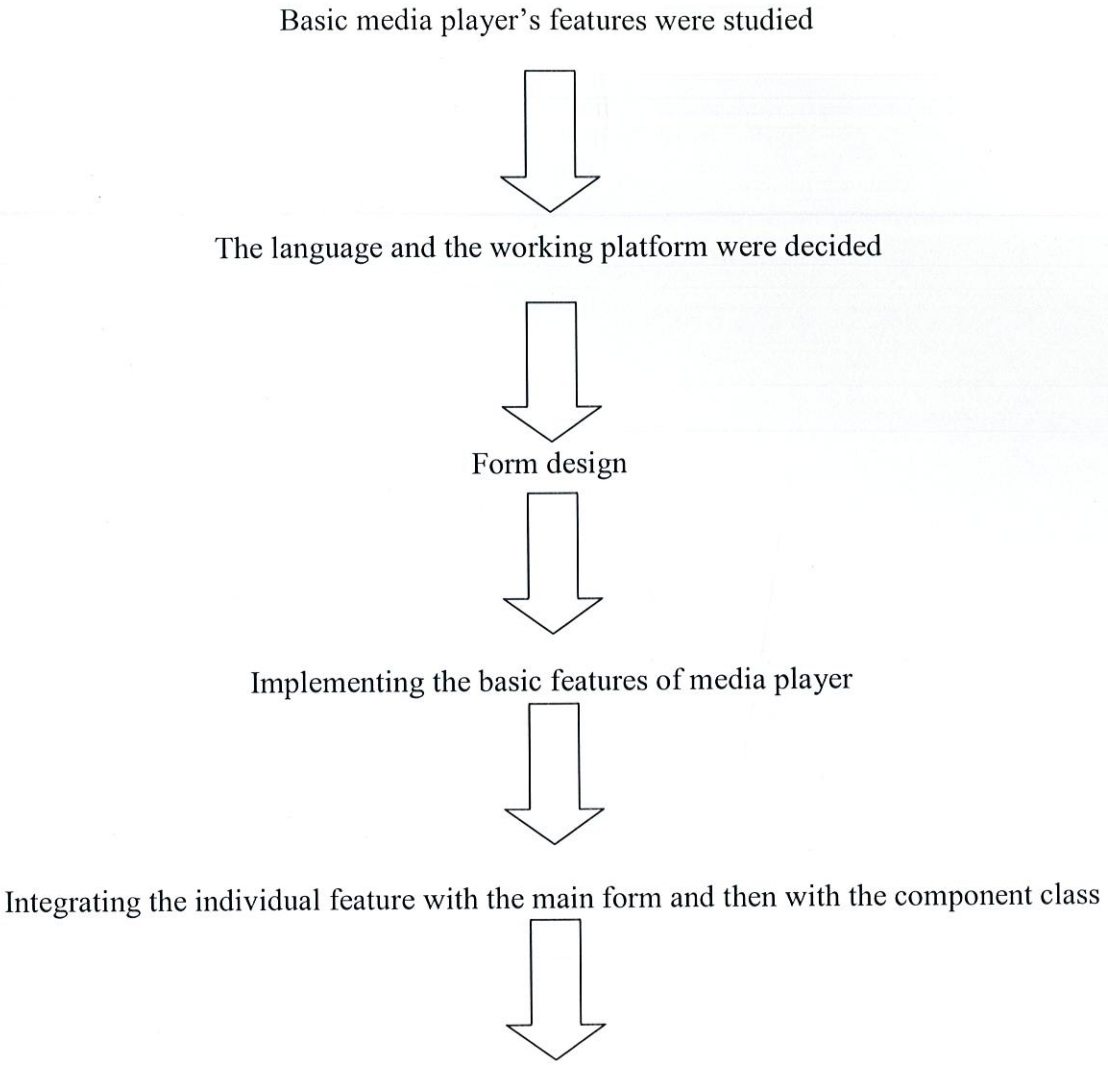

Figure 4.1: Road Map

SNAPSHOTS:

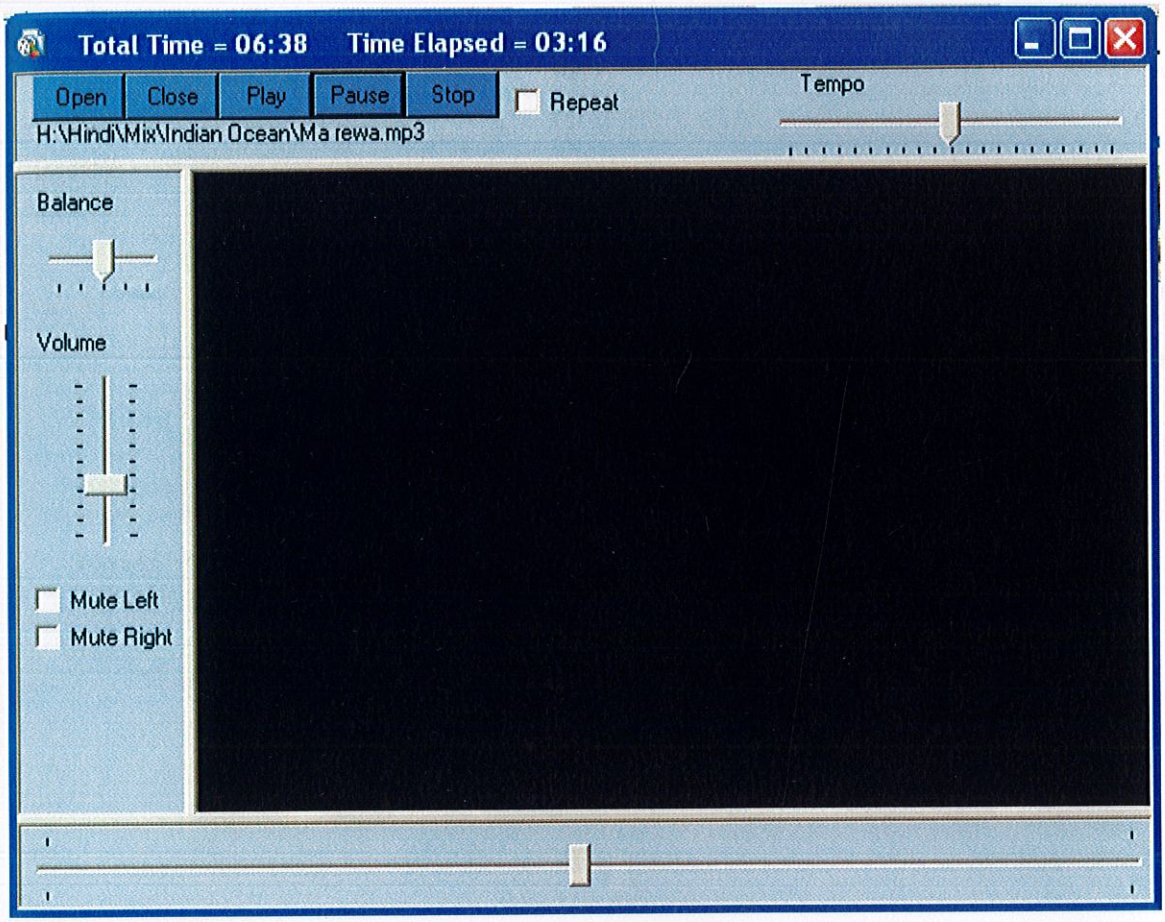

Figure 4.2: An audio file played on the player.

