VOICE MASKING SYSTEM

By

Humera Ayub

(2000-NUST-BIT-241)

A project submitted in partial fulfillment of
the requirements for the degree of
Bachelors in Information Technology

In

NUST Institute of Information Technology
National University of Sciences & Technology
Rawalpindi Pakistan
(2004)

VOICE MASKING SYSTEM

By

Humera Ayub

(2000-NUST-BIT-241)

A project submitted in partial fulfillment of
the requirements for the degree of
Bachelors in Information Technology

In

NUST Institute of Information Technology
National University of Science and Technology
Rawalpindi, Pakistan

(2004)

ERP [illegible]

Certified that the contents and form of project entitled: **"Voice Masking System"** submitted by **Ms. Humera Ayub** have been found satisfactory for the requirements of the degree.

HUM
BIT-1

Supervisor: [illegible]

Mr. Nasir Mahmood

Member: [illegible] 6/1/04

Dr. Saeed Murtaza

Member: [illegible]

Mr. Kamran Shafi

Member: [illegible]

Mr. Mohammad Aatif

Member: [illegible]

Mr. Samran Afzal

Dedication

To my parents and my brother.

Preface

The central theme of this project is to study the different attributes and properties of sound, to study the methods for modification of voice, to learn advance programming techniques of Visual C++ and to come up a software which can be used to modify the sound and transmit it on Public Switched Telephone Network.

The work on this project began in November 2003. Major time was spent on studying digital audio processing software examples. Different voice masking techniques were also studied. Various digital sound processing libraries and their usage was also studied before making a feasible software design.

Voice Making System project can be used in game programming, cartoons and animated movies. This system can be used to replace voice masking hardware such as Voice Changer II . Enhanced versions of it can be implemented as a full fledged Enterprise Telephone Security System.

Humera Ayub

February 2004

Acknowledgements

I am greatly indebted to my Mr. Nasir Mahmood, my supervisor for his good guidance and encouragement throughout this project. I would like to thank him for the invaluable guidance and encouragement, which he has provided throughout my research. The calm with which he handled obstacles we faced during the course of research really helped a lot in keeping my enthusiasm high. This work would not have been possible but for his support.

Above all I would like to thank my parents for their absolute confidence in me, I would not have been able to pursue my BIT degree and this project without their support. They were, are and will always be the source of inspiration in all my endeavors.

Table of Contents

List of Abbreviations

API	Application Programming Interface
CD	Compact Disk
DFT	Discrete Fourier Transform
DLL	Dynamic Link Library
DSP	Digital Signal Processing
FFT	Fast Fourier Transform
Hz	Hertz
IO	Input Output
MSDN	Microsoft Development Network
PCM	Pulse Code Modulation
PSTN	Public Switched Telephone Network
RIFF	Resource Interchange File Format
SP	Service Pack
STFT	Short Time Fourier Transform
TAD	Telephone Answering Device
TAPI	Telephony Application Programming Interface
UI	User Interface
VC	Visual C
VoIP	Voice over IP
VMS	Voice Masking System

List of Figures

Abstract

This project discusses the real-time processing of digital audio. Voice Masking System is a software that allows the real-time modification of waveform audio. Thus, it allows the user to directly hear the effects of parameter settings. Simultaneously, it records the audio input from the microphone.

The software also has a dialup module which can be used to transmit the masked or unmasked voice on the telephone. The system works using Telephone Application Programming Interface TAPI for dial up purposes which is used to make and disconnect calls from PC to phone.

For voice masking, the system uses SoundTouch sound processing library which provides different functions for sound manipulation. In addition the software can also record voice from mike in wave format and can also play it. Enhanced version of the software can be used in game programming, animated movies, IP based voice chatting systems and intelligence agencies. By adding voice recognition features it can be used for home and enterprise telephone security.

We hope Voice Masking System will help the software developers and the IT students, who are interested in working on voice applications, to get better understanding of the domain.

INTRODUCTION

Voice masking actually means to mask the voice that is to convert the voice into some other voice by changing its frequency, pitch amplitude etc. The software solution provides a way to modify voices. The main theme of the project is to study voice and different techniques used to modify the voice.

Voice Masking System will enables digitized voice to be changed from high pitch to low from female to male or child voices (or vice versa). The opportunity to modify voices in real-time, as well as pre-recorded speech, provides a range of substantial benefits, allowing people to stay anonymous during communication, fit the person type they prefer as well as to understand each other in a more efficient way.

The enhanced version will provide benefits in both world-wide and in-house products, and will be useful in the wide range of applications such as VoIP based applications, for people to change their voices while chatting, conferencing, etc., could be perfect for private investigators, phone systems, and many entertainment applications. An opportunity to change voice rate provides a range of benefits to education programs, allowing easier and more convenient material adoption.

Voice masking is used extensively in game programming. Microsoft Xbox multiplayer games and Xbox live uses voice masking capabilities for online and multiplayer games. Within game-play, players arc now able to morph their voices to that of the character they are currently playing.

1.1 Scope Of The Project

The scope includes study of voice, to learn the advance programming techniques, to study the voice modification techniques and the development of software that provides,

1. Call dialing facilities.
2. Voice masking facilities.
3. Voice recording facilities.

1.2 Project Plan

The project basically consists of the following modules.

1. Building Dialup Software.
2. Voice Capturing from mike, recording and playing of voice.
3. Masking of voice.

1.2.1 Dialup Software

The first module in our project is building dialup software. For this purpose we will be using Microsoft Telephone API. Microsoft provides more than 100 functions as part of TAPI library. The TAPI version being used is 1.4 supporting PSTN (Public Switched Telephone Network) Telephony. This dialup software will be capable of making and disconnecting calls only.

1.2.2 Voice Capturing from Mike, Recording and Playing of Voice

The second module of our project will be related to capturing voice from mike and recording and playing of voice. For this purpose we will be using multimedia facilities provided by VC++. The voice will be saved in *wave* format. The input for this module will be the voice that is received from the Mike and it will be saved and played from our software.

1.2.3 Masking of Voice

The third module of our project will be masking of voice. In this module different techniques for masking of voice were studied. Different sound processing libraries were also studied to mask the input voice. The module will change the pitch of the wave file. The wave file can also be played and sound data can be sent on to the telephone line using the modem.

1.3 Requirements

Here we list the software and hardware requirements of our project.

Software

- A C/C++ compiler targetting the Win32 platform, preferably Microsoft Visual C++ 6.0 on Windows 98, 2K, ME, XP
- SP5 for Visual C++ 6.0
- Processor Pack 5 for Visual C++ 6.0
- SoundTouch Library for processing Sound / Wave files

Hardware

- A microphone and Speakers / Microphone with Headphones
- Voice Modem

- Sound Card
- Telephone Line
- Audio Cable / TAD Cable

1.4 System Functionality

- Initially the user will speak from mike and can save his voice as well
- The user can modify his voice using the mask voice slider
- Then the user will dial a phone number by entering a phone number and pressing the dial button
- Eventually user will listen the ringing and then voice of the person he has called.
- The user can start conversation using mike plugged into the sound card. The person receiving the call on telephone set will be hearing the masked voice.

