1999

Telephony-based answering machine application

Army Fithry

Telephony-based Answering Machine Application

By

Army Fithry

Project submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Technology

Department of Information Technology
Rochester Institute of Technology

August 1999

Rochester Institute of Technology
Department of Information Technology

Master of Science in Information Technology
Project Approval Form

Student Name: Army Fithry

Student Number: _____

Thesis Title: Telephony-based Answering Machine Application

Project Committee

Name	Signature	Date
Prof. Rayno Niemi Chair		8/25/99
Mr. Robert Hermanns Committee Member		8/28/99
Prof. Alec Berenbaum Committee Member		8/25/99

TABLE OF CONTENTS

ABSTRACT 1

1. INTRODUCTION 2
1.1. Background 2
1.2 Purpose 3

2. PROJECT OVERVIEW 4
2.1 Project Requirement 4
2.1.1. Hardware 4
2.1.2. Software 4

3. FUNCTIONAL DESCRIPTION 5
3.1. Configuring the Application 5
3.2. Establishing an Outbound Call 5
3.3. Answering an Inbound Call 6
3.4. Answering Machine is Enabled 7
3.5. Creating Voice Mailbox 7
3.6. Managing Messages 8

4. WINDOWS TAPI CONCEPT 10
4.1. Windows Application Programming Interface (API) 10
4.1.1. Windows API and Visual Basic 10
4.1.2. Parameters List 11
4.1.3. Call Results 11
4.2. Windows TAPI 11
4.2.1. Synchronous / Asynchronous Operation 13
4.2.2. Synchronous Functions 13
4.2.3. Asynchronous Functions 13
4.2.4. Callback Function 13
4.2.5. TAPI-Compliant Telephony Device 13
4.3. Microsoft UniModemV 14
4.3.1. Modem.inf Files Play a Key Role 14
4.4. Media Access and Multimedia API 14
4.4.1. Waveform Audio 15

5. APPLICATION FLOW 16
5.1. Initialization 16
5.1.1. ID File 16
5.1.2. Wave device's format 16
5.1.3. TAPI Initialization 16
5.1.4. API Negotiation 17
5.1.5. Line and Phone Capabilities 18

5.1.6. Opening Line and Phone Device 19
5.1.7. Setting Status Messages 20
5.2. Establishing a Call 20
5.2.1. Entering the Phone Number 20
5.2.2. Filtering Keyboard Hit 21
5.2.3. Making a Call 21
5.2.4. Listening to the Conversation 22
5.3. Receiving a Call 22
5.3.1. Incoming Call Detection 22
5.3.2. Answering a Call 24
5.3.3. Answering an Incoming Call Manually 24
5.3.4. Answering an Incoming Call Automatically 25
5.4. Managing Waveform Audio Data over the Telephone Connection 25
5.4.1. Playing Waveform Audio Data 26
5.4.2. Recording Waveform Audio Data 26
5.5. Callback Function 26
5.6. Detecting Digits 27
5.6.1. Digit Options 27
5.6.2. Digit States 27
5.7. Managing Voice Mailboxes 28
5.8. Ending a Call 29
5.9. Closing TAPI 30
5.9.1. Closing the Line/Phone 30
5.9.2. Shutdowning the TAPI 30

6. CONCLUSIONS 31

7. FUTURE ENHANCEMENTS 32

APPENDIX 33

REFERENCES 35

ABSTRACT

Advances in hardware and software technology make computer and communications integration becomes a reality. The popularity of multimedia board for PC together with modem that can handle voice makes operating system's vendor, like Microsoft, add some kind of abilities to their product to explore them.

The Telephony API, one of the most significant API sets to be released by Microsoft, is a single set of function calls that allows programmers to manage and manipulate any type of communications link between the PC and the telephone line(s). With visual programming tools, such as Visual Basic, TAPI become a powerful tool to create a telephony application that works like or even more sophisticated than a real telephony device.

The Answering Machine Application is application that will be able to handle outbound call and inbound call. The outbound calls can be made by entering and dialing telephone number using GUI interface provided by the application. However, the biggest part of application is to handle inbound calls. The inbound calls could be answered by person or by application. It gives caller the possibility to listen to the announcement or to leave a message. The application also provide facility to create and maintain multiple voice mailboxes and each of which can contain multiple messages.

CHAPTER 1

INTRODUCTION

1.1. Background

The term "telephony" refers to any of a number of technologies that are used to transmit voice from one person to another across a distance. Since the telephone was invented, many technological improvements have been made to bring easy of use to both the user and provider. The use of telephone is now shifting from human to human interaction into human to machine interaction. Computer technology with its sophisticated peripherals make this possible.

The use of Personal Computer (PC) in telephony application was not popular for the first decade after PC was invented. The biggest problem was lack of hardware and operating system's support. The invention of the sound card, improvements in modem technology, and improvement on PC operating system alleviated much of this problem.

Before Windows 95 operating system and Telephony Application Program Interface (TAPI) was launched, any application that wanted to dial a telephone had to access the telephone line directly via a COM port and the COMM (Communication) API. This required complicated setup on the user's part and complicated code on the developer's part. Now, TAPI eliminates that hard side of telephony application development.

Windows 95 operating system provides the most powerful and flexible platforms for the development and use of Computer Telephony Integration (CTI) applications. TAPI is supported by a full range of complementary Application Programming Interface (APIs) to enable a broad range of powerful, easy-to-use telephony and communications applications for a wide range of customers. Microsoft ActiveX Controls provide developers with powerful and easy-to-implement "plug-in" software components that simplify the development process, allowing developers to focus on creating their own value-added applications.

For developers and vendors of CTI software and hardware, the combination of Windows and TAPI creates a new world of opportunities. As the computer and telephone are merged to create a new generation of communicating PCs, there should be an explosive demand for telephony applications, and Windows 95 operating systems represent huge markets for such products.

TAPI's hardware abstraction frees developers from having to create separate applications for each switch, just as TAPI's UniModem support protects developers from the exhaustive chore of having to directly support the vast array of modems.

CTI is beginning to revolutionize the manner in which people interact with telecommunications. This compelling combination of openness, comprehensiveness,

scalability, and integration makes TAPI and Windows-based telephony the platform of choice for what should be an exciting and booming market.

1.2. Purpose

The purpose of this project is to develop a telephony application by using Windows 95, TAPI as the primary tool, and Visual Basic as the application development tool.

Since Windows 95, a Graphical User Interface (GUI) based Operating System, was launched by Microsoft, many GUI based application development tools have been developed. One of those tools is Visual Basic, which was launched by Microsoft as well. The evolvement of Visual Basic is based on the improvement of Windows 95. One of many features that make Visual Basic able to compete with other languages is the ability to access hardware configuration directly by calling the appropriate function within Windows API.

The Answering Machine Application is application that will be able to handle outbound and inbound calls. The outbound calls can be made by entering and dialing telephone number using GUI interface provided by the application. However, the biggest part of application is to handle inbound calls. The inbound calls can be answered by person or by the application. The caller listens to an announcement and may leave a message. The application provides the facility to create and maintain multiple voice mailboxes and each of which can contain multiple messages.

CHAPTER 2

PROJECT OVERVIEW

2.1. Project Requirement

Hardware and software that will be required to implement the application are described below :

2.1.1. Hardware :

a. IBM PC compatible (minimum 486 processor)
b. 16 MB RAM or more
c. Super VGA monitor
d. 100 MB free hard disk space
e. Sound card
f. Microphone and speaker
g. TAPI-compliant telephony device
h. Telephone line

A telephony device is a card that plugs into a computer (or its serial port) and interface to a Plain Old Telephone Service (POTS) line. A telephony device can answer the phone, place calls, transmit and recognize tones, and more. Some telephony devices are fax machines on a card, and some also work as data modems. A voice modem is a combination telephone and data modem.