The title bar shows the total time and the time elapsed. Other features (Tempo, volume, mute, balance, repeat, etc.) of the player can also be seen.

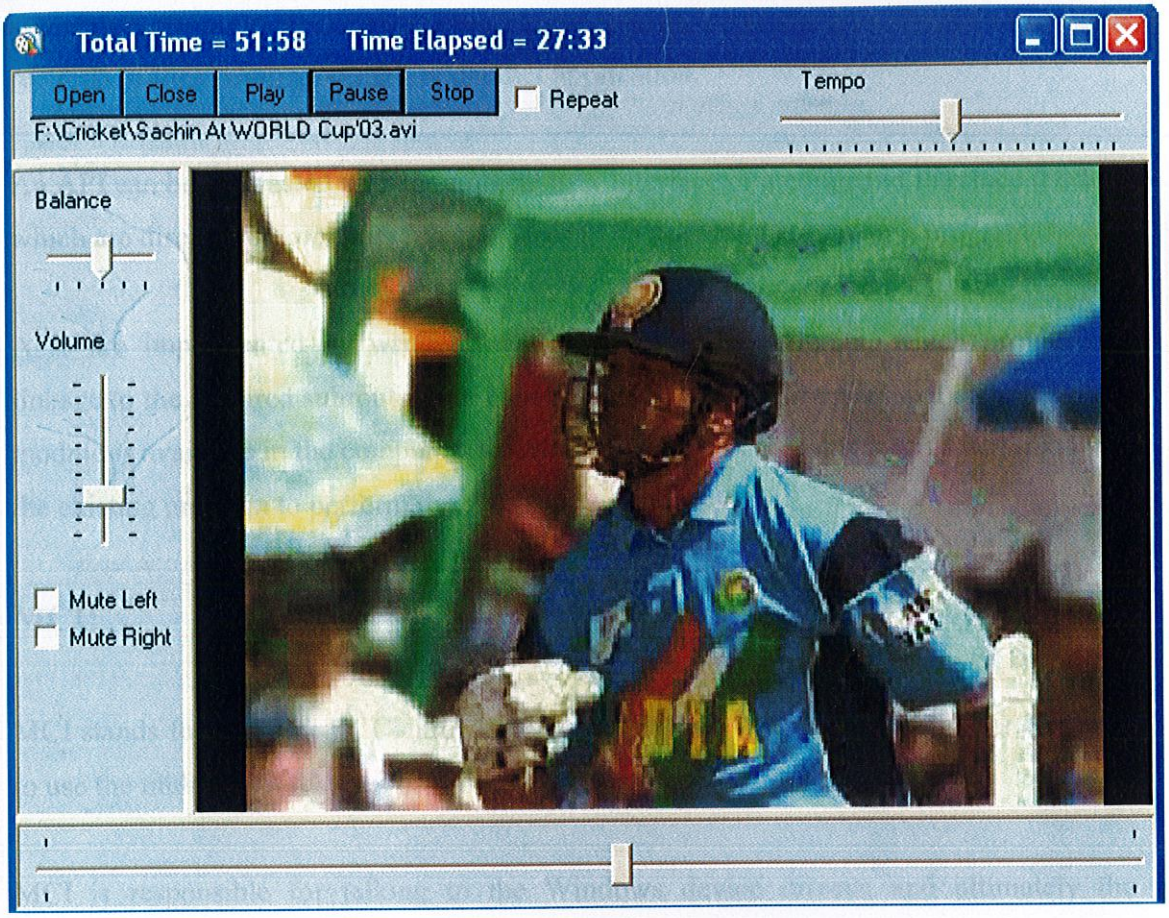

Figure 4.3: A video file played on the player.

API:

An application program interface (API - and sometimes spelled application programming interface) is the specific method prescribed by a computer operating system or by an application program by which a programmer writing an application program can make requests of the operating system or another application.

An API can be contrasted with a graphical user interface or a command interface (both of which are direct user interfaces) as interfaces to an operating system or a program.

APIs are implemented by writing function calls in the program, which provide the linkage to the required subroutine for execution. Thus, an API implies that some program module is available in the computer to perform the operation or that it must be linked into the existing program to perform the tasks.

MCI:

MCI stands for Multimedia Control Interface, which provides a device-independent way to use the multimedia features of Windows through code

MCI is responsible for talking to the Windows device drivers, and ultimately the multimedia hardware. The programmer, issue commands to the MCI using the API call *mciSendString()*. These commands are then translated into calls to the appropriate Windows device driver. To put it into .NET terms then, the MCI is a built-in Windows class.

SENDING STRINGS TO MCI:

mciSendString() function:

error = mciSendString(sCmd , sRetStr, iReturn, hCallback);

sCmd--themci command string (specifies command & device)

sRetStr--return string buffer (NULL if none used)

iReturn--size of return string buffer (0 if none used)

hCallback--Handle to Callback window (NULL if none used)

- Returns 0 if command is successful, error code if not.

Some MCI Command String Commands:

- **open** -- initializes a multimedia device.
- **play** – starts playing an open device.
- **stop** -- stops playing from an open device.
- **seek** -- move to a specified position on device.
- **close** -- closes a device and associated resources.
- **set** -- establish control settings for the device.