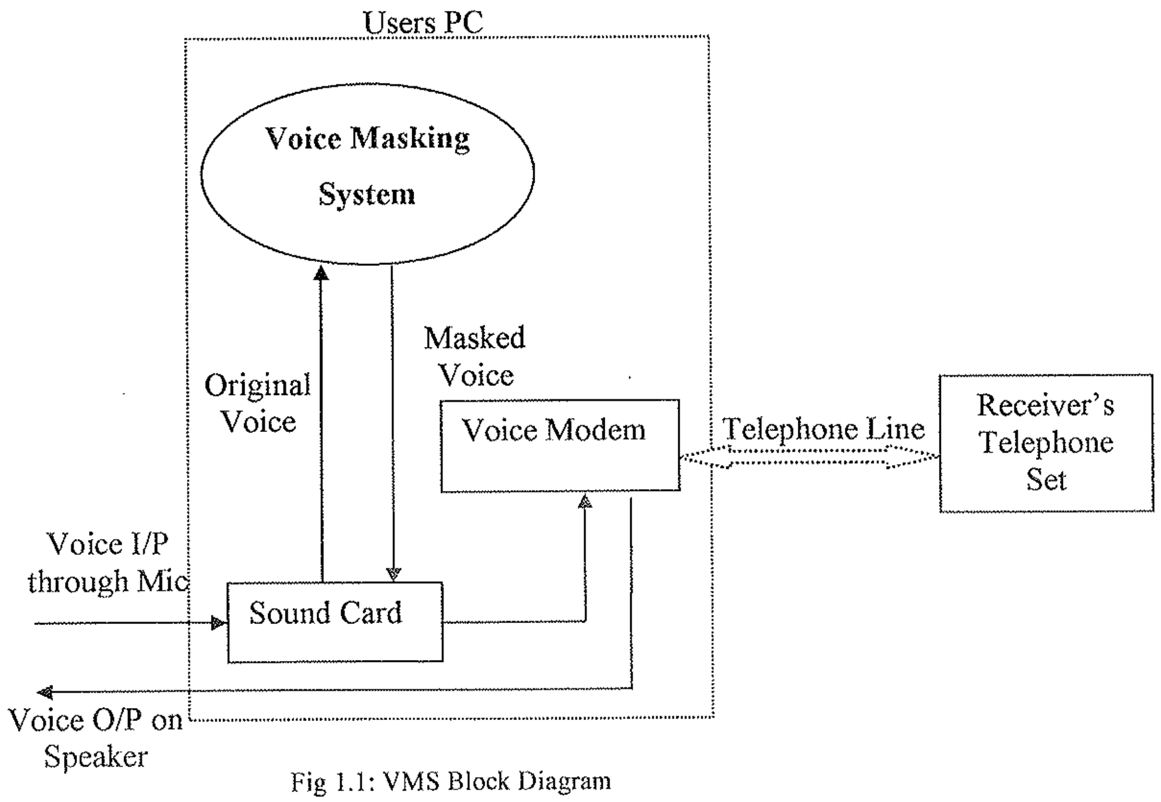


Fig 1.1: VMS Block Diagram

LITERATURE REVIEW

A lot of research is being carried out on digital audio processing and on modifying and analyzing voice using FFT, STFT, and Wavelet Transform [29] etc. Microsoft has also been working on developing a real time voice masking algorithm. A lot of research is being carried out by International Audio Engineering Society [27] and IEEE Signal Processing Society [28].

Work on VMS started off by consulting various mailing lists as to how to go by the topic and the project. As for programming Microsoft's Development Network library and various books on C programming are consulted. The names of the books [17] are given in the reference.

The background study and the literature review have been done in the domains as under:

- Sound / Characteristics of Sound [14]
- PCM / Sampling Sound [19]
- Wave Formats (RIFF etc.) [4]
- Multimedia File IO Functions [18]

Technology Learning [17]

- Windows programming
- Creating DLLs
- Win32 Application
- Multithreading
- Message Maps

Masking of voice

- Different techniques for modification of the voice signals like FFT [3], STFT [20], Wavelet Transforms [29] etc.
- Digital Sound Processing Libraries [21]

2.1 Introduction to digital audio

The most common type of digital audio [5] recording is called pulse code modulation (PCM). Pulse code modulation is what compact discs and most WAV files use. In PCM recording hardware, a microphone converts a varying air pressure (sound waves) into a varying voltage. Then an analog-to-digital converter measures (samples) the voltage at regular intervals of time. For example, in a compact disc audio recording, there are exactly 44,100 samples taken every second. Each sampled voltage gets converted into a 16-bit integer. A CD contains two channels of data: one for the left ear and one for the right ear, to produce stereo. The two channels are independent recordings placed "side by side" on the compact disc. (Actually, the data for the left and right channel alternate...*left, right, left, right,* ... like marching feet.)

The data that results from a PCM recording is a function of time. It often amazes people that a sequence of millions of integers on a compact disc recording can yield music and speech. People tend to wonder, "How can a stream of numbers sound like an entire orchestra?" It seems magical, and it is! Yet the magic is not in the digital recording; it's in your ear and your brain. To understand why this is true, imagine that you could place a microscopic movie camera in your ear to film your ear drum in slow motion. Suppose the movie camera was so fast that it could take a picture every 1/44,100 of a second. Also, suppose that the images this camera captured on film were so crisp and sharp that you could discern 65,536 (64K) distinct positions of the ear drum's surface as it moved back and forth in response to incoming sound waves. If you used this hypothetical technology to film your ear drum while listening to your best friend saying your name, then took the resulting movie and wrote down the numeric position of your ear drum in every frame of the movie, you would have a digital PCM recording. If you could later make your ear drum move back and forth in accordance with the thousands of numbers you had written down, you would hear your friend's voice saying your name exactly as it sounded the first time. It really doesn't matter what the sound is - your friend, a crowded party, a symphony - the concept still holds. When you hear more than one thing at a time, all the distinct sounds are physically mixed together in your ears as a single pattern of varying air pressure. Your ears and your brain work together to analyze this signal back into separate auditory sensations.

2.2 Resource Interchange File Format Services (RIFF)

RIFF [21] is a file format for storing many kinds of data, primarily multimedia data like audio and video. It is based on chunks and sub-chunks. Each chunk has a type, represented by a four-character tag. This chunk type comes first in the file, followed by the size of the chunk, then the contents of the chunk.

The entire RIFF file is a big chunk that contains all the other chunks. The first thing in the contents of the RIFF chunk is the "form type," which describes the overall type of the file's contents. So the structure of a RIFF file looks like this:

Offset (hex)	Contents
0000	'R', 'I', 'F', 'F'
0004	Length of the entire file - 8 (32-bit unsigned integer)
0008	form type (4 characters)

000C	first chunk type (4 character)
0010	first chunk length (32-bit unsigned integer)
0014	first chunk's data ... ...

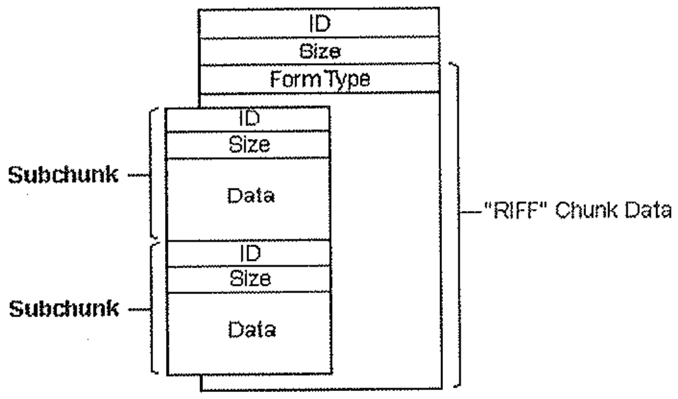


Fig 2.1: RIFF Chunk

All integers are stored in the Intel low-high byte ordering (usually referred to as "little-endian"). "RIFF" chunks include an additional field in the first four bytes of the data field. This additional field provides the form type of the field. The form type is a four-character code identifying the format of the data stored in the file. For example, Microsoft waveform-audio files have a form type of "WAVE".

2.3 Wave File Format

The WAVE file format is a subset of Microsoft's RIFF specifications, which can include lots of different kinds of data. It was originally intended for multimedia files, but the specifications is open enough to allow pretty much anything to be placed in such a file, and ignored by programs that read the format correctly.

The WAVE format is a subset of RIFF used for storing digital audio. Its form type is "WAVE", and it requires two kinds of chunks:

- fmt chunk, which describes the sample rate, sample width, etc., and

```
000C   first chunk type (4 character)
0010   first chunk length (32-bit unsigned integer)
0014   first chunk's data   ...   ...
```

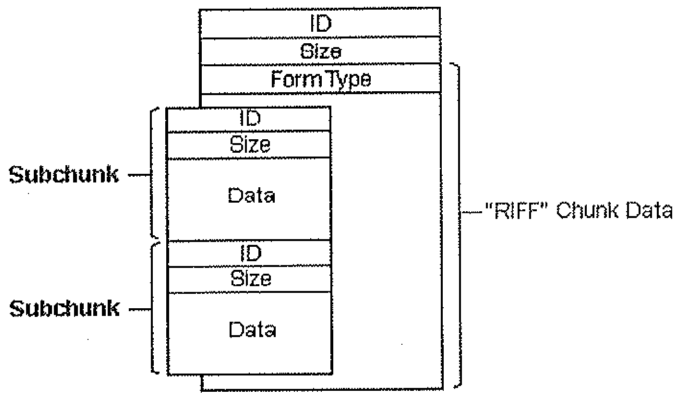


Fig 2.1: RIFF Chunk

All integers are stored in the Intel low-high byte ordering (usually referred to as "little-endian"). "RIFF" chunks include an additional field in the first four bytes of the data field. This additional field provides the form type of the field. The form type is a four-character code identifying the format of the data stored in the file. For example, Microsoft waveform-audio files have a form type of "WAVE".

2.3 Wave File Format

The WAVE file format is a subset of Microsoft's RIFF specifications, which can include lots of different kinds of data. It was originally intended for multimedia files, but the specifications is open enough to allow pretty much anything to be placed in such a file, and ignored by programs that read the format correctly.

The WAVE format is a subset of RIFF used for storing digital audio. Its form type is "WAVE", and it requires two kinds of chunks:

- fmt chunk, which describes the sample rate, sample width, etc., and

- data chunk, which contains the actual samples.

WAVE can also contain any other chunk type allowed by RIFF, including LIST chunks, which are used to contain optional kinds of data such as the copyright date, author's name, etc. Chunks can appear in any order.