TAPI-Compliant means that a device's manufacturer distributes a driver, called a Telephony Service Provider (TSP), that makes it compatible with Microsoft's Telephony Application Program Interface (TAPI), which serves as a standard interface to TAPI-compliant hardware, but is woefully complex itself. Not every telephony device is TAPI-compliant in all aspects. A device can be TAPI-compliant in some aspects but non-compliant in others.

2.1.2. Software :

a. Operating system : MS Windows 95
b. Application development tool : Visual Basic 6.0
c. Telephony dynamic link library : TAPI32.dll
d. Telephony Service Provider : UnimodemV
e. Multimedia dynamic link library : Winmm.DLL

CHAPTER 3

FUNCTIONAL DESCRIPTION

The application has following capabilities :

a. Establishing and disconnecting an outbound call
b. Answering an inbound call
c. Creating up to 5 voice mailboxes
d. Creating up to 100 voice mails for each voice mailbox
e. Managing the voice mailbox by calling the system
f. Managing the voice mailbox by opening the menu in computer

3.1. Configuring the Application

Before making an outbound call or receiving an incoming call the user does not need to configure anything related to line or phone device selection because the application will automatically detect the available line or phone device and select the one that has required capability.

3.2. Establishing an Outbound Call

Steps that must be done by the user to establish an outbond call are :

a. Entering phone number
There are 2 ways to enter phone number.The first one is by entering the number directly into display panel on the top of keypad. The second one is by clicking the number on keypad. (Figure 3.1. Main Windows). The display panel will display numbers only. Any alphabets entered by the user will be converted to numbers. (For detail convertion of the alphabets see chapter 4.2.1.)

b. Press "Dial" button
Once the phone number is completed, the application is ready to make a call. To do this job the user must hit Dial button

c. Press "Enable Speaker Phone" button
To have a conversation with another party when making an outbound call, the user must activate the microphone and speaker as a telephone set. To do that, the just needs to hit the "Enable Speaker Phone" button

d. Press "Hangup" button
When the conversation is done, the connection must be disconnected by the user by hitting "Hangup" button

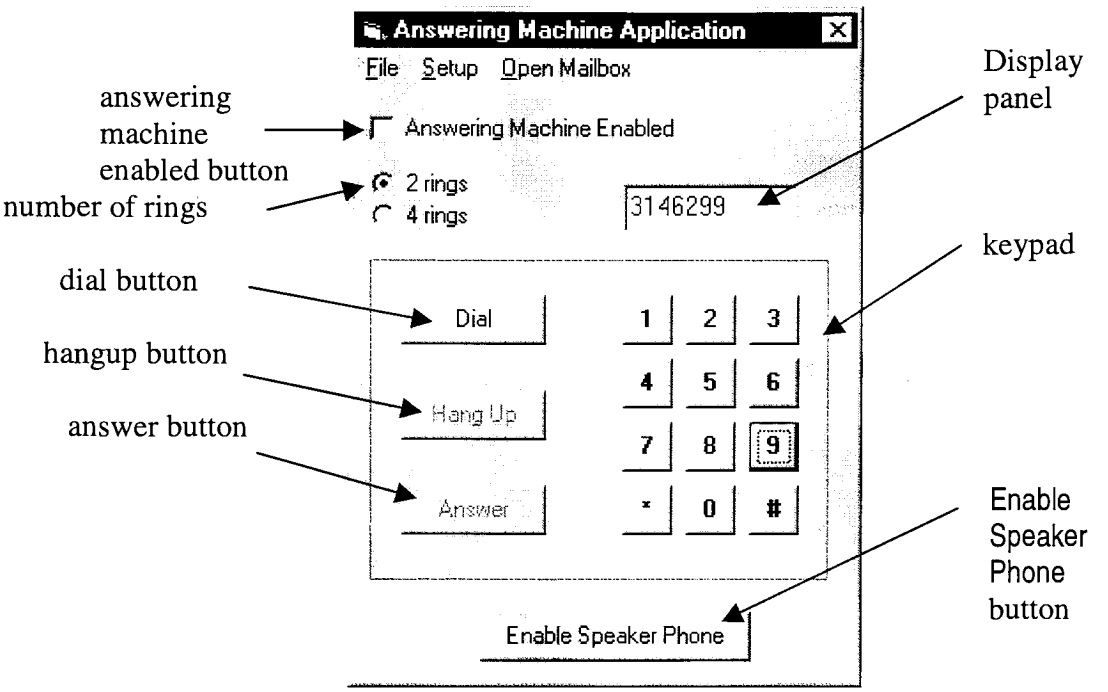

Figure 3.1. Main windows

3.3. Answering an Inbound Call

There are 2 ways to answer an inbound call :

a. Manually

In order to answer an inbound call manually the user must uncheck the "Answering Machine Enabled" option. Then, when the call is detected, the user must hit "Answer" button. To have a conversation with another party when receiving an inbound call, the user must activate the microphone and speaker as a telephone set. To do that, the user needs to hit the "Enable Speaker Phone" button

b. Automatically

Steps that must be done by the user to let the application answer the inbound call automatically are :

- Check the "Answering Machine Enabled" option.
- Select the number of rings selection.
  The application will wait until that number of rings are completed before answering the call.

3.4. Answering Machine is enabled

When the "Answering Machine Enabled" option is checked, the application acts as an answering machine. Any incoming calls will be answered automatically after a specific number of rings. The application answers the calls by playing an announcement. The announcement gives caller several options which can be selected by pressing an appropriate button on the telephone set.

The options provided for the caller are
- Press 1 to leave a message into voice mailbox 1
- Press 2 to leave a message into voice mailbox 2
- Press 3 to leave a message into voice mailbox 3
- Press 4 to leave a message into voice mailbox 4
- Press 5 to leave a message into voice mailbox 5

The number of voice mailboxes that can be used depends on how many voice mail box that have been created.

3.5. Creating Voice Mailbox

The application provides capabilities to manage up to 5 voice mailboxes. Before using more that 1 voice mailboxes, the user must create them first. Steps that must be done by user to create voice mailbox are :

- Click Setup Option (Figure 3.2. Setup and Open Mailbox Option)
  The application will show the Setup windows
- Enter the voice mailbox password (Figure 3.3. Voice Mailbox Setup Windows)
  To change the number of voice mailbox user must enter the voice mailbox password first. This prevents an unauthorized person to use this feature
- Select the needed number of voice mailbox
  The user can create up to 5 voice mailboxes each of which can contain up to 100 messages