MAIN MODULES:

OPEN FILE:

```
Private Sub BtnOpen_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnOpen.Click

    Dim AskFile As New OpenFileDialog

    AskFile.AddExtension = True
    AskFile.CheckFileExists = True
    AskFile.CheckPathExists = True
    AskFile.Multiselect = False
    AskFile.ShowReadOnly = False

    If AskFile.ShowDialog(Me) = DialogResult.OK Then
      MyApiVideo.Open(AskFile.FileName)

      If MyApiVideo.TotalTime >= 0 Then
        Label1.Text = AskFile.FileName

        TrackOffset.Value = 0
        TrackOffset.Minimum = 0
        TrackOffset.Maximum = CInt(MyApiVideo.TotalTime)
      Else
        MsgBox("The file can not be opened. Please check the file extension",
MsgBoxStyle.OKOnly + MsgBoxStyle.Exclamation + MsgBoxStyle.DefaultButton1 +
MsgBoxStyle.SystemModal, "Error")
        Label1.Text = ""

      End If

      ' Play the file right away.
      MyApiVideo.Play()

    End If
    AskFile.Dispose()

  End Sub
```

This module opens the open file dialogue box and then checks for the file if its a file with valid file format or not. If the file format is a valid one then it interacts with the open function in ApiVideo class and then calls the Play() function of the class to play the media else it displays an error message to the user.

PLAY FILE:

```
Private Function [Play](ByVal WithClipStart As Boolean, ByVal WithClipEnd As
Boolean, ByVal WithPauseState As Boolean) As Boolean

    ' Start playing the media.
    If pOpenSuccess = True Then
      If pClipFormat.Length > 0 Then
        pLastError = mciSendString("set " & pAlias & " time format " & pClipFormat,
vbNullString, 0, IntPtr.Zero)
      End If
      pLastError = mciSendString("play " & pAlias & CStr(IIf((pClipStart <> -1) And
WithClipStart = True, " from " & CStr(pClipStart), "")) & CStr(IIf((pClipEnd <> -1) And
WithClipEnd = True, " to " & CStr(pClipEnd), "")) & " notify", vbNullString, 0,
Me.Handle)
      If pLastError = MCIERR.MCIERR_NO_ERROR Then
        pPlaying = True
        If WithPauseState = True And pPaused = True Then
          Me.Pause()
        Else
          pPaused = False
        End If
      End If
      Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

  End Function
```

If the file is opened successfully MCI command string "seek" is used to establish control settings for the device. If no MCIERR error is found then the media is played. After that the media is checked for pause state, if true then it is paused else played.

CHAPTER 5
CONCLUSION

LIMITATIONS

- The media player is unable to play compact file formats like real player format.
- A few .avi video media files are giving the error *"Cannot find 'vids: x264' decompressor"*.

FUTURE PROSPECTS:

- A setup or executable file may be provided so that the media player can be installed on any windows platform for general use.

- The player may also be able to play files directly from the internet.

We need to create an executable (.exe) file for this application. To do that select *Build->Build* from the main menu this builds *Deploy.exe*. Next, we need to create an installer file for Deploy (which is the example) which is a file with .msi extension. For that, select *File->Add Project->New Project* which opens the new project dialogue. Select "*Setup and Deployment Projects*" icon in the projects type box and Setup Wizard in the templates box.

It looks like the image below:

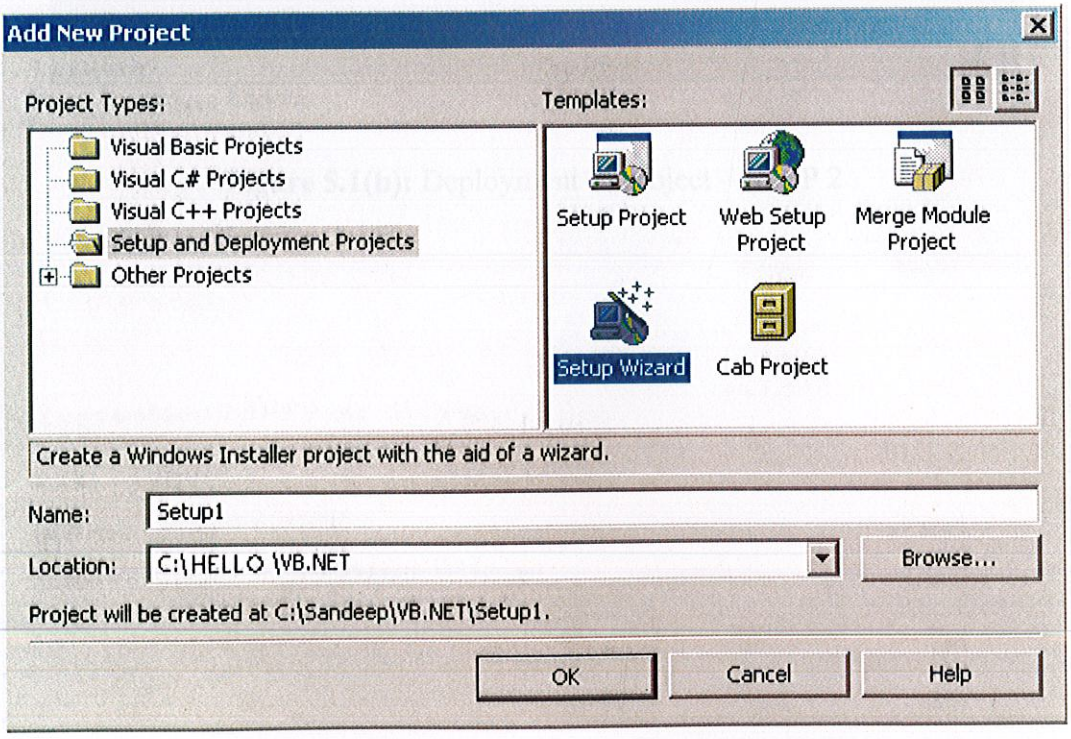

Figure 5.1(a): Deployment of project – STEP 1

Click OK to open the Setup Wizard. The Setup wizard looks like the image below:

Figure 5.1(b): Deployment of project – STEP 2

Click next on the above pane to take you to second pane in the Wizard. The new pane allows us to create deployment projects both for Windows and Web Applications. Here, select the radio button which says "Create a setup for Windows Application" as this is deploying a windows application and click next.