2.4 The Fourier Transform as a Mathematical Concept

The Fourier Transform [3] is based on the discovery that it is possible to take any periodic function of time $x(t)$ and resolve it into an equivalent infinite summation of sine waves and cosine waves with frequencies that start at 0 and increase in integer multiples of a base frequency $f_0 = 1/T$, where T is the period of $x(t)$. Here is what the expansion looks like:

$$x(t) = a_0 + \sum_{k=1}^{\infty}\left(a_k \cos\left(2\pi k f_0 t\right) + b_k \sin\left(2\pi k f_0 t\right)\right)$$

An expression of the form of the right hand side of this equation is called a Fourier Series. The job of a Fourier Transform is to figure out all the a_k and b_k values to produce a Fourier Series, given the base frequency and the function $x(t)$. You can think of the a_0 term outside the summation as the cosine coefficient for k=0. There is no corresponding zero-frequency sine coefficient b_0 because the sine of zero is zero, and therefore such a coefficient would have no effect.

Of course, we cannot do an infinite summation of any kind on a real computer, so we have to settle for a finite set of sines and cosines. It turns out that this is easy to do for a digitally sampled input, when we stipulate that there will be the same number of frequency output samples as there are time input samples. Also, we are fortunate that all digital audio recordings have a finite length. We can pretend that the function $x(t)$ is periodic, and that the period is the same as the length of the recording. In other words, imagine the actual recording repeating over and over again indefinitely, and call this repeating function $x(t)$. The math for the FFT then becomes simpler, since it will start with the base frequency f_0 which spans one wavelength over the width of the recording. In other words, f_0 = samplingRate / N, where N is the number of samples in the recording.

2.4.1 Applications of the FFT

The FFT algorithm tends to be better suited to analyzing digital audio recordings than for filtering or synthesizing sounds. A common example is when you want to do the software equivalent of a device known as a spectrum analyzer, which electrical engineers use for displaying a graph of the frequency content of an electrical signal. You can use the FFT to determine the frequency of a note played in recorded music, to try to recognize different kinds of birds or insects, etc. The FFT is also useful for things which have nothing to do with audio, such as image processing (using a two-dimensional version of the FFT). The FFT also has scientific/statistical applications, like trying to detect periodic fluctuations in stock prices, animal populations, etc. FFTs are also used in analyzing seismographic information to take "sonograms" of the inside of the Earth.

The main problem with using the FFT for processing sounds is that the digital recordings must be broken up into chunks of n samples, where n always has to be an integer power of 2. One would first take the FFT of a block, process the FFT output array (i.e. zero out all frequency samples outside a certain range of frequencies), then perform the inverse FFT to get a filtered time-domain signal back. When the audio is broken up into chunks like this and processed with the FFT, the filtered result will have discontinuities which cause a clicking sound in the output at each chunk boundary. For example, if the recording has a sampling rate of 44,100 Hz, and the blocks have a size n = 1024, then there will be an audible click every 1024 / (44,100 Hz) = 0.0232 seconds, which is extremely annoying to say the least.

2.5 The Short Time Fourier Transform

Any sampled signal can be represented by a mixture of sinusoid waves, which is called partials. Besides the most obvious manipulations that are possible based on this representation, such as filtering out unwanted frequencies, we will see that the "sum of sinusoids" model can be used to perform other interesting effects as well. It appears obvious that once we have a representation of a signal that describes it as a sum of pure frequencies, pitch shifting must be easy to implement. As we will see very soon, this is almost true.

To understand how to go about implementing pitch shifting in the "frequency domain", we need to take into account the obvious fact that most signals we encounter in practice, such as speech or music, are changing over time. Actually, signals that do not change over time sound very boring and do not provide a means for transmitting meaningful auditory information. However, when we take a closer look at these signals, we will see that while they appear to be changing over time in many different ways with regard to their spectrum, they remain almost constant when we only look at small "excerpts", or "frames" of the signal that are only several milliseconds long. Thus, we can call these signals "short time stationary", since they are almost stationary within the time frame of several milliseconds.

Because of this, it is not sensible to take the Fourier transform of our whole signal, since it will not be very meaningful because all the changes in the signals' spectrum will be averaged at each chunk boundary. For example, if the recording has a sampling rate of 44,100 Hz, and the blocks have a size n = 1024, then there will be an audible click every 1024 / (44,100 Hz) = 0.0232 seconds, which is extremely annoying to say the least.

2.5 The Short Time Fourier Transform

Any sampled signal can be represented by a mixture of sinusoid waves, which is called partials. Besides the most obvious manipulations that are possible based on this representation, such as filtering out unwanted frequencies, we will see that the "sum of sinusoids" model can be used to perform other interesting effects as well. It appears obvious that once we have a representation of a signal that describes it as a sum of pure frequencies, pitch shifting must be easy to implement. As we will see very soon, this is almost true.

To understand how to go about implementing pitch shifting in the "frequency domain", we need to take into account the obvious fact that most signals we encounter in practice, such as speech or music, are changing over time. Actually, signals that do not change over time sound very boring and do not provide a means for transmitting meaningful auditory information. However, when we take a closer look at these signals, we will see that while they appear to be changing over time in many different ways with regard to their spectrum, they remain almost constant when we only look at small "excerpts", or "frames" of the signal that are only several milliseconds long. Thus, we can call these signals "short time stationary", since they are almost stationary within the time frame of several milliseconds.

Because of this, it is not sensible to take the Fourier transform of our whole signal, since it will not be very meaningful because all the changes in the signals' spectrum will be averaged together and thus individual features will not be readily observable. If we, on the other hand, split our signal into smaller "frames", our analysis will see a rather constant signal in each frame. This way of seeing our input signal sliced into short pieces for each of which we take the DFT is called the "Short Time Fourier Transform" (STFT) of the signal.

2.6 Telephony API Overview

The Telephony Application Programming Interface (TAPI) [23] provides a uniform set of commands for any supported telephony device that is connected to your computer

2.6.1 How TAPI works

When you use a Windows program to send faxes, connect to a telephone, make a call using IP, join a conference, or perform other TAPI-supported activities, there are three layers of software that enable you to use a telephony device: an application program, TAPI, and a TAPI service provider.

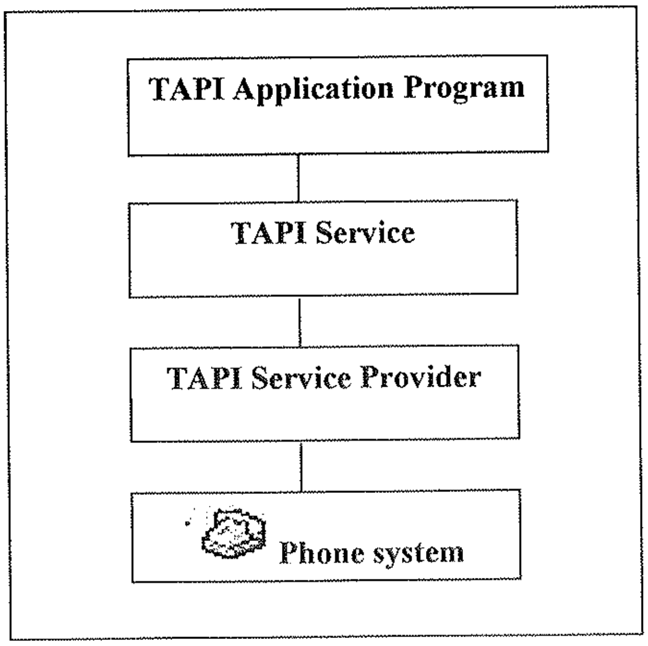


Fig 2.2: TAPI Architecture

An application program enables you to make phone calls, send and receive data or faxes, or Join conferences. Microsoft and independent software vendors provide application programs that incorporate TAPI functionality. Examples include Phone Dialer, HyperTenninal, and Fax Service included with the Windows 2000 operating system. TAPI provides telephony functions for application programs, such as dialing. A TAPI service provider translates the commands for a telephony device or telephony protocol. TAPI service providers for modems and several telephony protocols are installed with the Windows operating system, and others are provided by independent hardware vendors.

2.6.2 Why Use TAPI?

Telephony application programming interface is used if we want to use computer for following purposes.

- Use a computer as an answering machine or voice mailbox.
- Use a computer to send and receive faxes.
- Use a microphone and speaker or a speakerphone connected to a multimedia computer as a phone system.
- Log on to a computer from a remote location.

These solutions are typically supported by a direct connection from a desktop computer to a phone line with a modem.

Corporate organizations use TAPI based applications due to following reasons.

- Use a computer either as a multiline phone system or as a multifunction PBX controller.
- Provide a voice-mail system across your entire enterprise.
- Provide a fax system across your entire enterprise.

- Provide on-demand audio information services that allows callers to retrieve prerecorded or computer-generated text-to-speech information.
- Fax information to customers on demand , based on touch-tone keys or other input.
- Deliver database information related to a call (such as a customer profile or account information) at the same time it is switched to a desktop.
- Create a single, easy-to-use client application to manage all communications, such as voice, pager, e-mail, and conferencing.
- Provide dial-up access to the network.

2.6.3 TAPI Phone Devices

TAPI supports three types of phone devices, that is to say a device that can be taken off the hook or placed on hook by the users and has a microphone and speaker or earphone. The supported devices are :

- Handset, something you would recognize as a traditional telephone set
- Speakerphone, a loud speaking telephone set used for hands free calls. Not suitable for use in shared or noisy environments.
- Headset, used for hands free calls, convenient for people spending long periods of time communicating by telephone.