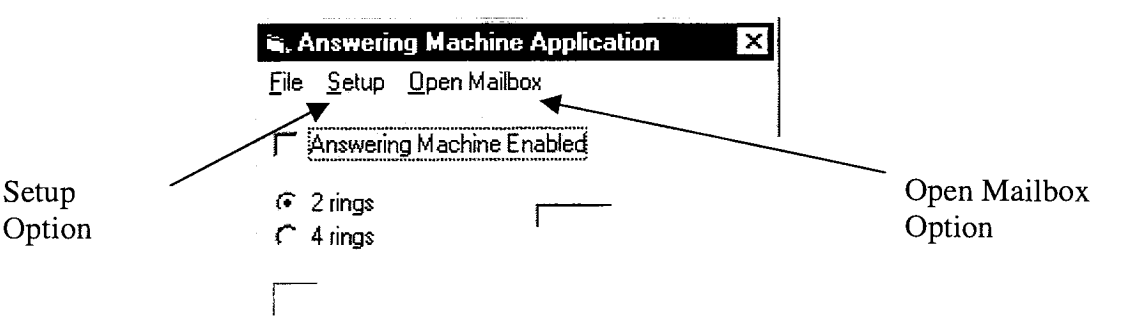

Figure 3.2. Setup and Open Mailbox Option

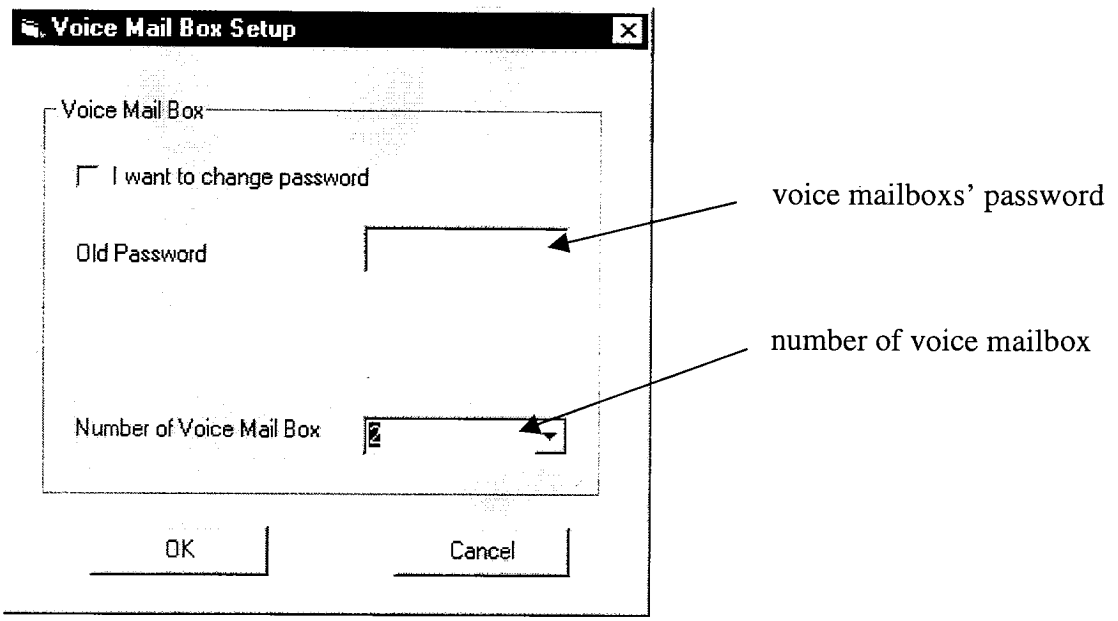

Figure 3.3. Voice Mailbox Setup Windows

3.6. Managing Messages

After messages are created in the mailbox(es), the user can open, play and delete them. These features of messages management can be done by opening Voice Mailbox windows.

Steps that must be done for these messages operation are :

- Select the voice mailbox number
- Select the message of that voice mailbox number that will be played
- Play the message by clicking "Play Message" button.
- If the message is not needed anymore, user can delete it by clicking "Delete Message" button

When the user dials into the application from a telephone, the only messages management that can be done is messages playing. The option for this operation can be done by pressing button 8 of telephone keypad after the announcement is played.

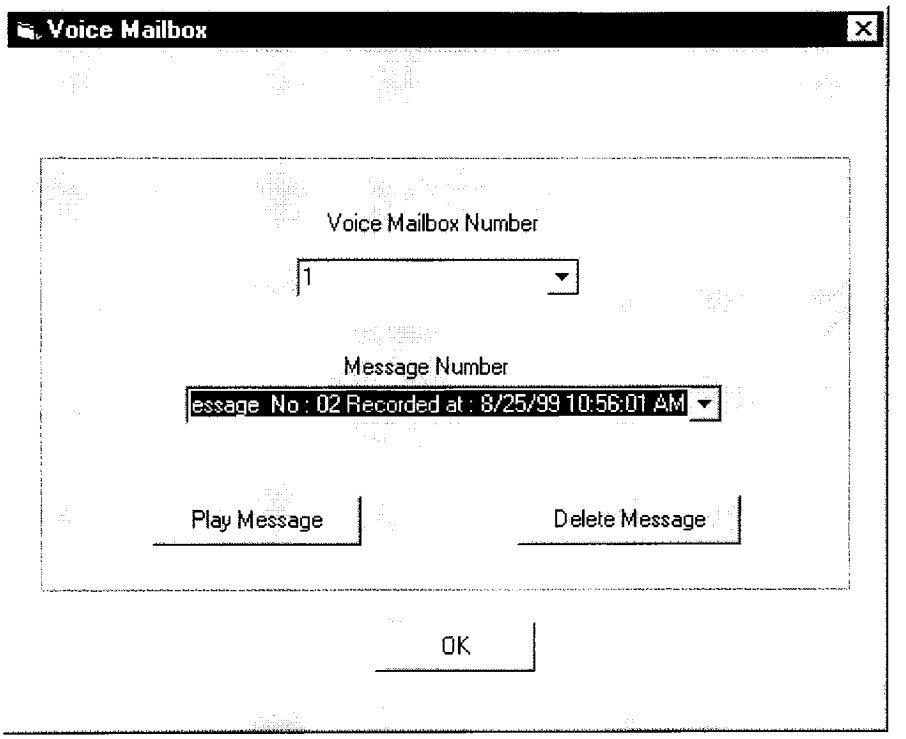

Figure 3.4. Voice Mailbox Windows

CHAPTER 4

WINDOWS TAPI CONCEPT

This chapter will provide brief discussion about important things that could help in developing telephony application using Windows TAPI.

4.1. Windows Application Programming Interface (API)

API is an acronym for Application Programming Interface. Simply put, an API is the set of functions which an operating system or a hardware device provides to allow software to use it. Without it, the software would have to provide its own drivers for all possible hardware configurations, which is extremely wasteful and annoying. An API provides a relatively easy way for programmers to use the full functionality of the hardware or the operating system without conceming themselves as much with the internal operations. It should be noted that the term API is often used colloquially to refer to API functions.

The Windows API is the API provided by the Windows operating system. It is a collection of literally hundreds of functions that allow all Windows-based software to do the things that normally associated with Windows. For example, some of the API functions provided in Windows provide ways to use files, create and use windows, access the system registry, play sounds and videos, use printers and other output devices, etc. Programmers don't need to manually create their own windows from scratch; rather they use the Windows API to tell Windows to create a window for them.

Almost all of the functions that make up the Windows API are contained in DLL (Dynamic Link Library) files. These DLLs are inside Windows's System directory. The functions are grouped together based on general function, but most functions are in the kernel32.dll, user32.dll, and gdi32.dll files.

Since dynamic link libraries are typically written in C, programmers who use other than C or C++ must find out how to use those API functions. Visual Basic gives a way out of the problem by adding a reference explicitly linking the program to an API function. Statement for that reference is a Declare statement.

4.1.1. Windows API and Visual Basic

The Visual Basic Declare statement is used to import a DLL function into Visual Basic. It informs Visual Basic where a DLL function may be found, and serves to let Visual Basic know what types of parameters a DLL function expects and what type of value it returns.

Once a DLL function is properly declared, it appears to the Visual Basic programmer similar to any other Visual Basic function or subroutine.

The most important consideration of declaring API or DLL functions is that they be properly declared. The function declaration in Visual Basic must correspond exactly to the DLL function in terms of numbers and types of parameters and the type of value returned. Any errors in this declaration are likely to lead to a fatal exception.

4.1.2. Parameter List

A parameter list is a list of dummy parameter names indicating the parameters that are passed to the function. In order to understand how to create parameter lists for dynamic link libraries, it is necessary to first examine the types of function parameters that DLLs may use. Since dynamic link libraries are typically written in C, they can use a wide variety of parameters that are not supported directly by Visual Basic. The choice of appropriate Visual Basic variable types is not always obvious and an incorrect choice can lead a fatal exception or runtime error.

4.1.3. Call Results

Almost every Windows API function returns a result. In some cases the result provides the information that was requested. In other cases the result indicates whether or not the function succeeded, where for some API functions a nonzero result indicates success and a zero result indicates failure and vice versa for the other API functions. This means that the result must be compared to zero explicitly when testing the result of API functions.