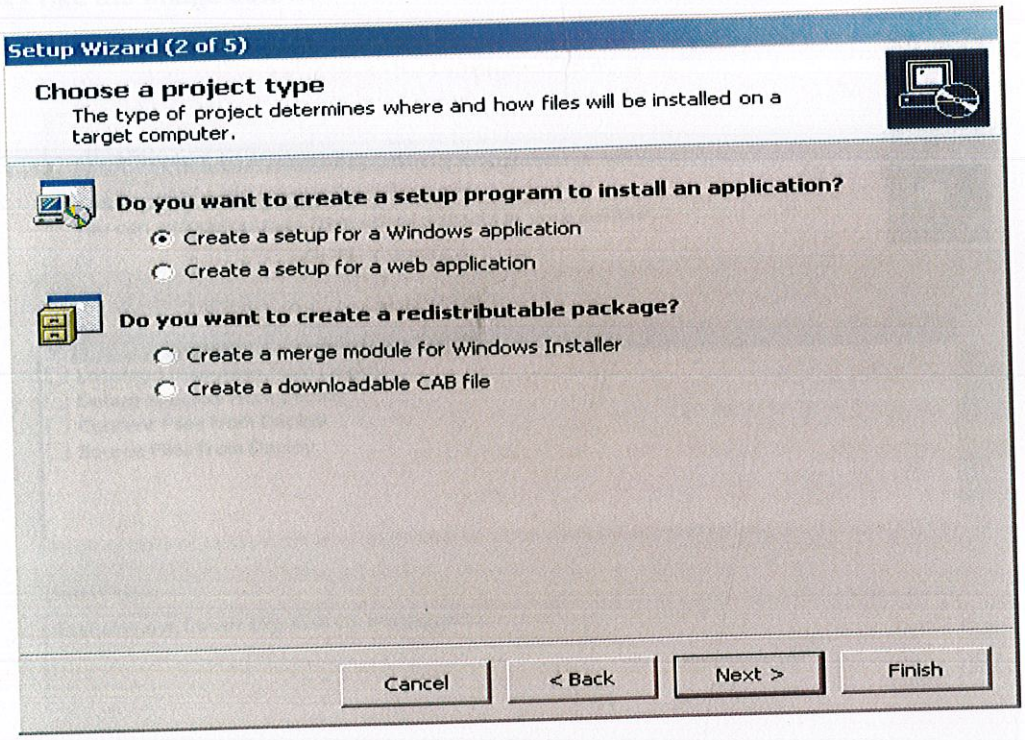

Figure 5.1(c): Deployment of project – STEP 3

Clicking next opens a new pane which has options like Deploying only primary output from the project or both the project and source code or content files. Check the checkbox which you want, in this case check the checkbox that says "Primary Output from Deploy" and click next.

It looks like the image below:

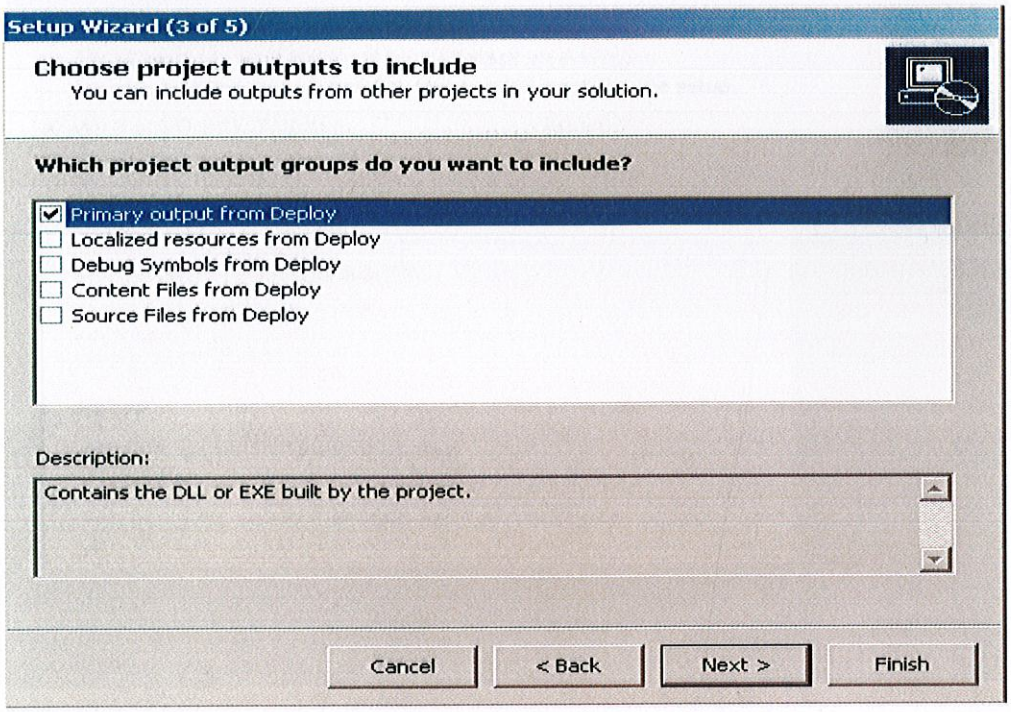

Figure 5.1(d): Deployment of project – STEP 4

Clicking next opens a new pane which asks if you want any additional files to be added. If you wish, you can include other files, like an icon for the application. In this example don't include any files and click next.

It looks like the image below:

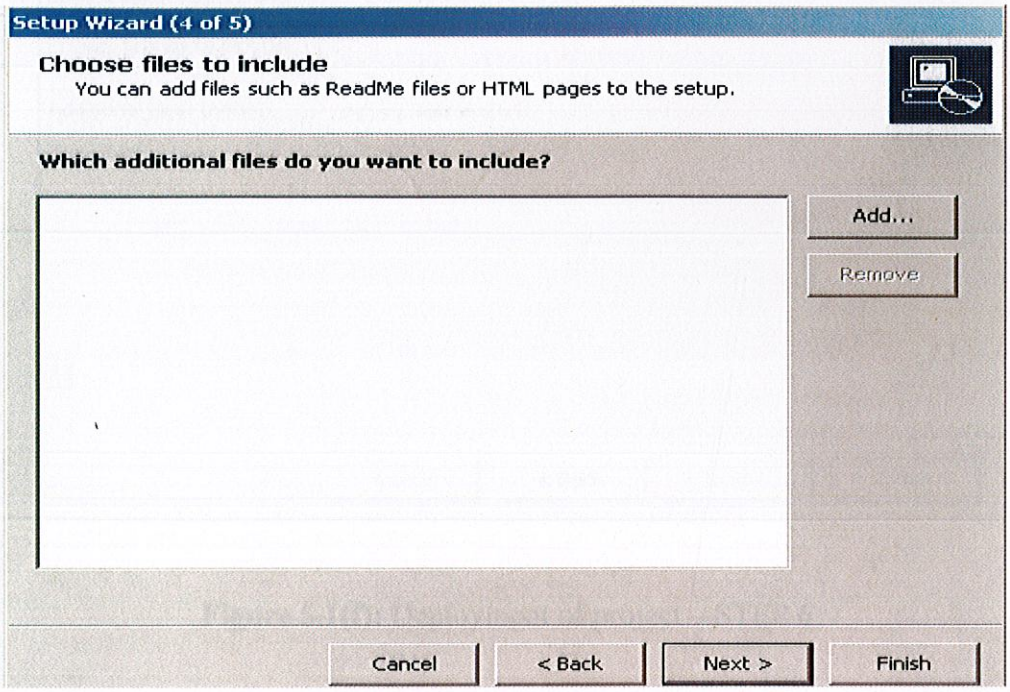

Figure 5.1(e): Deployment of project – STEP 5

Doing that brings up the last pane of the Setup Wizard which looks like the image below. Click Finish on this pane.