All of these features are supported by TAPI but obviously depend upon the telephony hardware and the TSP capabilities. An application can request from the TSP the capabilities of the telephony device. Do you support a "Speaker Phone", do you support a "Headset" etc. this allows the application to adapt itself depending on the supported feature set.

Chapter 3

SYSTEM DESIGN

System design is the process of developing a plan for an improved system based on the results of literature review and research. This chapter specifies the high level design of the Voice Masking System that I intend to develop. I will present here the use case and class diagram of the system.

3.1 Use Case Diagram

Use case diagrams depict the functional capabilities of the system. They detail the way in which different types of end users or external software components (actors) interact with the system. Use cases are an analysis artifact, not a design artifact and as such, do not depict the inner working of the system.

> "A use case is a specific way of using the system by using some part of its functionality. Each use case constitutes a complete course of events initiated by an actor and its specifies the interaction that takes place between an actor and the system.
>
> ...
>
> The collected use cases specify all the existing ways of using the system"
>
> Jacobson

MaskVoice Usecase Specification

Actor action	System Response
1) The user will move the slider control to mask his/her voice.	2) System can send the masked voice on line and can also play it back.

StartRecording Usecase Specification

Actor action	System Response
1) The user will press the recording button and will speak in mike	2) System will save the voice of the user in wave format. (masked or unmasked)

StopRecording Usecase Specification

Actor action	System Response
1) The user will press the stop recording button.	2) System will stop recording the voice.

PlaySound Usecase Specification

Actor action	System Response
1) The user will press the play button for playing sound.	2) System will play the recorded sound.

OpenSoundFile Usecase Specification

Actor action	System Response
	1) System will open the sound file for playing or recording of sound.

CloseSoundFile Usecase Specification

Actor action	System Response
	1) System will close the sound file.

WriteSoundData Usecase Specification

Actor action	System Response
	1) System will write the sound data in wave format on hard disk

ReadSoundFile Usecase Specifiction

Actor action	System Response
	1) System will read sound file for playing.

3.2 Domain Model

A "problem domain" refers to the real world things and concepts pertaining to the problem that the system being designed, is to solve. Domain modeling is the task of discovering the objects (classes) that represent those things and concepts. Included in this chapter is a class diagram that represents the static view of the problem domain.

The domain model focuses on what the system will do, not how it will do it.

3.2.1 Domain Class Diagram

The diagram below shows the domain level classes needed by the system.

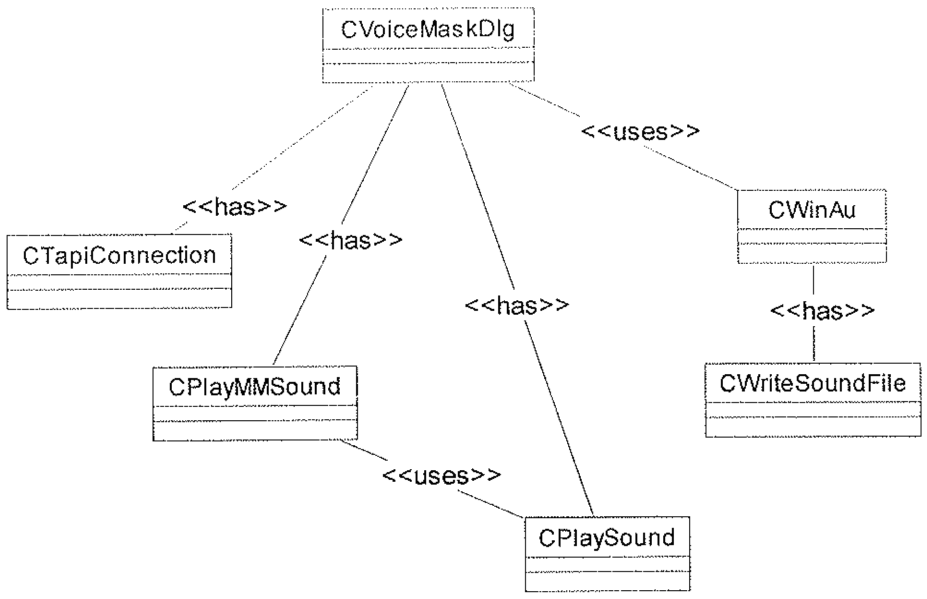


Fig 3.2: Class diagram of Voice Masking System

3.2.2 Class Definitions

A brief description of the classes depicted in the Domain Class diagram follows:

CTapiConnection

CTapiConnection class supports the basic functionality needed to use the Microsoft Windows® Telephony Application Programming Interface (TAPI) to automatically dial the telephone for a voice connection. The class provides the most basic TAPI functionality of initialization, ability

The domain model focuses on what the system will do, not how it will do it.

3.2.1 Domain Class Diagram

The diagram below shows the domain level classes needed by the system.

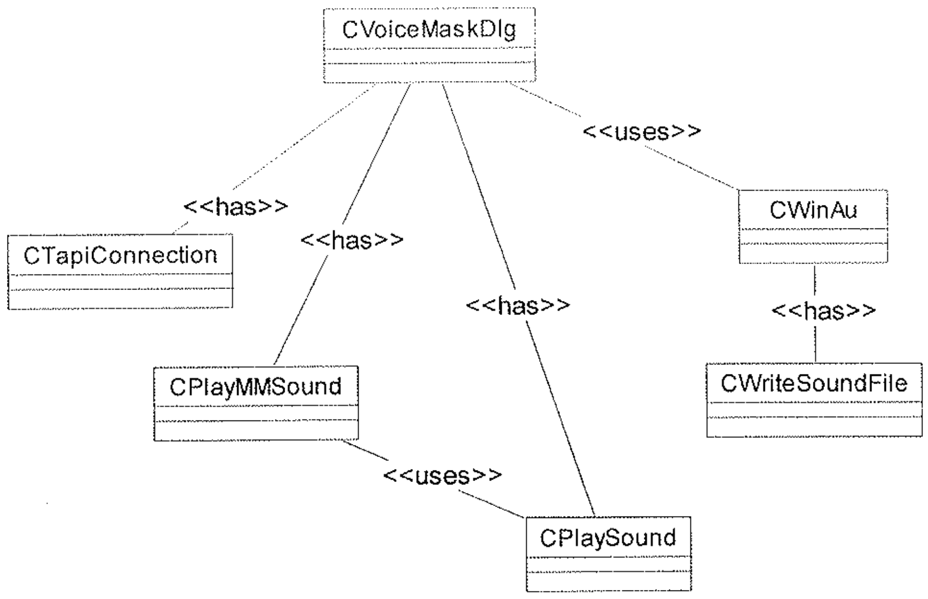


Fig 3.2: Class diagram of Voice Masking System

3.2.2 Class Definitions

A brief description of the classes depicted in the Domain Class diagram follows:

CTapiConnection

CTapiConnection class supports the basic functionality needed to use the Microsoft Windows® Telephony Application Programming Interface (TAPI) to automatically dial the telephone for a voice connection. The class provides the most basic TAPI functionality of initialization, ability to obtain a line, dial voice calls, drop a line, and shut down. I have made a DLL of this Class and will use the DLL in our application

CWinAu

It implements and handles Audio API and calls SoundTouch library functions. This class will provide facility for recording voice from mike. It takes voice input from microphone and sends sound buffers to the `CWriteSoundFile` class to write them to a wave(.wav) file. `MMWimData()` opens the wave input device, creates header, prepares buffers for input and invokes the message of the `CWriteSoundFile` class to creates sound file onto which sound data will be written. The class also provides other message handlers for Window messages like `MM_WIM_OPEN, MM_WIM_CLOSE, MM_WOM_OPEN, MM_WOM_DONE, MM_WOM_CLOSE` etc. The class also provides a method `PShiftAudioSignal()` for shifting the pitch so as to mask the sound buffers.

CPlayMMSound

This class will read the wave file and play to the output device. It contains one important user defined message `WM_PLAYMMSOUND_PLAYFILE` along with three other messages. `PlaySoundFile()` is the message handler for this message. This function opens the .wav file and verifies the .wav format of the sound file. After verification new thread is started which call the `PlaySound()` function. `PlaySound()` invokes the message of `CPlaySound` which opens the wave output device and then plays the sound to the output device.

CPlaySound

This class will provide functionality for playing the recorded voice file. It works along with `CPlayMMSound` class and implements wave output functions. This class receives messages from `CPlayMMSound` and provides implementation for the message handler functions. Important handlers are `OnStartPlaying`, `OnStopPlaying`, `OnWriteSoundData`, `OnEndPlaySoundData`.

CWriteSoundFile

`CWriteSoundFile` receives sound buffers from `CWinAu` and writes them to a wave disk file. It implements four user defined messages. One for creating the sound file. Second for writing to a sound file and third for properly closing the file.

CVoiceMaskDlg

This class is inherited from `CDialog` class and provides a user friendly UI. It provides functionalities for masking, recording, playing and dialing.