4.2. Microsoft Windows TAPI

The Telephony Application Programming Interface (TAPI) is one of the most significant API sets to be released by Microsoft. The telephony API is a single set of function calls that allows programmers to manage and manipulate any type of communications link between the PC and the telephone line(s).

In many ways, TAPI parallels other Win32 models. As an abstraction layer, its function is to provide a consistent, device-independent programming model to applications that use it. In Win32, an application does not need to know whether it's drawing a line on a 16-color VGA display or an accelerated super VGA display with millions of colors. All it needs to know is where it's drawing the line. TAPI aims to do for telephony what Graphic Device Interface (GDI) does for graphics.

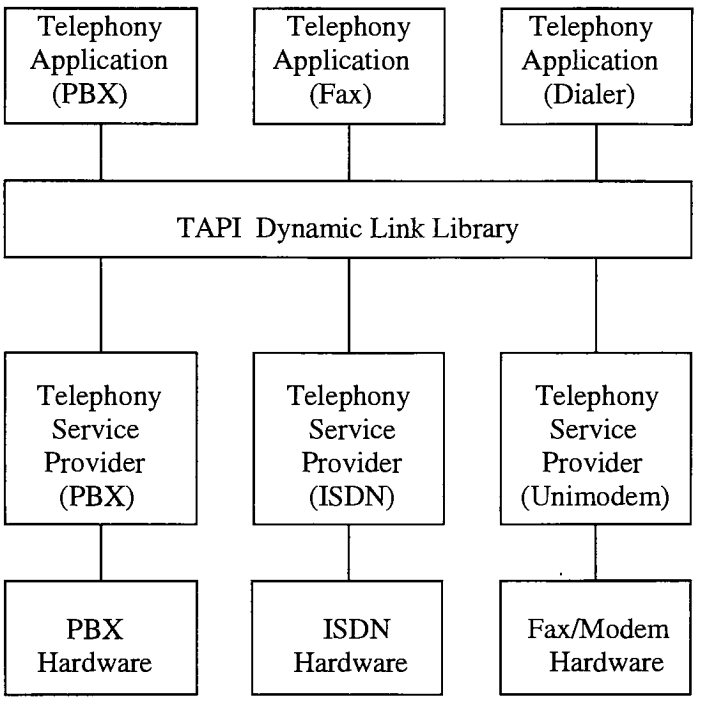

Figure 4.1. TAPI architecture

Based on the principles of the Windows Open Services Architecture (WOSA), TAPI provides some central services and holds some global state information. Its main purpose, however, is to provide connections between the Telephony Service Provider (TSP) and TAPI applications. Applications are programmed using TAPI, while TSPs implement the Telephony Service Provider Interface (TSPI) functions that are used by the TAPI implementations. Each TSP then uses whatever interface is appropriate to control its telephony hardware.

This layered approach makes it possible for an application to be developed without the developers needing worry about the specific hardware provided on a particular machine. Any telephony hardware vendor can then implement the appropriate parts of the TSPI without worrying about what telephony applications have been installed. This separation makes applications and hardware independent of each other. Applications and hardware can come and go without directly affecting each other.

TAPI uses components provided by the operating system or third parties to provide call control functionality. TAPI programs often use media streams such as voice recordings, but TAPI itself does not provide media control. For media control (for example, the recording and playback of voice messages), you must use the APIs and methods appropriate for the media (it will be discussed later)

4.2.1. Synchronous / Asynchronous Operation

The interactive nature of telephony requires that TAPI be a real-time operating environment. Many of TAPI's functions are required to complete quickly and return their results to the application synchronously. Other functions may not be able to complete as quickly and therefore operate asynchronously. Any given operation always completes either synchronously or asynchronously.

4.2.2. Synchronous Functions

An operation that completes synchronously performs all of its processing in the function call made by the application. The function returns different values depending on its success or failure :

- Synchronous Success. If the request or service corresponding to the function has been carried out successfully, the function returns zero, indicating success. Any values computed as a result of the function call are reliable and can be used immediately. Synchronous Failure. If the function detects an error and the request is not carried out, a negative number is returned to identify the error.

4.2.3. Asynchronous Functions

An operation that completes asynchronously performs part of its processing in the function call made by the application and the remainder of it in an independent execution thread after TAPI has returned from the function call. To ensure completion of the call's processing, the service provider vectors the request to another active entity in the system and then returns to the application. At this time, either a negative error result or a positive request identifier is returned to the application.

4.2.4. Callback function

Callback function is a function which is built into the main program. The main program uses it to communicate with the hardware device. Any interrupt from hardware device, which is the highest priority interupt, is received as a message by the callback function. Then, the callback function pass the message to the application for next useful process. For telephony application using telephony device, the callback function receives message from the telephony device. All of messages received by the device is a result of telephony operation. See chapter 5.5. for detail description of callback function related to the application.

4.2.5. TAPI-Compliant Telephony Device

A telephony device is a card that plugs into a computer (or its serial port) and gives it the functions of a telephone. A telephony device can answer the phone, place calls, transmit and recognize tones, and more. Some telephony devices are fax machines on a card, and

some also work as data modems. A voice modem is a combination telephone and data modem.

TAPI-Compliant means that a device's manufacturer distributes a driver, called a Telephony Service Provider (TSP), that makes it compatible with Microsoft's Telephony Application Program Interface (TAPI). TAPI serves as a standard interface to TAPI-compliant hardware, but is woefully complex itself. Not every telephony device is TAPI-compliant. A device can be TAPI-compliant in some aspects but non-compliant in others.

4.3. Microsoft UniModemV

Windows 95 can support multiple TSPI packages installed at the same time. Windows 95 comes standard with UniModem, a very simple TSPI for the most basic MODEM functions.

Microsoft recently announced the availability of UniModemV for Windows 95, a highly improved general purpose TSPI for MODEMs. UniModemV is available today as a download software package from different Microsoft sources. The latest version of UniModemV provided by Microsoft and used in the application is version 4.10.1343.

The most important differences between UniModem and UniModemV is handling voice functions like playing and recording a wave file.

Although the newest general TSPI provided by Microsoft can handle voice functions like play and record sound file, but there are many limitations to providing high quality telephony application. The big problem is the sound quality that supported by UniModemV only one, i.e. mono, 8,000 Hz and 16-bit sampling.

Another thing that will be needed for implementing telephony application is a modem.inf file, tailored for telephony hardware that will be used. The telephony device used for this project is Zoltrix Internal Speaker Phone Voice Modem, therefore, the modem.inf is provided by Zoltrix International Inc.

4.3.1. Modem.inf Files Play a Key Role

The modem.inf file is a key component in this process. It is the link between the Microsoft TAPI components and your specific hardware. This file includes a variety of data lines that tell UniModemV how to handle telephony signal. The modem.inf files are always tailored to specific vendor hardware and model numbers.

4.4. Media Access and Multimedia API

The media mode is the form in which data is transmitted on a line. The four main types of media mode are voice, speech, fax, and data. With TAPI, calls can be established

independently of the call's media mode. The media stream is the actual stream of information that travels on the line. Phone devices and calls on line devices are capable of carrying media streams. The TAPI line and phone device provide a wide range of control operations for these devices, but access to the media stream itself is not provided by TAPI. Instead, the application must use other APIs for the Windows environment to access or manage these media streams. These APIs include the Waveform API, the Comm API, and the Media Control Interface (MCI). The Waveform API is used for multimedia programming, the MCI provides a high level generalized interface of controlling media services, and the Comm API is the set of communications functions provided by the Microsoft. For example, for line devices, an application can use TAPI to establish a connection to another party. Once the connection is established, the application can use the Waveform API (or the MCI Waveaudio API) on the associated device to play back (send) and record (receive) audio data over the connection.

4.4.1. Waveform Audio

Waveform audio is the workhorse of PC multimedia sound. With waveform audio, anything can be done within the practical limitations of memory, disk storage, processor speed, and the capabilities of a sound card.