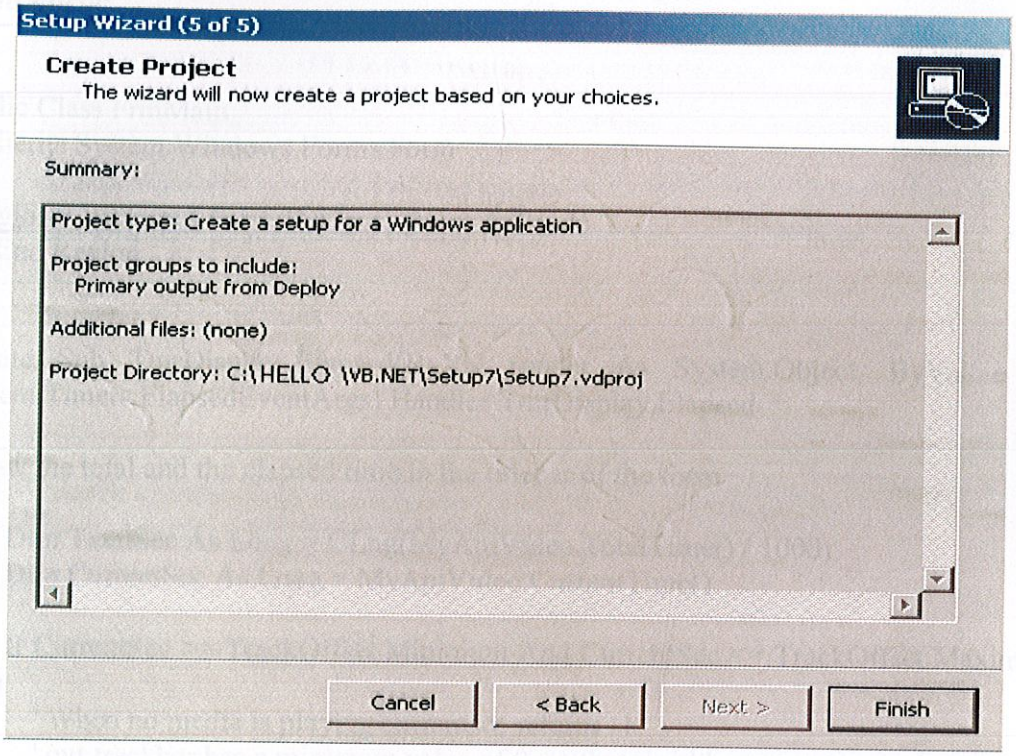

Figure 5.1(f): Deployment of project – STEP 6

Using the above mentioned procedure, we can easily make a setup file of our project, so that it can be installed on any windows platform for general use.

CODE:

Main Form:

```
Public Class FrmMain
  Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "
  #End Region


Private Sub TmrDisplay_Elapsed(ByVal sender As System.Object, ByVal e As
System.Timers.ElapsedEventArgs) Handles TmrDisplay.Elapsed

' Show the total and the elapsed time in the titlebar of the form

    Dim TotalSec As Long = CLng(MyApiVideo.TotalTime() / 1000)
    Dim CurrentSec As Long = MyApiVideo.CurrentTime()

    If CurrentSec >= TrackOffset.Minimum And CurrentSec <= TrackOffset.Maximum
Then
      ' When no media is playing currentsec returns -1
      ' our trackbar has a minimum value of 0, so that would cause an exception.
      TrackOffset.Value = CInt(CurrentSec)
    Else
      TrackOffset.Value = TrackOffset.Minimum
    End If

    CurrentSec = CLng(CurrentSec / 1000)
    Me.Text = "   Total Time = " & CStr(Int(TotalSec / 60)).PadLeft(2, CChar("0")) &
":" & CStr(CInt(TotalSec Mod 60)).PadLeft(2, CChar("0")) & "     Time Elapsed = " &
CStr(Int(CurrentSec / 60)).PadLeft(2, CChar("0")) & ":" & CStr(CInt(CurrentSec Mod
60)).PadLeft(2, CChar("0"))


  End Sub


Private Sub TrackOffset_Scroll(ByVal sender As Object, ByVal e As System.EventArgs)
Handles TrackOffset.Scroll

    ' Mute the audio whilst scrolling.
    ' (audio is muted on first scroll and restored after the scroll

    Dim Mute As ApiVideo.Channels = MyApiVideo.Mute
```

```
        MyApiVideo.Mute = ApiVideo.Channels.Both
        MyApiVideo.MoveToTime(TrackOffset.Value)
        MyApiVideo.Mute = Mute

    End Sub


    Private Sub ChkMute_CheckedChanged(ByVal sender As System.Object, ByVal e As
    System.EventArgs)              Handles              ChkMuteRight.CheckedChanged,
    ChkMuteLeft.CheckedChanged

        MyApiVideo.Mute = (ChkMuteLeft.Checked And ApiVideo.Channels.Left) Or
    (ChkMuteRight.Checked And ApiVideo.Channels.Right)

    End Sub


    Private Sub TrackVolume_Scroll(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles TrackVolume.Scroll

        ' range between 0 and 1000 where 500 is default
        MyApiVideo.Volume = TrackVolume.Value

    End Sub


    Private Sub TrackBalance_Scroll(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles TrackBalance.Scroll

        ' range between 0 and 1000 where 500 is both sides equal
        MyApiVideo.Balance = TrackBalance.Value

    End Sub


    Private Sub TrackSpeed_Scroll(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles TrackSpeed.Scroll

        ' range between 1 and 2000 where 1000 is normal
        MyApiVideo.Speed = TrackSpeed.Value

    End Sub
```

```
Private Sub BtnClose_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnClose.Click

    MyApiVideo.Close()
    Label1.Text = ""

End Sub


Private Sub BtnPlay_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnPlay.Click

    MyApiVideo.Play()

End Sub


Private Sub BtnPause_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnPause.Click

    MyApiVideo.Pause()

End Sub


Private Sub BtnStop_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnStop.Click

    MyApiVideo.Stop()

End Sub


Private Sub CheckRepeat_CheckedChanged(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles CheckRepeat.CheckedChanged

    MyApiVideo.Repeat = CheckRepeat.Checked

End Sub


Private Sub BtnOpen_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnOpen.Click

    Dim AskFile As New OpenFileDialog

    AskFile.AddExtension = True
    AskFile.CheckFileExists = True
```

```
        AskFile.CheckPathExists = True
        AskFile.Multiselect = False
        AskFile.ShowReadOnly = False

        If AskFile.ShowDialog(Me) = DialogResult.OK Then
            MyApiVideo.Open(AskFile.FileName)

            If MyApiVideo.TotalTime >= 0 Then
                Label1.Text = AskFile.FileName

                TrackOffset.Value = 0
                TrackOffset.Minimum = 0
                TrackOffset.Maximum = CInt(MyApiVideo.TotalTime)
            Else
                MsgBox("The file can not be opened. Please check the file extension",
MsgBoxStyle.OKOnly + MsgBoxStyle.Exclamation + MsgBoxStyle.DefaultButton1 +
MsgBoxStyle.SystemModal, "Error")
                Label1.Text = ""
            End If

            ' Play the file right away.
            MyApiVideo.Play()

        End If
        AskFile.Dispose()

    End Sub

End Class
```