IMPLEMENTATION

This project is made in Visual C++ 6.0, with Microsoft Visual Studio as the development environment. The application uses MFC framework for creating a dialog based GUI and Windows and Multimedia messages for dealing and manipulating the wave data. The initial part of this chapter presents a brief overview of the MFC architecture and also describes the integration of the various parts of the software, some important Methods (functions) along with code snippets to explain the software implementation. The later part of this chapter describes the three main modules i.e the Dialup Module, the Recording and Playing Module and the Masking module.

4.1 MFC Architecture

MFC is the C++ class library Microsoft provides to place an object-oriented wrapper around the Windows API. MFC version contains nearly 200 classes, some of which you will use directly and others of which will serve primarily base classes for classes of your own. In an MFC program, you rarely need to call the Windows API directly. Instead, you create objects from MFC classes and call member functions belonging to those objects. Many of the hundreds of member functions defined in the class library are thin wrappers around the Windows API and even have the same names as the corresponding API functions. MFC is not only a library of classes. MFC is also an application framework. More than merely a collection of classes, MFC helps define the structure of an application and handles many routine chores on the application's behalf. Starting with CWinApp, the class that represents the application itself, MFC encapsulates virtually very aspect of a program's operation. The framework supplies the WinMain() function, and WinMain() in turn calls the application object's member functions to make the program go. One of the CWinApp member functions called by WinMain() is Run() that encapsulates the message loop that literally runs the program. The framework also provides abstractions that go above and beyond what the Windows API has to offer. For example MFC Document/View architecture builds a powerful infrastructure on top of the API that separates a program's data from the graphical representation, or use, of that data. Such abstractions are totally foreign to the API and don't exist outside the framework of MFC.

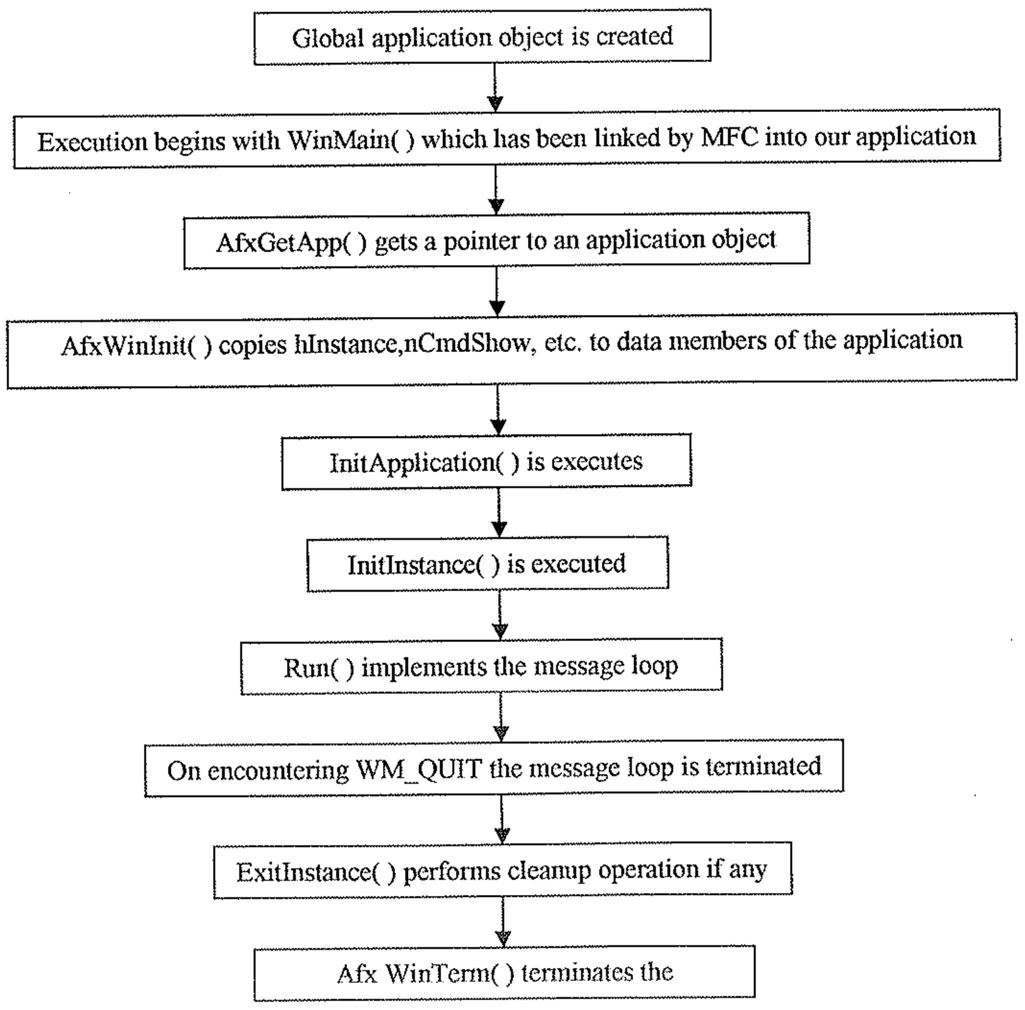

Global application object is created
Execution begins with WinMain( ) which has been linked by MFC into our application
AfxGetApp( ) gets a pointer to an application object
AfxWinInit( ) copies hInstance,nCmdShow, etc. to data members of the application
InitApplication( ) is executes
InitInstance( ) is executed
Run( ) implements the message loop
On encountering WM_QUIT the message loop is terminated
ExitInstance( ) performs cleanup operation if any
Afx WinTerm( ) terminates the

Fig 4.1: MFC based Windows program execution

In fact not all of the functions that MFC offers are members of classes. MFC provides a set of functions whose names begin with `Afx`. Class member functions can be called only in the context of the objects of which they belong., but `Afx` functions are available to everyone.

4.2 Implementation Details

In a dialog based application MFC generates two main classes, the application class (`CVoiceMaskApp`) and the dialog class (`CVoiceMaskDlg`). The application class has a global application object named `theApp` which calls the `WinMain()`, which in turn calls the `AfxGetApp()` and then all the functions are called as shown in fig 4.1. Since VMS is a dialog bases application, the `CVoiceMaskDlg` object gets instantiated in the `InitInstance()` of the `CVoiceMaskApp` class.

4.2.1 Functionality behind the dialog

VMS is a dialog based application. The main dialog gets instantiated when the `OnInitDialog( )` function is called from the application class. This function sets the minimum and maximum ranges of the slider control for masking voice and calls the function to create `PlayMMSound` and `PlaySound` threads.

The details of the message handlers for the dialog controls is as follows:

- Start Recording Button
  - Function : `OnBnRecord`
  - Description : Calls save dialog for entering wave file name which is used to used to create and record. It first calls the user defined dialog and asks for the file name as follows:

```
result = saveDlg->DoModal();
```

From here the control is passed onto the `CSaveDlg` class which gets the file name, stores it to an extern variable which is then used by the `MMWimData( )` function to create the file

- Play Button
    - Function : `OnBnPlay`
    - Description : Gets the file name to be played from the open file dialog by the following call

```
    if(openFileDlg.DoModal() == IDOK)
    {
         fName = openFileDlg.GetFileName();
         f = new TCHAR[fName.GetLength()+1];
         strcpy(f, fName);
     }
```

fName now has the name of the file to be played and then sends the file name to `WM_PLAYMMSOUND_PLAYFILE` message of `CPlayMMSound` class

- Edit Control
    - Function : `OnChangeEphoneno`
    - Description : As something is written into the edit box the dial button is enabled as shown in the following code snippet:

```
GetDlgItem(IDC_BN_DIAL)->EnableWindow(TRUE);
```

If the edit box is empty then the dial button is disabled as shown below:

```
if(m_CSPhoneNo.GetLength()==0)
  GetDlgItem(IDC_BN_DIAL)->EnableWindow(FALSE);
```

- Dial Button
  - Function : `OnBnDial`
  - Description : copies the contents (telephone number) of the edit box to the local variable *`buf`* and then passes it onto the `DialCall( )` function by making use of an object of class `CTapiConnection`. If the functions fails the "Dialing Error" message is displayed

```
     strcpy(buf , m_CSPhoneNo);
          if(!m_connection.DialCall(buf)) {
           MessageBox("Dialing Error"); }
```

- Disconnect Button
  - Function : `OnBnDisconnect`
  - Description : Calls function to disconnect the telephone call through a `m_connection` (object of class `CTapiConnection`) which is a member variable of the `CVoiceMaskDlg` as follows

```
m_connection.HangupCall();
```

- Clear Button
  - Function : `OnBnClear`
  - Description : Clears the contents of the edit box by following code

```
m_CSPhoneNo = "";
```

- Zero to Nine Buttons
  - Function : `OnBn0` to `OnBn9`

- Description : These ten buttons append the respective digit to the edit box variable an example of digit "9" is shown as follows:

```
m_CSPhoneNo = m_CSPhoneNo + "9";
```

- Slider Control
  - Function : `OnVScroll`
  - Description : The framework calls this member function when the user clicks the window's vertical scroll bar of the slider and sets the value of semitones as follows:

```
SemiToneValue = -m_PitchSlider.GetPos();
```

4.2.2 Multimedia and Windows Messages

VoiceMask2.cpp is the helping file which provides the functions to register the class when constructor of the dialog is called. While registering the window class, it is passed a pointer to the a `CALLBACK` window procedure for handling windows and multimedia messages.