Waveform audio is a digital medium. It supports a variety of formats, from 8-bit monophonic at a sample rate of 11,025 samples per second (11,025 bytes per second) to 16-bit stereo at a rate of 44,100 samples per second (176,400 bytes per second). The waveform data format - in other words, the sampling rate, number of channels (mono versus stereo), and bit resolution (8 versus 16) - should depend not only on the capabilities of the sound card or other cards (like voice modem) on which the sounds were developed, but also on the capabilities of the sound card or other cards on which they must eventually play.

The Windows 95 multimedia system provides both low-level and high level sets of functions. The low-level functions call device drivers and require more programming. The high-level functions, on the other hand, hide the details as they pass messages to the MCI, which in turn interprets the messages and calls the low-level functions to access the appropriate device drivers.

CHAPTER 5

APPLICATION FLOW

5.1. Initialization

The first part of application is initialization part. It will assign every setup variable with initial value, retrieve setup variable's value from ID file, assign wave device's format with default value and do TAPI initialization.

5.1.1. ID file

As described in the beginning part of this documentation, the application has capability to create up to 5 voice mailboxes. Each mailbox could contain up to 100 voice mails. The number of voice mails for every mailbox will be kept in indexed global variable `gintNoOfMsg`. Every time the application receives voice mail, `gintNoOfMsg` will be increased by one. If `gintNoOfMsg` reaches the maximum number of voice mail (100), it will be assigned to 1. ID file will keep the number of each voice mailbox. As the gintNoOfMsg changes from 100 to 1, all the voice mail related to that voice mail number still exist until the user delete them manually by using Windows 95 file management.

Default value of the number of voice box is 1. It only could be changed by opening Voice Mailbox Setup Menu. This value, then, will be kept in ID file.

ID file also keeps password for voice mailbox. The password will be used when retrieving voice mails with telephone. Default value for password is "1234" but user has chance to change it later either from telephone or by opening Voice Mailbox Setup Menu.

5.1.2. Wave device's format

One of the drawbacks of typical voice modems (like described on previous chapter) is limitation to handle audio data. They could handle only wave format that is mono, 16 bit sampling and 8 Khz. To set this value up as a default format of wave handling, it must be included in initialization part.

5.1.3. TAPI Initialization

The big part of initialization part is TAPI initialization. Before using TAPI, the application must initialize the parts of the API that will be used. TAPI has two parts : the line and the phone. Each is initialized separately via `lineInitialize` and `phoneInitialize`. Since the application will use both of them, the line and the phone part of TAPI needs to be initialized. The `lineInitialize` function looks like this :

```
lineInitialize(
      lngptrhdlLineApp,
      lnghdlLineInstance, _
      lngadrLineCallBack,
      "Project Line",
      lngptrNumLines)
```

The `phoneInitialize` function looks like this :

```
phoneInitialize(
      lngptrhdlPhoneApp,
      lnghdlPhoneInstance,
      lngadrPhoneCallBack,
      "Project Phone",
      lngptrNumPhones)
```

The `lngptrhdlLineApp` and `lngptrhdlPhoneApp` that are returned from these call are application's licenses to use the line or phone part of the API, that is, the TAPI functions with the "line" or "phone" prefix. The "Project Line"and "Project Phone" are for use in other applications if they are interested in where a call was originated. The last parameter `lngptrNumLines` and `lngptrNumPhones` are pointers to address locations that will be filled with the number of line devices available to the application. The most important parameter is the pointer to the line callback function. This callback function is called by TAPI for any telephony event of concern to the application. See "Application Notifications" part of previous chapter.

5.1.4. API Negotiation

The first part of proper TAPI initialization is to call `lineInitialize` and `phoneInitialize`. The second part is to negotiate version numbers for all the lines and the phones the application are going to use.

Over time, different versions may exist for TAPI, applications, and service providers for a line or phone. New versions may define new features, new fields in data structures, and so on. Version numbers therefore indicate how to interpret various data structures.

To allow optimal interoperability of different versions of applications, versions of TAPI itself, and versions of service providers by different vendors, TAPI provides a simple, two-step version negotiation mechanism for applications. Two different versions must be agreed on by the application, TAPI, and the service provider for each line device. The first is the version number for Basic and Supplementary Telephony and is referred to as the API version. The other is for provider-specific extensions, if any, and is referred to as the extension version.

For basic and simple TAPI application, those parts are done via the `lineNegotiateAPIVersion` and `phoneNegotiateAPIVersion` functions :

```
lineNegotiateAPIVersion(
        lngptrhdlLineApp,
        Line,
        &H10000,
        &H10004,
        lngptrAPIVersion,
        lngptrlineExtId)

phoneNegotiateAPIVersion(
        lngptrhdlPhoneApp,
        phoneNumber,
        &H10000,
        &H10004,
        lngptrAPIVersion,
        lngptrPhoneExtId)
```

&H10000 means the low version number of TAPI that could be supported is 1.0
&H10004 means the high version number of TAPI that could be supported is 1.4

The version number is made up of a major and a minor version stored in the high- and low-order words, respectively. The `lineNegotiateAPIVersion` and `phoneNegotiateAPIVersion` function looks at the range of functionality that the application supports and tries to match it with the TSP that controls the line (as specified by the second parameter) and TAPI itself. For example, a Windows 3.x TSP installed on Windows 95 will support only TAPI 1.3 functionality. TAPI under plain Windows 95 supports versions 1.3 and 1.4. A Win32 telephony application could support versions 1.3, 1.4, and 2.0. The result of this negotiation would be the highest version that all three parties support, that is, 1.3.

5.1.5. Line and Phone Capabilities

Once TAPI has been initialized, a suitable line for a call must be found. To discover the capabilities of a line, an application calls the `lineGetDevCaps` and/or `phoneGetDevCaps` functions to fill in the `LINEDEVCAPS` and/or `PHONECAPS`, respectively.

These functions tell whether the line or phone device support the functionality needed by the calls to be made or received, such as their required media mode. These functions are also used to get the name of the line or phone device.

Before obtaining the capabilities of line and phone devices application has to prepare enough memory location to hold them. LINEDEVCAPS and PHONECAPS structure, like nearly every structure in TAPI, are of variable length. Therefore, programming languages that do not have capabilities to assign variable-length structures, like Visual Basic, will have difficulty to precisely allocate them. The best way that could be used is to allocate enough space for fix fields and allocate some spaces for variable fields. The space for variable fields is at least 1 byte for each variable.

5.1.6. Opening Line and Phone Device

After obtaining the capabilities of a line, application must open the line device before it can access telephony functions on that line. This is done by calling lineOpen and phoneOpen functions. When a line or phone device has been opened successfully, the application receives a handle for it. The name of parameters for these handles are lngptrhdlLineApp and lngptrhdlPhoneApp.

```
lineOpen(
      lngptrhdlLineApp,
      lngLineID,
      lngptrhLine,
      &H10004,
      &H0,
      lngadrLineCallBack,
      LINECALLPRIVILEGE_OWNER,
      LINEMEDIAMODE_AUTOMATEDVOICE,
      udtLineCallParams)

phoneOpen(
      lngptrhdlPhoneApp,
      lngPhoneID,
      lngptrhPhone,
      &H10004,
      &H0,
      lngadrPhoneCallBack,
      PHONEPRIVILEGE_OWNER)
```

The two important parameters for the lineOpen function that must be used when calling those functions are Privilege and Media mode. In that functions those are represented by LINECALLPRIVILEGE_OWNER and LINEMEDIAMODE_AUTOMATEDVOICE.

LINECALLPRIVILEGE_OWNER means that call will have privilege as an owner. An owner is an application that can affect the state of a call. Owner receives notifications for all events associated with the call. An application is an owner of a call when it places a

call. Optionally, an application can be the owner of an incoming call on a line for which it set the proper privileges.