ApiVideo Class:

```
Option Strict On

Public Class ApiVideo
   Inherits Control


' The ApiVideo class plays media using the mciSendString API,
' not using the Media Player Component


   ' MUTE Channels
   Public Enum Channels
      None = 0
      Left = 1
      Right = 2
      Both = 3
   End Enum

   Public Event OnEnd(ByVal sender As Object, ByVal e As System.EventArgs)

#Region "API"
#End Region

#Region "Variables"

Private pFileName As String     ' The media file currently open.
Private pAlias As String        ' Each instance of this control gets its own unique alias.
Private pLastError As MCIERR    ' The error returned by the last call to mciSendString.
Private pOpenSuccess As Boolean ' Indicates that the Open command was successful.
Private pSpeed As Integer       ' Playback speed (normal = 1000).
Private pMute As Channels       ' The channels that are muted.
Private pBalance As Integer     ' Left and right balance.
Private pVolume As Integer      ' Volume.
Private pRepeat As Boolean      ' Indicates playback is looping.
Private pTotalTime As Long      ' The length of the media in milliseconds.
Private pTotalFrames As Long    ' The length of the media in frames.
Private pClipStart As Long      ' The start frame or milliseconds of the play sequence.
Private pClipEnd As Long        ' The end frame or milliseconds of the play sequence.
Private pClipFormat As String   ' The time format for clipping, either "frames" or "msec".
Private pPlaying As Boolean     ' Indicates if we are playing or not.
Private pPaused As Boolean      ' Indicates if we are paused or not.

#End Region
```

```
Public ReadOnly Property FileName() As String
    Get
        ' file passed to the Open function.
        Return pFileName
    End Get
End Property


Public Function [Open](ByVal File As String) As Boolean

    ' Close the previous file.
    If pOpenSuccess = True Then
        Me.Close()
    End If

    ' Get the device type
    Dim Device_Type As String = "MPEGVideo"
    Dim MciExtension As Microsoft.Win32.RegistryKey =
Microsoft.Win32.Registry.LocalMachine.OpenSubKey("SOFTWARE\Microsoft\Windo
ws NT\CurrentVersion\MCI Extensions", False)
    If Not MciExtension Is Nothing Then
        Device_Type =
CStr(MciExtension.GetValue(Replace(System.IO.Path.GetExtension(File), ".", ""),
"MPEGVideo"))
    End If

    ' Try to open the file.
    pLastError = mciSendString("open """ & File & """ type " & Device_Type & " alias
" & pAlias & "  parent " & Me.Handle.ToString & " style child", vbNullString, 0,
IntPtr.Zero)
    If pLastError = MCIERR.MCIERR_NO_ERROR Then
        pOpenSuccess = True
        pPlaying = False
        pPaused = False
        pFileName = File

        SizeMediaWindow()
        DoSpeed()
        DoMute()
        DoBalance()
        DoVolume()

        pTotalTime = GetTotalTime()
        pTotalFrames = GetTotalFrames()
        Return True
    End If
    Return False
End Function
```

```
Public Function [Close]() As Boolean

    ' Close the media file.
    If pOpenSuccess = True Then
      pLastError = mciSendString("close " & pAlias, vbNullString, 0, IntPtr.Zero)
      If pLastError = MCIERR.MCIERR_NO_ERROR Then
        pOpenSuccess = False
        pPlaying = False
        pPaused = False
        pFileName = ""     ' There is no file open.
        pTotalTime = -1    ' No media.
        pTotalFrames = -1  ' No media.
        Return True
      End If
    End If
    Return False

End Function


Private Function [Play](ByVal WithClipStart As Boolean, ByVal WithClipEnd As
Boolean, ByVal WithPauseState As Boolean) As Boolean

    ' Start playing the media.
    If pOpenSuccess = True Then
      If pClipFormat.Length > 0 Then
        pLastError = mciSendString("set " & pAlias & " time format " & pClipFormat,
vbNullString, 0, IntPtr.Zero)
      End If
      pLastError = mciSendString("play " & pAlias & CStr(IIf((pClipStart <> -1) And
WithClipStart = True, " from " & CStr(pClipStart), "")) & CStr(IIf((pClipEnd <> -1) And
WithClipEnd = True, " to " & CStr(pClipEnd), "")) & " notify", vbNullString, 0,
Me.Handle)
      If pLastError = MCIERR.MCIERR_NO_ERROR Then
        pPlaying = True
        If WithPauseState = True And pPaused = True Then
          Me.Pause()
        Else
          pPaused = False
        End If
      End If
      Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function
```

```
Public Function [Stop]() As Boolean

    ' Stop the media.
    If pOpenSuccess = True Then
      pLastError = mciSendString("stop " & pAlias, vbNullString, 0, IntPtr.Zero)
      If pLastError = MCIERR.MCIERR_NO_ERROR Then
        ' After stop we rewind the media and get it to the start
        pLastError = mciSendString("seek " & pAlias & " to start", vbNullString, 0,
IntPtr.Zero)
      End If
      pPlaying = False
      pPaused = False
      Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function


Public Function [Pause]() As Boolean

    ' Pause the media.
    If pOpenSuccess = True Then
      pLastError = mciSendString("pause " & pAlias, vbNullString, 0, IntPtr.Zero)
      pPaused = (pLastError = MCIERR.MCIERR_NO_ERROR)
      Return pPaused
    End If
    Return False

End Function


Public Property Repeat() As Boolean
    Get
      Return pRepeat
    End Get
    Set(ByVal Value As Boolean)
      If pRepeat <> Value Then
        pRepeat = Value
      End If
    End Set
End Property


Public ReadOnly Property OriginalSize() As Size
    Get
      ' Obtain the original screen size of the media
      If pOpenSuccess = True Then
```

```
            Dim SizeStr As String = Space(128)
            pLastError = mciSendString("where " & pAlias & " source", SizeStr,
Len(SizeStr), IntPtr.Zero)
            If pLastError = MCIERR.MCIERR_NO_ERROR Then
              Dim Items() As String = Split(Trim(SizeStr), " ")
              Return New Size(CInt(Items(2)), CInt(Items(3)))
            End If
          End If
          Return New Size
        End Get
    End Property


    Public ReadOnly Property TotalFrames() As Long
        ' Returns the number of frames in the media or -1 when no
        ' media is opened or if the media doesn't support frames
        Get
          Return pTotalFrames
        End Get
    End Property


    Public ReadOnly Property TotalTime() As Long
        ' Returns the total time of the media in milliseconds.