The messages and their description is as follows:

`WM_CREATE`

This message is sent when an application requests that a window be created by calling the `CreateWindowEx` or `CreateWindow` function. (The message is sent before the function returns.) The window procedure of the new window receives this message after the window is created, but before the window becomes visible.

`WM_DESTROY`

This message is sent when a window is being destroyed. It is sent to the window procedure of the window being destroyed after the window is removed from the screen.

`MM_WOM_CLOSE`

The message is sent to a window when a waveform-audio output device is closed. The device handle is no longer valid after this message has been sent.

`MM_WOM_DONE`

This message is sent to a window when the given output buffer is being returned to the application. Buffers are returned to the application when they have been played, or as the result of a call to the `waveOutReset` function.

`MM_WOM_OPEN`

This message is sent to a window when the given waveform-audio output device is opened.

`MM_WIM_DATA`

This message is sent to a window when waveform-audio data is present in the input buffer and the buffer is being returned to the application. The message can be sent either when the buffer is full or after the `waveInReset` function is called.

`MM_WIM_OPEN`

This message is sent to a window when a waveform-audio input device is opened.

MM_WIM_CLOSE

This message is sent to a window when a waveform-audio input device is closed. The device handle is no longer valid after this message has been sent.

As it is known that a callback function uses a switch statement and in our case the above messages are handled by our code. The message handler's implementation is provided in the CWinAu class.

4.2.3 Multimedia and Windows Message Handlers

The message handlers of the above described messages are summarized in the fig 4.2.

Messages	**Message Handler Functions**

WM_CREATE	InitCreate(hWnd)
	InitAuIn()
	InitAuOut()

WM_DESTROY	CloseAll()
MM_WOM_CLOSE	MMWomClose()
MM_WOM_DONE	MMWomDone(wParam,lParam)
MM_WOM_OPEN	MMWomOpen(wParam,lParam)
MM_WIM_DATA	MMWimData(wParam,lParam)

MM_WIM_OPEN	MMWimOpen(wParam,lParam)
MM_WIM_CLOSE	MMWimClose()

Fig 4.2: Multimedia and Windows Messages and their Handlers

The description of some the important message handlers is as follow:

InitAuIn() : Sets format for input and opens the wave input device ie the microphone. 'pcm' is an object of a structure named WAVEFORMATEX defined in mmsystem.h. The code snippet shown below shows the setting up of wave format for input

```
pcm.wFormatTag      = WAVE_FORMAT_PCM;
pcm.nChannels       = 2;
pcm.wBitsPerSample  = 16;
pcm.nSamplesPerSec  = 44100;
pcm.nAvgBytesPerSec = pcm.nSamplesPerSec*(pcm.wBitsPerSample/8);
pcm.nBlockAlign     = (pcm.wBitsPerSample/8)*pcm.nChannels;
```

After this wave input device is opened by calling waveInOpen( ) function which takes handle to the input device and the object of WAVEFORMATEX.

InitAuOut(): Sets format for playback and opens the wave out device. The format for output must be the same as for the input. The function to open the wave out device is shown below

```
waveOutOpen(&hWaveOut,WAVE_MAPPER,(struct tWAVEFORMATEX

*)&pcm,(UINT)hParentWnd,0L,CALLBACK_WINDOW);
```

`MMWomDone()`: Prepares a waveform-audio data block and sends it to the given waveform-audio output device

```
waveOutPrepareHeader(hWaveOut,pw,sizeof(WAVEHDR));
waveOutWrite(hWaveOut,pw,sizeof(WAVEHDR));
```

`MMWimOpen()`: prepare two buffers for the sound input device

```
result=waveInPrepareHeader(hWaveIn, pWaveHdr1, sizeof(WAVEHDR));
result=waveInAddBuffer(hWaveIn, pWaveHdr1, sizeof(WAVEHDR));
// prepare second data buffer, and add into wavein device
pWaveHdr2->lpData    = (char *)pBuffer2;
pWaveHdr2->dwFlags   = 01;
result=waveInPrepareHeader(hWaveIn, pWaveHdr2, sizeof(WAVEHDR));
result=waveInAddBuffer(hWaveIn, pWaveHdr2, sizeof(WAVEHDR));
```

`MMWimData()`: Gets filled sound buffers from microphone and creates and writes to wave file.

Create wave file of specified file name shown in the following code snippet:

```
if(m_pWrite && btnTxt=="Stop Recording" && fileCreated==FALSE)
{
   PWRITESOUNDFILE pwsf= (PWRITESOUNDFILE) new  WRITESOUNDFILE;
   ZeroMemory(pwsf,sizeof(WRITESOUNDFILE));
   char *p = pwsf->lpszFileName;
   fname+=".wav";
   strcpy(p,fname);
   memcpy(&pwsf->waveFormatEx,&pcm,sizeof(pcm));
   m_pWrite->PostThreadMessage(
   WM_WRITESOUNDFILE_FILENAME,0,(LPARAM)pwsf);
   fileCreated = TRUE; }
   }
```

`MMWimClose()`: Frees all buffers

```
GlobalFreePtr(pBuffer1);
GlobalFreePtr(pBuffer2);
GlobalFreePtr(pBuffer3);
```

`MMWomOpen()`: Set and prepare wave output headers

```
pWaveOutHdr->lpData   = pOutBuffer;
```

4.3 Dialup Module

I used a class, `CTapiConnection`, from Microsoft Development Network that supports the basic functionality needed to use the Microsoft Windows® Telephony Application Programming Interface (TAPI) [23] to automatically dial the telephone for a voice connection. I have made a DLL of the CTapiConnection and used it in my project. The class provides the following functionality:

- Initialize TAPI
- Obtain a phone line
- Place a call
- End a call

4.3.1 CTapiConnection Class

The `CTapiConnection` class provides the application developer with a simple method of establishing a TAPI connection. It contains the following simple functions:

- Create
- DialCall
- HangupCall
- lineCallbackFunc

Note: The line callback function is not called directly by the application using the class; rather, it is called by the system for line notifications.

Initializing TAPI

The first thing an application must do before it uses any telephony services is to initialize TAPI. This means that the application must establish some way to communicate between itself and TAPI. TAPI uses a callback function to facilitate this communication. The application tells TAPI the address of this callback function when the application makes a call to `lineInitialize()`.

The `Create()` function creates the TAPI connection with the application. It call the `lineInitialize()` function which fills in two values passed to it: a usage handle (`m_hLineApp`) and the number of line devices available to the application.

Obtaining a Line

Now that TAPI has been initialized, the application needs to interact with the user to know what type of call to make. An Edit box lets the user enter the desired phone number to dial and then click the Dial button to dial.

The application needs to obtain a line handle. This is done by a call to the `lineOpen()` function. Before the application can make a call to `lineOpen()` it has to make sure that the line can support the type of call that the application is trying to make.

Placing the Call

An application uses the `lineMakeCall()` function to place a call. This function takes the following parameters:

- A handle to the open line obtained from the `lineOpen()` call.
- A pointer to the handle for the call. This will be filled in by `lineMakeCall()`.
- The destination address (the area code and telephone number).
- The country code (which can be set to zero to use the default value).
- A pointer to line parameters. This allows the application to specify how the call should be set up (that is, the data rate, dialing parameters, and so on). If this is set to NULL, the default setup is used.

The `lineMakeCall()` function returns a positive "request ID" (saved in the `m_dwRequestID` member variable of my class) if the function will be completed asynchronously, or a negative error number if an error has occurred.

After the `lineMakeCall()` function successfully sets up the call, the application receives a `LINE_REPLY` message (the asynchronous reply to `lineMakeCall`). The application gets this message through its callback function. This means that a call at the local end has been established (that is, we have a dial tone). The `LINE_REPLY` message also informs the application that the call handle returned by `lineMakeCall()` is valid.

Several other messages or notifications are sent to the application's callback function. For instance, as the call is placed, the call passes through a number of states, each of which results in a `LINE_CALLSTATE` message sent to the callback function. After the message indicating the connected state is received, the application can begin sending data. A handler is provided for `LINE_CALLSTATE` message that prints debug messages indicating the current call status.

Sending Data

The user would specify, through some user interface widget, what file or data to send and then initiate the data transmission. The TAPI functions continue to manage the opened line and the call in progress, but the actual transmission is started and controlled by non-TAPI functions (that is, COMM functions). This is the type of transmission that the TAPICOMM sample demonstrates.

TAPI continues to monitor the line and call, but if there is anything special we need to do, it is up to us. I used a function that allows us to resynchronize a TAPI line call by waiting for the `LINE_REPLY` message. In other words, it waits until a `LINE_REPLY` is received or the line is shut down. The function is called from the same thread that made the call to `lineInitialize()`. If the call was dropped while in a wait state, this function can potentially be re-entered.