`LINEMEDIAMODE_AUTOMATEDVOICE` allows application to automatically handle any incoming calls.

5.1.7. Setting Status Messages

TAPI sends the application numerous messages dealing with status changes of a line or an address. `LineSetStatusMessages` and `phoneSetStatusMessages` are used to allow the application to set which notification messages it wants to receive for events related to status changes for the line and the phone device.

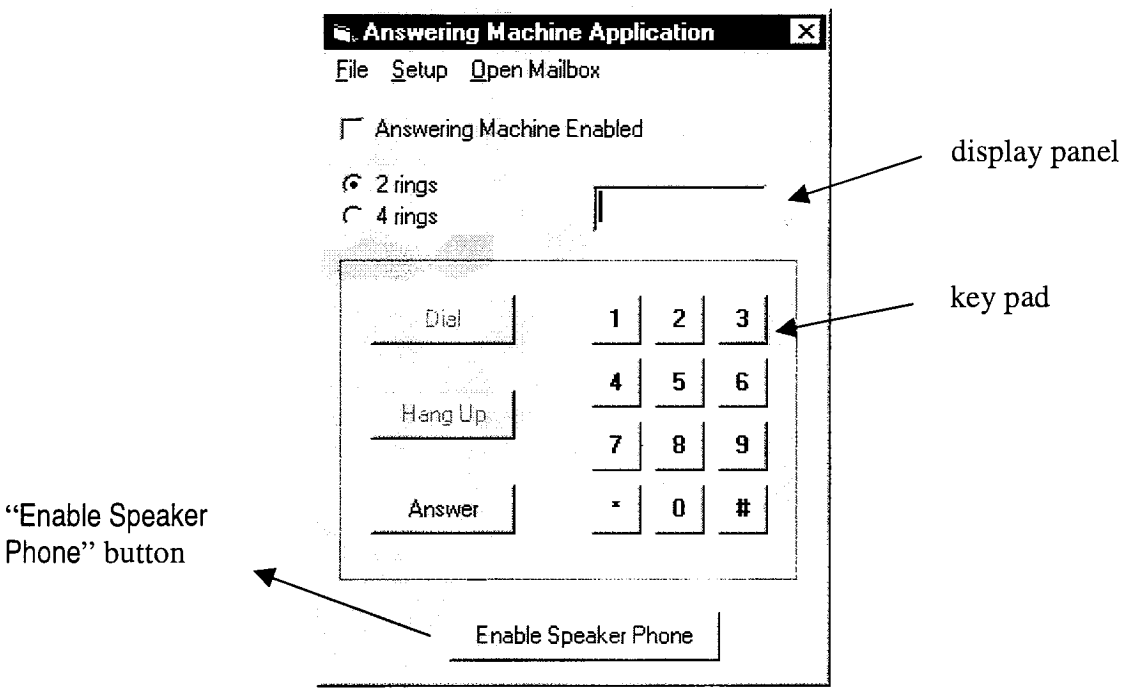

Figure 5.1. Display panel, keypad, and "Enable Speaker Phone" button

5.2. Establishing a Call

5.2.1. Entering the Phone Number

Once the application has determined the lines available for making a call, it is ready to take the phone number from user. There are 2 ways to enter phone number that could be used by user. The first one is by entering the number directly into display panel on the top of keypad. The second one is by clicking the number on keypad (Figure 5.1. Display panel, key pad, and "Enable Speaker Phone" button). From the keypad, the application

allows user to enter numbers (0 to 9), * (star), and # (pound) keys only. The display panel will display numbers only. Any letters entered by the user will be converted to numbers. The application maps alphabet keys entered by user to number based on conversion in this table.

Alphabets	Will be map to
A , B, and C	2
D, E, and F	3
G, H, and I	4
J, K, and L	5
M, N, and O	6
P, Q, R, and S	7
T, U, and V	8
W, X, Y, and Z	9

Table 5.1. Alphabets to number map

5.2.2. Filtering Keyboard Hit

In order to supply a valid telephone number to TAPI, the application must filter keyboard pressed by user. The application uses Visual Basic's internal function KeyPress to know which key the user pressed. The parameter that will be received from KeyPress every time a key is hit is KeyAscii. KeyAscii is ASCII code of the key hit by user. The application will only allow keys with following ASCII code

ASCII code	Key
35	"#"
43	"*"
48 – 57	"0" to "9"
65 – 89	"A" to "Y"
97 – 121	"a" to "y"

Table 5.2. Valid keys and their ASCII code

5.2.3. Making A Call

Once the application has opened the line device, it places the call with `lineMakeCall`, specifying address in the `phoneNumber` parameter and the media mode in the `udtLineCallParams` parameter.

```
lineMakeCall(
      lngptrhLine,
      lnghdlCall,
      phoneNumber,
      0,
      udtLineCallParams)
```

This function returns a positive request ID if the function will be completed asynchronously, or a negative error number if an error has occurred. If dialing completes successfully, messages are sent to the application to inform it about the call's progress. Later, when the `lineMakeCall` function has successfully set up the call, the application receives a `LINE_REPLY` message (the asynchronous reply to `lineMakeCall`). This message informs the application that the call handle returned by `lineMakeCall` is valid. No call operations can be made with that call handle until the `LINE_REPLY` has come in with a 0 on second call back function's parameter.

As the call is placed, it passes through a number of states, each of which results in a `LINE_CALLSTATE` message sent to the application. These states include dialtone, dialing, ringback, and, if connection succeeds, `LINECALLSTATE_CONNECTED` (Figure 5.2. Process call for an outbound call)

5.2.4. Listening to the Conversation

To have a conversation with another party when making an outbound or answer an incoming call the user must activate the microphone and speaker as a telephone set. To do that, the user needs to hit the "Enable Speaker Phone" button. (Figure 5.1. Display panel, key pad, and "Enable Speaker Phone" button)

5.3. Receiving a Call

5.3.1. Incoming Call Detection

As stated before, to be able to detect and receive an incoming call, the line device must be opened as an Owner (`LINECALLPRIVILEGE_OWNER`).

The application is informed of call arrivals with the `LINE_CALLSTATE` message. This message provides the call handle, the application's privilege to the call, and the call's new state. For an unanswered incoming call, the call state is `offering`. Before answer the call, application will invoke `lineGetCallStatus` which allows the application to obtain the current status of a call. This is contained in a `LINECALLSTATUS` structure that holds information such as the current call state and calls' features information.

With the call back function, the application can detect the arrival of incoming call. The application recognizes it by trapping the LINE_CALLSTATE message that contain LINECALLSTATE_OFFERING value.