        Get
          Return pTotalTime
        End Get
    End Property


    Public ReadOnly Property CurrentFrame() As Long
        ' Returns the current frame index in the media.
        Get
          If pOpenSuccess = True Then
            pLastError = mciSendString("set " & pAlias & " time format frames",
vbNullString, 0, IntPtr.Zero)
            If pLastError = MCIERR.MCIERR_NO_ERROR Then
              Dim PosStr As String = Space(128)
              pLastError = mciSendString("status " & pAlias & " position", PosStr,
Len(PosStr), IntPtr.Zero)
              Return CLng(Trim(PosStr))
            End If
          End If
          Return -1
        End Get
    End Property
```

```
    Public ReadOnly Property CurrentTime() As Long
        ' Returns the current time index (milliseconds) in the media.
        Get
            If pOpenSuccess = True Then
                pLastError = mciSendString("set " & pAlias & " time format milliseconds",
vbNullString, 0, IntPtr.Zero)
                If pLastError = MCIERR.MCIERR_NO_ERROR Then
                    Dim PosStr As String = Space(128)
                    pLastError = mciSendString("status " & pAlias & " position", PosStr,
Len(PosStr), IntPtr.Zero)
                    Return CLng(Trim(PosStr))
                End If
            End If
            Return -1
        End Get
    End Property


    Public Property Speed() As Integer
        ' Set or return the playback speed
        Get
            Return pSpeed
        End Get
        Set(ByVal Value As Integer)
            If Value >= 0 And Value <= 2000 And Value <> pSpeed Then
                pSpeed = Value
                DoSpeed()
            End If
        End Set
    End Property


    Public Property Mute() As Channels
        ' Mute the specified channels
        Get
            Return pMute
        End Get
        Set(ByVal Value As Channels)
            If Value >= Channels.None And Value <= Channels.Both And Value <> pMute
Then
                pMute = Value
                DoMute()
            End If
        End Set
    End Property
```

```
Public Property Balance() As Integer
    ' Sets or returns the left-right audio balance
    Get
      Return pBalance
    End Get
    Set(ByVal Value As Integer)
      If Value >= 0 And Value <= 1000 And Value <> pBalance Then
        pBalance = Value
        DoBalance()
      End If
    End Set
End Property


Public Property Volume() As Integer
    ' Sets or returns the audio volume level
    Get
      Return pVolume
    End Get
    Set(ByVal Value As Integer)
      If Value >= 0 And Value <= 1000 And Value <> pVolume Then
        pVolume = Value
        DoVolume()
      End If
    End Set
End Property


Public Function MoveToStart() As Boolean
    ' Move the media to its beginning
    If pOpenSuccess = True Then
      pLastError = mciSendString("seek " & pAlias & " to start", vbNullString, 0,
IntPtr.Zero)
      pPlaying = False
      pPaused = False
      Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function


Public Function MoveToEnd() As Boolean
    ' Move the media to its end.
    If pOpenSuccess = True Then
      pLastError = mciSendString("seek " & pAlias & " to end", vbNullString, 0,
IntPtr.Zero)
      pPlaying = False
```

```
            pPaused = False
            Return (pLastError = MCIERR.MCIERR_NO_ERROR)
        End If
        Return False

End Function


Public Function MoveToFrame(ByVal Frame As Long) As Boolean
        ' Move the media to the desired frame.
        Return MoveToPosition(Frame, "frames")

End Function


Public Function MoveToTime(ByVal Milliseconds As Long) As Boolean
        ' Move the media to the desired time index.
        Return MoveToPosition(Milliseconds, "milliseconds")

End Function


Private Function MoveToPosition(ByVal Index As Long, ByVal TimeFormat As String)
As Boolean
        ' Move the media to its desired index using the specified time format.
        If pOpenSuccess = True Then
            pLastError = mciSendString("set " & pAlias & " time format " & TimeFormat,
vbNullString, 0, IntPtr.Zero)
            If pLastError = MCIERR.MCIERR_NO_ERROR Then
                pLastError = mciSendString("seek " & pAlias & " to " & CStr(Index),
vbNullString, 0, IntPtr.Zero)
                If pLastError = MCIERR.MCIERR_NO_ERROR Then
                    If pPlaying = True Then
                        Me.Play(False, True, True)
                    End If
                End If
                Return (pLastError = MCIERR.MCIERR_NO_ERROR)
            End If
        End If
        Return False

End Function
```

```
Public Function ClipFrame(ByVal [Start] As Long, ByVal [End] As Long) As Boolean
    ' Clip the media to only play between the start and end frame.
    ' Set both values to -1 to undo clipping.
    Return Clip([Start], [End], Me.CurrentFrame, "frames")

End Function


Public Function ClipTime(ByVal [Start] As Long, ByVal [End] As Long) As Boolean
    ' Clip the media to only play between the start and end time.
    ' Set both values to -1 to undo clipping.
    Return Clip([Start], [End], Me.CurrentTime, "milliseconds")

End Function


Private Function Clip(ByVal [Start] As Long, ByVal [End] As Long, ByVal Current As
Long, ByVal TimeFormat As String) As Boolean

    If pOpenSuccess = True Then
      If [Start] <> pClipStart Or [End] <> pClipEnd Or TimeFormat <> pClipFormat
Then
        pClipStart = [Start]
        pClipEnd = [End]
        If pClipStart = -1 And pClipEnd = -1 Then
          pClipFormat = ""
        Else
          pClipFormat = TimeFormat
        End If

        ' We are playing so we need to apply the clip now.
        If pPlaying = True Then
        ' If we are currently positioned before the start of the clip, we skip to the start.
        ' If we are positioned in the clip range, or after it, MCI can handle it by itself.
          Me.Play(([Start] > Current And [Start] <> -1), True, True)
        End If
        Return True
      End If
    End If
    Return False

End Function
```

```
Private Function SizeMediaWindow() As Boolean
    ' Size the media to fit our window, preserving aspect ratio.
    If pOpenSuccess = True Then
      Dim OptimalSize As Size = Me.OriginalSize
      If OptimalSize.IsEmpty = False Then
        ' Calculate the ratio for width