Ending the Call

When the user finishes the phone call, the application receives a `LINE_CALLSTATE` message telling it that the state of a line device has changed. At this point the application can disconnect the call. The application disconnects the call when the user clicks the Disconnect button.

Before the application disconnects the call, it checks to see if a call is already in progress. If not, the application calls the `lineDrop()` function to place the call in the IDLE state. The call handle must then be released for the finished call. This is done by the `lineDeallocateCall()` function. Finally, `lineClose()` is called to close the line. At this point, there will be no more incoming or outgoing calls using that line handle.

At this point the application is finished using TAPI and can continue to do whatever else it was designed to do.

4.4 Recording and Playing Module

The following classes, `CWinAu` and `CPlaySound`, record sound and play PCM sound. There are two more classes, `CWriteSoundFile` and `CPlayMMSound`. `CWriteSoundFile` receives sound buffers from `CWinAu` and writes them to a WAV disk file. `CPlayMMSound` opens these WAV files and plays them to the sound device. All sound files are currently PCM samples.

With these classes, sound can be both recorded and played. Continuous sound can be monitored and upon appropriate queues, sound can be played back. The multithreading allows other actions to take place while sound recording i.e., sound can be saved to WAV files.

`CPlaySound` is being invoked by the following code:

```
m_pPlaySound = new CPlaySound();

m_pPlaySound->CreateThread();
```

Similar calls begin playing of sound:

```
m_PlayThread->PostThreadMessage(WM_PLAYSOUND_STARTPLAYING,0,0);
```

`CWriteSoundFile` has the following messages associated with it:

For creating file of the specified name, the user is provided with an input dialog. The user can enter the file name and the that file name is then set into `PWRITESOUNDFILE` structure and then this structure is type casted into `LPARAM` as shown:

```
m_WriteSoundThread->
PostThreadMessage(WM_WRITESOUNDFILE_FILENAME,0,(LPARAM)
PWRITESOUNDFILE &structWriteSoundFile);
```

For writing blocks to the wave file the following message is used:

```
m_WriteSoundThread->
PostThreadMessage(WM_WRITESOUNDFILE_WRITEBLOCK,0,
(LPARAM)(WAVEHDR)pWaveHdr);
```

For closing sound file the following message is used:

```
m_WriteSoundThread->
PostThreadMessage(WM_WRITESOUNDFILE_CLOSEFILE,0,0);
```

To write a WAV file you must provide a WRITESOUNDFILE structure. The `WRITESOUNDFILE` structure has the following definition:

```
typedef struct writesoundfile_tag {
  char lpszFileName[MAX_PATH];
  WAVEFORMATEX waveFormatEx;
  TCHAR buffer[100];
} WRITESOUNDFILE, *PWRITESOUNDFILE;
```

You must provide the filename, then the WAVEFORMATEX structure that defines the file to be written. With non PCM formats there are extra style specific information at the end of the structure, hence the 100 bytes of buffer space.

This class receives WAVEHDR blocks created by CWinAu. Pushing the "Start Recording" button will save it to a filename entered by the user.

CPlayMMSound will read a WAV file and play it to the sound device. It uses a pointer to a CPlaySound thread to acheive this. Its messages are:

The user is again provided with the facility to play the sound file of his/her choice. For that purpose the filter of .wav extension is used. Then the file name returned by the file open dialog is sent to the following message.

```
m_pPlayMMSound->
PostThreadMessage(WM_PLAYMMSOUND_PLAYFILE,0,(LPARAM)fileName)
```

For stop playing the sound the following message is invoked.

```
m_pPlayMMSound->PostThreadMessage(WM_PLAYMMSOUND_CLOSEFILE,0,0);
```

To get the pointer of the CPlaySound class the following message is used.

```
m_pPlayMMSound->
PostThreadMessage(WM_PLAYMMSOUND_PLAYSOUNDPTR,0,(LPARAM)
(CPlaySound*)m_pPlaySound);
```

The WM_PLAYMMSOUND_PLAYFILE messages opens a WAV file for processing. It automatically sends off a worker thread to play the file.

`CPlaySound` thread must be provided for this to work. That is the job of the `WM_PLAYMMSOUND_PLAYSOUNDPTR` message. You can stop the play at any time by the `WM_PLAYMMSOUND_CLOSEFILE` message.

Multimedia file I/O functions [18] provided in the MMSYSTEM.H have been used extensively to complete the Recording and the playing module.

4.5 Masking Module

Initially different digital audio processing techniques like FFT, STFT and wavelet transforms were studies for masking of voice. These algorithms or techniques tend to be better suited to analyzing digital audio recordings than for filtering or synthesizing sounds. In addition to inefficiency of these algorithms the technique is more inclined towards DSP and requires an in depth knowledge of DSP [24]. In order to get the work done, different sound processing libraries like OpenAL [10], SndObj [25], Audiere [26] etc were studies and the one that best meets our requirements was chosen. The one which was chosen was SoundTouch [9] sound processing library which seemed to be more suited and works quite efficiently.

SoundTouch is an open-source audio processing library that allows changing the sound tempo, pitch and playback rate parameters independently from each other, i.e.:

- Sound tempo can be increased or decreased while maintaining the original pitch
- Sound pitch can be increased or decreased while maintaining the original tempo
- Change playback rate that affects both tempo and pitch at the same time
- Choose any combination of tempo/pitch/rate

Some important SoundTouch Library procedures used in my project are as follows:

- `void setTempo(float newTempo);`

Sets new tempo control value. Normal tempo = 1.0, smaller values represent slower tempo, larger faster tempo.

- `void setRateChange(float newRate);`

Sets new rate control value as a difference in percents compared to the original rate (-50 .. +100 %)

- `void setTempoChange(float newTempo);`

Sets new tempo control value as a difference in percents compared to the original tempo (-50 .. +100 %)

- `void setPitch(float newPitch);`

Sets new pitch control value. Original pitch = 1.0, smaller values represent lower pitches, larger values higher pitch.

- `void setPitchOctaves(float newPitch);`

Sets pitch change in octaves compared to the original pitch (-1.00 .. +1.00)

- `void setPitchSemiTones(int newPitch);`

Sets pitch change in semi-tones compared to the original pitch (-12 .. +12)

- `void setChannels(uint numChannels);`

Sets the number of channels, 1 = mono, 2 = stereo

- `void setSampleRate(uint srate);`

Sets sample rate.

Apart from these programming interfaces, a helper function called PShiftAudioSignal( ) has been provided which does special processing on the input sound buffers for changing the voice. The function takes three arguments an input parameters as shown below:

```
PShiftAudioSignal(short *bufferToProcessPtr, int ItsLength, int
SemiToneVal);
```

The first parameter is the input sound buffer, the second is the length of the input buffer and the third is the SemiToneValue which is passed to the function from the slider control present on the Dialog for changing the pitch of the sound. The function returns the masked (changed) sound buffers to be saved in the WAV files of to be played on the out put devices.

The three modules were integrated keeping in view the various factors like execution efficiency, memory management and utilization and code compactness in mind. Object oriented approach has been followed while coding the software except for one file i.e VoiceMask2.cpp. There are places where certain C library routines have also been used but wherever they are used it is kept in mind that it does not effect the overall design and the efficiency of the software.

CONCLUSIONS AND FUTURE WORK

This chapter concludes the project document giving a brief summary followed by conclusion and the future work.

5.1 Summary

The project was aimed at presenting the basic concepts of digital audio processing and to make a Voice Masking System, which was one of the main objectives of this project. With few modifications the software could be used as a full-fledged voice masking system or even as an Enterprise Telecommunication Security System.

Chapter 1 presents the over all idea of the software along with the goals and uses. During the initial stages of the project lot of time was spent on studying and analyzing algorithms for digital audio processing. The basic idea behind using FFT was to get a time domain representation of digital sound to its frequency domain equivalent and thereby to play with different parameters that are only available in frequency domain like phase, amplitude and frequency. The algorithm was more suited for analysis and drawing the sound waves. That is why, after the discussing the issue with the team, it was decided to make use of the off the shelf sound processing libraries freely available on the Internet. This is discussed in Chapter 2 of the document. Chapter 3 describes the detailed design of the software. It also presents the in depth details about the Use case and the Class diagrams. Chapter 4 presents the implementation details followed by Chapter 5 with presents the summary and the future prospects.

This our summarized point of view but we feel that there are still lots of Voice Masking techniques which still have to be explored and lot of applications and uses of Voice Masking which are still left unexploited.

5.2 Conclusions

Voice processing is a field in which a lot of research and development is being carried out in the present time. The opportunity to modify voices in real-time, as well as pre-recorded speech provides a range of substantial benefits, allowing people to stay anonymous during communication, fit the person type they prefer as well as to understand each other in a more efficient way. Voice masking could also be used in making animated movies and in game programming.

The project and its related work can be used to get an overview of voice and its different characteristics, how to modify voice and the effects of changing the frequency / pitch on voice data. It can also be used to get a better understanding of the future trends in the applications that use voice like VoIP, voice security systems etc. We have come up with software which can be used in place of the hardware device known as the Voice Changer II.