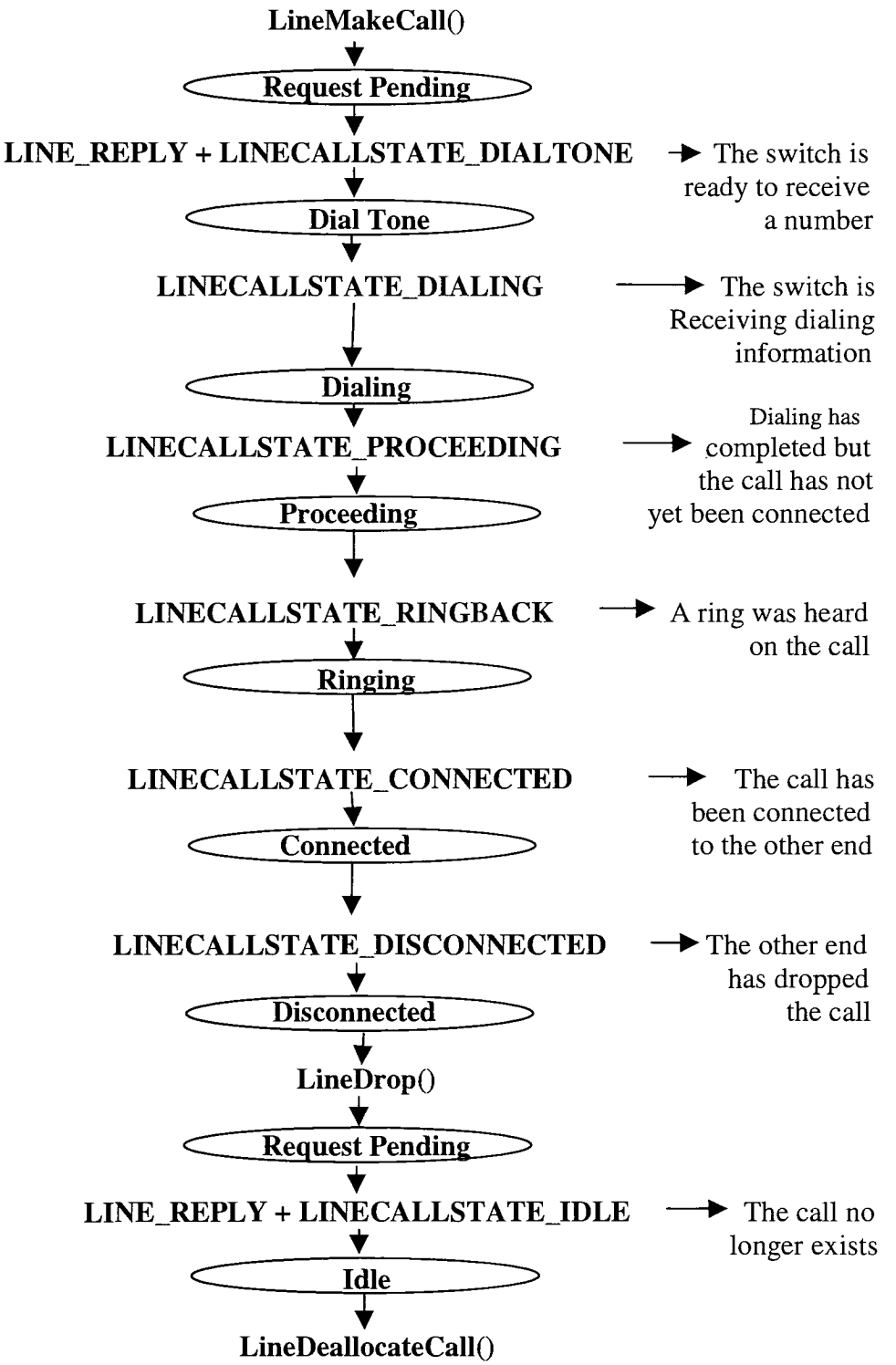

Figure 5.2. Process Call for an Outbound Call

5.3.2. Answering a Call

Once incoming call is detected the call can be answered. Answering an incoming call can be done manually by the user (press the answer button) or automatically by the application. To answer the call manually, user must uncheck the "Answering Machine Enabled "option (Figure 5.3. Answer button and "Answering Machine Enabled" option). By checking this option user intends to let the application answer the call automatically. In other words, the application acts as an answering machine.

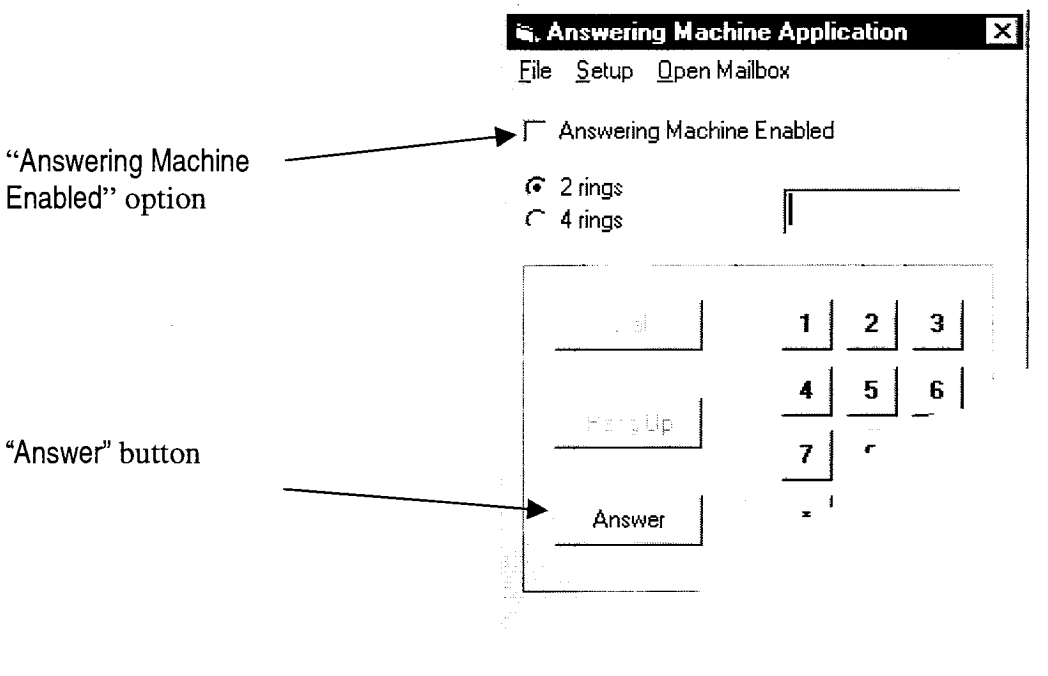

Figure 5.3. "Answer" button and "Answering Machine Enabled" option

For both methods of answering an incoming call, application uses the `lineAnswer` function to answer it.

5.3.3. Answering an Incoming Call Manually

In order to answer an incoming call manually user must hit Answer button. In this condition user should be able to talk and listen to the voice from the other party. In order to do that user has to activate microphone and speaker as a telephone set. By pressing "Enable Speaker Phone" user completes the steps (Figure 5.1. Display panel, keypad, and "Enable Speaker Phone" button).

5.3.4. Answering an Incoming Call Automatically

By checking "Answering Machine Enabled" option user intends to activate the application as an Answering Machine. The first step that will be done by the application is to obtain wave device ID that will be able to play and record waveform audio data. To do this job the application calls the `lineGetID` function.

The application calls `lineGetID` for obtaining wave device ID for playing waveform audio data by entering "wave/out" in the last parameter and for obtaining wave device ID for recording waveform audio data by entering "wave/in" in the last parameter. The information returned from the function will be kept in `VARSTRING` structure. The application must find the wave device ID by fetching the memory content starting from "`String Offset`" memory location for "`String Size`" bytes.

The function calls of `lineGetID` with "wave/in" and "wave/out" parameter look like this:

```
lineGetID(
      0,
      0,
      lnghdlCall,
      LINECALLSELECT_CALL,
      udtVarstring,
      "wave/in")

lineGetID(
      0,
      0,
      lnghdlCall,
      LINECALLSELECT_CALL,
      udtVarstring,
      "wave/out")
```

5.4. Managing Waveform Audio Data over the Telephone Connection

TAPI was designed from the ground up for call control, but it has no facility for direct access to the media available on a call. The media is the data available, for example, modem, fax, network, and voice. The designers of TAPI had several options from which to choose when deciding how to provide access to media via TAPI. One was to provide a way to access a specific API handle for use with an existing or future media API.

Once the call established, the application can obtain the media device such as communication, wave, or midi to manage the data over the telephone connection by calling `lineGetID`

Windows 95 provides a number of functions to manage multimedia data such as waveform audio data. These functions are categorized in 3 levels of multimedia API. The easiest level is the high level API that can be called through MCI API. Despite the lack of the capability of high level functions to precisely control the recording and playback of waveform audio, MCI functions still have enough features to manage it completely.

The function that will be used by the application is `mciSendCommand`. Parameters that will be needed by this function are ID of wave device, the command itself and parameters for this command.