        Dim wRatio As Double = (100 / OptimalSize.Width * Me.Width) / 100

        If OptimalSize.Height * wRatio > Me.Height Then
          wRatio = (100 / OptimalSize.Height * Me.Height) / 100
        End If
        ' Calculate the width and height for this ratio and Left and Top to center the
media.
        Dim wWidth As Integer = CInt(OptimalSize.Width * wRatio)
        Dim wHeight As Integer = CInt(OptimalSize.Height * wRatio)
        Dim wLeft As Integer = CInt((Me.Width - wWidth) / 2)
        Dim wTop As Integer = CInt((Me.Height - wHeight) / 2)

        pLastError = mciSendString("put " & pAlias & " window at " & CStr(wLeft) &
" " & CStr(wTop) & " " & CStr(wWidth) & " " & CStr(wHeight), vbNullString, 0,
IntPtr.Zero)
        Return (pLastError = MCIERR.MCIERR_NO_ERROR)
      End If
    End If
    Return False

End Function


Private Function DoSpeed() As Boolean
    ' Set the playback speed.
    If pOpenSuccess = True Then
      pLastError = mciSendString("set " & pAlias & " speed " & CStr(pSpeed),
vbNullString, 0, IntPtr.Zero)
      Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function


Private Function DoMute() As Boolean
    ' Mute the channels.
    If pOpenSuccess = True Then
      Select Case pMute
        Case Channels.None
          pLastError = mciSendString("set " & pAlias & " audio all on", vbNullString,
0, IntPtr.Zero)
```

```
            Case Channels.Both
                pLastError = mciSendString("set " & pAlias & " audio all off", vbNullString,
0, IntPtr.Zero)
            Case Channels.Left
                pLastError = mciSendString("set " & pAlias & " audio left off",
vbNullString, 0, IntPtr.Zero)
                pLastError = mciSendString("set " & pAlias & " audio right on",
vbNullString, 0, IntPtr.Zero)
            Case Channels.Right
                pLastError = mciSendString("set " & pAlias & " audio left on",
vbNullString, 0, IntPtr.Zero)
                pLastError = mciSendString("set " & pAlias & " audio right off",
vbNullString, 0, IntPtr.Zero)
        End Select
        Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function

Private Function DoBalance() As Boolean
    ' Set the balance factor.
    If pOpenSuccess = True Then
        pLastError = mciSendString("setaudio " & pAlias & " left volume to " &
CStr(1000 - pBalance), vbNullString, 0, IntPtr.Zero)
        pLastError = mciSendString("setaudio " & pAlias & " right volume to " &
CStr(pBalance), vbNullString, 0, IntPtr.Zero)
        Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function


Private Function DoVolume() As Boolean
    ' Set the volume factor.
    If pOpenSuccess = True Then
        pLastError = mciSendString("setaudio " & pAlias & " volume to " &
CStr(pVolume), vbNullString, 0, IntPtr.Zero)
        Return (pLastError = MCIERR.MCIERR_NO_ERROR)
    End If
    Return False

End Function
```

```
Private Function GetTotalFrames() As Long
    ' Return the total number of frames
    If pOpenSuccess = True Then
      pLastError = mciSendString("set " & pAlias & " time format frames",
vbNullString, 0, IntPtr.Zero)
      If pLastError = MCIERR.MCIERR_NO_ERROR Then
        Dim FrameStr As String = Space(128)
        pLastError = mciSendString("status " & pAlias & " length", FrameStr,
Len(FrameStr), IntPtr.Zero)
        If pLastError = MCIERR.MCIERR_NO_ERROR Then
          Return CLng(Trim(FrameStr))
        End If
      End If
    End If
    Return -1

End Function


Private Function GetTotalTime() As Long
    ' Return the total time in milliseconds
    If pOpenSuccess = True Then
      pLastError = mciSendString("set " & pAlias & " time format milliseconds",
vbNullString, 0, IntPtr.Zero)
      If pLastError = MCIERR.MCIERR_NO_ERROR Then
        Dim TimeStr As String = Space(128)
        pLastError = mciSendString("status " & pAlias & " length", TimeStr,
Len(TimeStr), IntPtr.Zero)
        If pLastError = MCIERR.MCIERR_NO_ERROR Then
          Return CLng(Trim(TimeStr))
        End If
      End If
    End If
    Return -1

End Function


Public Function GetLastError() As Integer
    ' Return the last MCI error code.
    Return pLastError

End Function

Public Function GetErrorString() As String
    ' Return a description for the last MCI error.
    If pLastError <> MCIERR.MCIERR_NO_ERROR Then
      Dim ErrorText As String = Space(128)
```

```
            If mciGetErrorString(pLastError, ErrorText, Len(ErrorText)) <> 0 Then
                Return Trim(ErrorText)
            End If
        End If

    End Function


    Public Sub New()
        ' Init all the variables to a default value jus like a constructor
        pFileName = ""                  ' No file loaded.
        pAlias = "ALIAS" & Me.Handle.ToString ' Create an unique alias (each instance of
    this control has an unique handle).
        pLastError = MCIERR.MCIERR_NO_ERROR   ' No error.
        pOpenSuccess = False              ' Not open.
        pSpeed = 1000                   ' Normal playback speed.
        pMute = Channels.None               ' No channels muted.
        pBalance = 500                  ' Normal left and right balance.
        pVolume = 500                    ' Normal volume.
        pRepeat = False                 ' Default to no playback looping.
        pTotalTime = -1                 ' No media.
        pTotalFrames = -1                ' No media.
        pClipStart = -1                ' Start at the beginning.
        pClipEnd = -1                  ' Play until the end.
        pClipFormat = ""                 ' No clip format.
        pPlaying = False                ' We are not playing.
        pPaused = False                 ' We are not paused.

    End Sub


    Protected Overrides Sub OnResize(ByVal e As System.EventArgs)
        ' Size the media window to the new size calculated
        SizeMediaWindow()

    End Sub


    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        Me.Close()
        MyBase.Dispose(disposing)

    End Sub

    End Class
```

REFRENCES:

- Books:
  - Visual Basic .NET Programming. By- Steven Holzner
  - Visual Basic.NET How to Program 2ed. By- Deitel
  - Thinking in Microsoft .NET
- Websites:
  - www.google.com
  - www.startvbdotnet.com
  - www.wikipedia.com
  - www.apple.com
  - www.msdn.microsoft.com
  - www.vbdotnetheaven.com