Finally this project will help the software developers and the IT students, who are interested in developing Voice applications, to get a better understanding of this domain.

5.3 Future Work

There are many promising opportunities for future work. Firstly, we plan to enhance our application by adding voice recognition features. Voice Recognition is itself a very vast area where lot of research can be done. Secondly, future work also includes comparison analysis amongst the three DSP techniques discussed earlier in chapter 2, for analyzing digital audio. We also plan to develop a tool for audio content analysis amongst the three algorithms. Besides this an IP based software tool can also be made through which people can chat on the Internet as well as the local area networks.

[1] "Voice Changer II" A hardware device for Voice Masking - http://www.smarthome.com/5128.htmlhttp://www.smarthome.com/5128.html

[2] "An Introduction to Fourier Theory" http://aurora.phys.utk.edu/~forrest/papers/fourier/index.html

[3] "Discrete Fast Fourier Transform" – A tutorial by Don Cross http://groovit.disjunkt.com/analog/time-domain/fft.html

[4] "WAVE File Format" - http://www.borg.com/~jglatt/tech/wave.htm

[5] "Digital Audio" – An overview by Dave Hillman http://dhillman.com/theplace/webintro/graphics/webaudio/

[6] "Sound Formats"- http://dio.cdlr.strath.ac.uk/standards/fileformats/soundformats.html

[7] "Mastering The Fourier Transform in One Day" by Stephan M. Bernsee http://www.dspdimension.com/html/dftapied.html

[8] "Free Audio, Sound, Music and Digitized Voice Libraries and Source Code" http://www.thefreecountry.com/sourcecode/audio.shtml

[9] "SoundTouch Audio Processing Library" http://sky.prohosting.com/oparviai/soundtouch

[10] "OpenAL, the Open Audio Library" http://www.gel.ulaval.ca/~dumais01/genu/tutorials/AdvanceTutorial2.html

[11] "Speech Recognition 2003 Project Page ChoirFish Tune Identification" http://www.liacs.nl/~sgroot/speech/

[12] "ASR-P" - A Speech Recognition Project by Derk Geene & Sander van der Maar http://www.liacs.nl/~dgeene/speech/experimentation.html

[13] "The Fastest Fourier Transform in the West - 27/11/2003" http://www.®tw.org

[14] "Characteristics of sound" http://www.geocities.com/musicallahari/soundchar.htm

[15] "Recording and Playing sound" - http://www.developer.com/tech/print.php/627481

[16] Telephony API - MSDN July2002 Edition

[17] Visual C++ Books
Using Visual C++ 6 *by Kate Gregory*
Sams Teach Yourself Visual C++ 6 in 21 Days
Practical Visual C++ 6 *by Bates Tompkins*
Visual C++ Programming *by Yashvant Kanetkar*

[18] Multimedia File IO Functions -
ms-help://MS.MSDNQTR.2002JUL.1033/multimed/mmfunc_27ar.htm

[19] Pulse Code Modulation – Radio TV and Audio Technical Reference Book *by S. W. Amos*, pg. 17-30

[20] STFT - http://cnmat.cnmat.berkeley.edu/~alan/MS-html/MSv2.html

[21] Digital Audio Libraries - http://www.thefreecountry.com/sourcecode/gui.shtml

[22] RIFF - ms-help://MS.MSDNQTR.2002JUL.1033/multimed/mmio_2uyb.htm

[23] TAPI Overview - ms-help://MS.MSDNQTR.2002JUL.1033/tapi/tapisdk_87nb.htm

[24] DSP- http://www.dspdimension.com/html/home.html

[25] The SndObj Sound Object Library
http://www.may.ie/academic/music/musictec/SndObj/main.html

[26] Audiere Sound Library - http://images.sourceforge.net/prdownloads/sf-stipple.png

[27] Audio Engineering Society
http://www.aes.org/sections/chicago/may01reviewa.html

[28] IEEE Signal Processing Society - http://www.dsp2002.gatech.edu

[29] Corey Cheng. Wavelet Signal Processing of Digital Audio with Applications in Electro-Acoustic Music. Master Thesis, Hanover, New Hampshire. http://www.eecs.umich.edu/~coreyc/thesis/thesis_html, 1996.

User Manual

Hardware Requirements

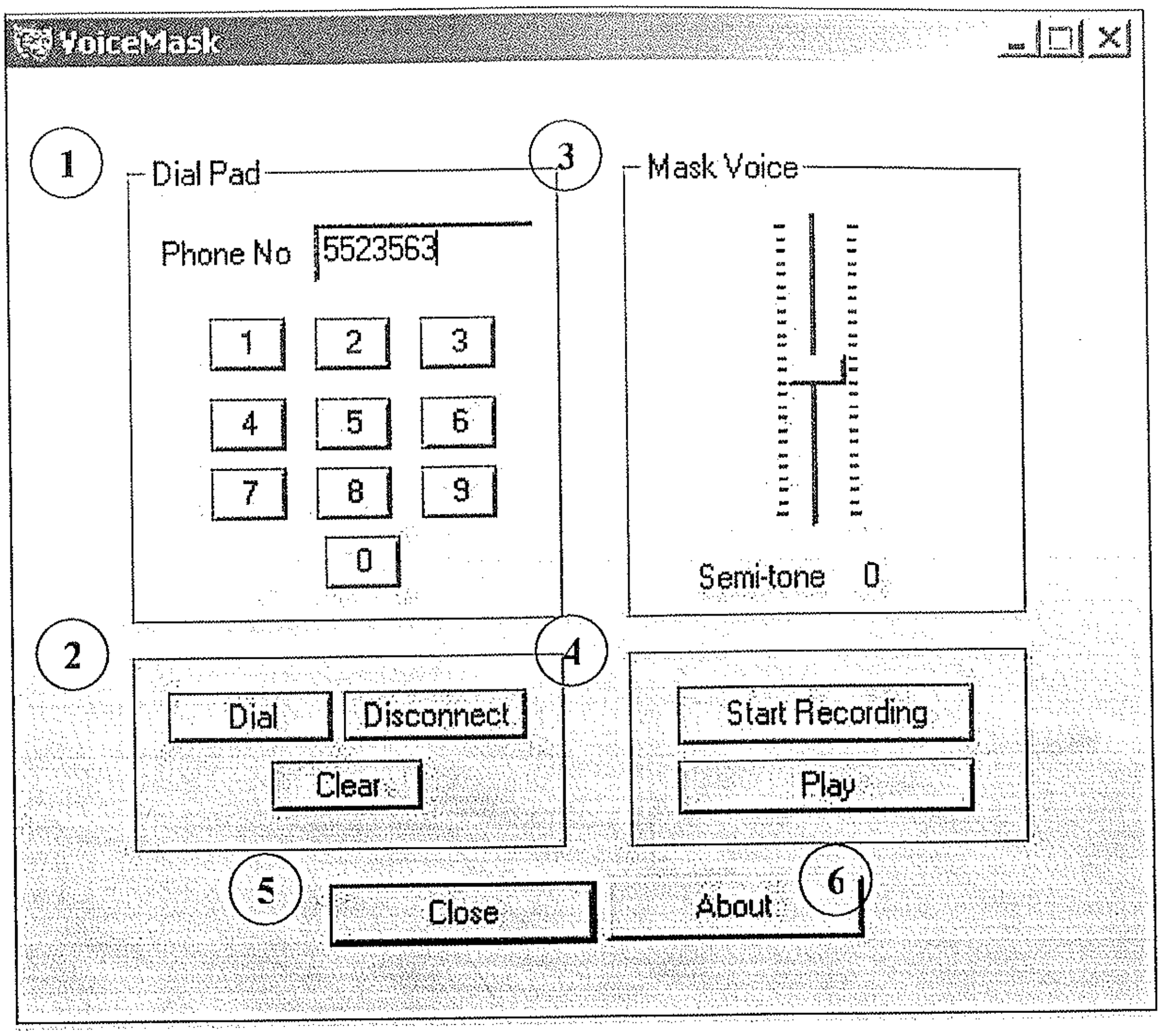
- Voice Modem
- Telephone Line
- Microphone
- Speakers
- Audio/TAD cable

Other Requirements

- Windows 98, ME, 2K

Voice Masking System Screen Shot

VMS is a dialog based application as shown in the figure below. The software has three modules which can be seen in the dialog. Brief description of the module and how the software can be used is given according to the numbers encircled.



1. *Dial Pad* can be used to enter a telephone number. An edit box is provided for this purpose.
2. After entering the number in the edit box, the user can press the *dial* button to make call. Likewise *disconnect* button is used to disconnect the call. *Clear* button can be used to clear the number from the phone no. edit box.
3. A slider is given on the dialog for the users to change or mask the voice. Level 0 (zero) represents the original voice i.e. no change in voice. As you go down towards the

negative values of the *Semi-tones* the voice gets grave. Positive values of the slider changes the voice to shrill voice.

4. The *Start Recording* button starts writing a wave (.wav) file on your hard disk by the name of "sound.wav". You can save your masked voice in this file and can play it by pressing *Play* button.
5. *Close* button closes the application.
6. By pressing *About* button you can know about the version and little description of the software.