5.4.1. Playing Waveform Audio Data

Other features provided by the application are playing announcement when the caller dials in and playing messages when the owner hits appropriate buttons. To perform these tasks the application must be able to play waveform audio data.

Steps for playing waveform audio data are:

Opening the wave device and obtaining ID that could be used by the application.
Setting up the opened wave device with specific wave format (Chapter 4.1.2.)
Playing the wave file
Closing the wave device.

5.4.2. Recording Waveform Audio Data

When the caller wants to leave a message into appropriate voice mailbox the application must be able to record the caller's voice as a waveform audio data into a wave file.

Steps for recording waveform audio data are :

- Opening the wave device and obtaining ID that could be used by the application
  Setting up the opened wave device with specific wave format (Chapter 4.1.2.)
  Start recording waveform audio data for a period of time (for example 10 seconds)
  Saving the recorded waveform audio data into wave file.
  The wave file's name is : "newmsgx-yy.wav"
  Where x is the voice mailbox number and yy is the message number.
  Closing the wave device.

5.5. Callback Function

As the engine of a TAPI application, the callback function plays the most important role of the system which listens to every hardware notification from the telephony device. The hardware notification received either as a result of completion of an asynchronous function or other telephony operation.

In order to use and manage the notification, the callback function filters the messages received from the hardware and returns them to the application. The messages filtered by the callback function are `LINE_CALLSTATE`, `LINE_REPLY`, and `LINE_MONITORDIGITS`. The `LINE_CALLSTATE` message is received when the status of the call changes. The `LINE_REPLY` message is received after an asynchronous function completes. The `LINE_MONITORDIGITS` message is received when a digit that is being monitored is detected.

5.6. Detecting Digits

To accomplish the task that will be wanted by the user, application must be able to detect button hit by the user. The application uses the `lineMonitorDigits` function to detect the button. This function is called by the application right after the call is answered, which means the connection was created.

The important parameter needed by `lineMonitorDigits` is digit mode. The application is using `LINEDIGITMODE_DTMF` for this parameter. It means digits are detected as Dual Tone Multiple Frequency (DTMF) tones. Valid digits for DTMF tones are 0 through 9, *, and #. DTMF tones are generated through Touch-Tone phones.

When the user presses the keypad, a LINE_MONITORDIGITS message is sent to the callback function. The application will check whether or not the digit is a DTMF digit. If a DTMF digit is detected the application will filter the digit and perform some operation based on it.

5.6.1. Digit Options

The application creates some different option to be used by the user. To indicate the appropriate option by telephone, user must push the specified button. The complete digit options created by the application are:

Buttons 1, 2, 3, 4, or 5 : to record a message to mailbox
Button 8 : to play message(s)
Button 9 : to change voice mailbox's password
- Button * : to playback the next message
Button # : to disconnect the call

5.6.2. Digit States

The application creates different states to identify the digit states of each button hit by the user. The digit states related to message playing are `OPEN_MESS`, `PLAY_MESS`, `MESS_COMPL`. The `OPEN_MESS` state means user wants to open the voice mailbox to listen to the message. User needs to enter the password of voice mail-box to be able to do

this. The PLAY_MESS state means user starts playing back the message. The MESS_COMPL state means user complete playing one message.

The digit states related to password changing are ORIGINAL_PASS, NEW_PASS, and NEW_PASS_COMPL. The ORIGINAL_PASS state means user intends to change the password of voice mail-box and must enter the original password first. The NEW_PASS state means user starts entering new password. The NEW_PASS_COMPL state means user complete entering the new password.

5.7. Managing Voice Mailboxes

The application provides capabilities to manage voice mailboxes. The first thing that could be done by the user is to create up to 5 voice mailboxes. To perform this task user must open Voice Mailbox Setup Windows by clicking Setup option (Figure 5.4. Setup and Open Mailbox Options). However, before creating the voice mailbox user must enter the password of it. (Figure 5.5. Voice Mail Box Setup Windows)

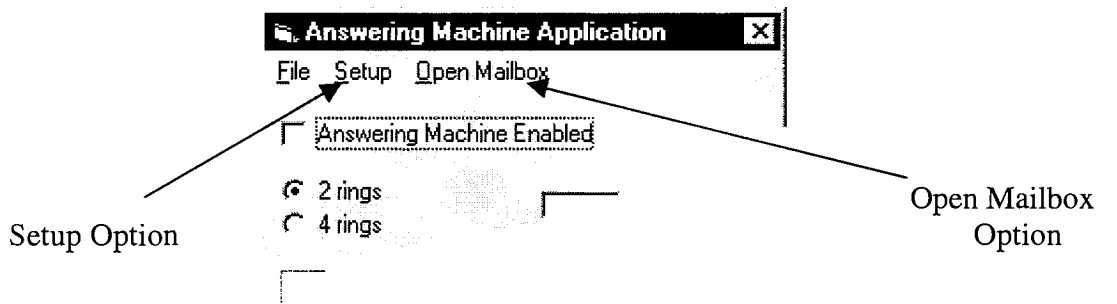

Figure 5.4. Setup and Open Mailbox Options

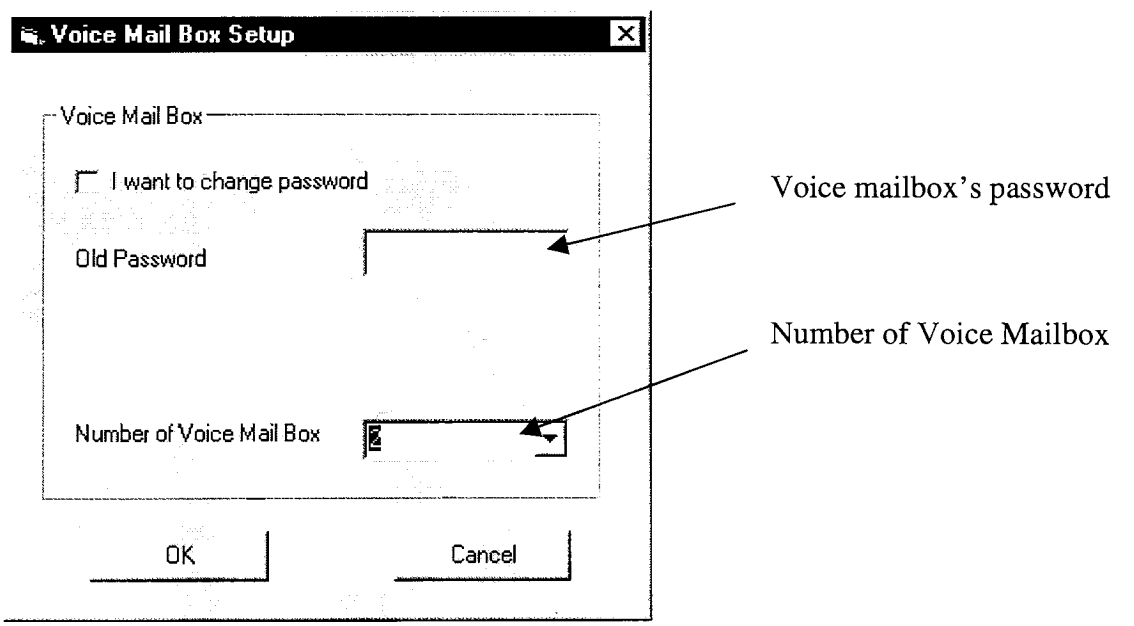

Figure 5.5. Voice Mail Box Setup Windows

The second thing that could be done by the user is playing and deleting the message(s). To perform these tasks user must open Voice Mailbox windows (Figure 5.6. Voice Mailbox windows) by clicking Open Mailbox option (Figure 5.4. Setup and Open Mailbox options).

To play and delete the message(s) user must select the voice mail of the appropriate voice mailbox and then click Play Message or Delete Message buttons

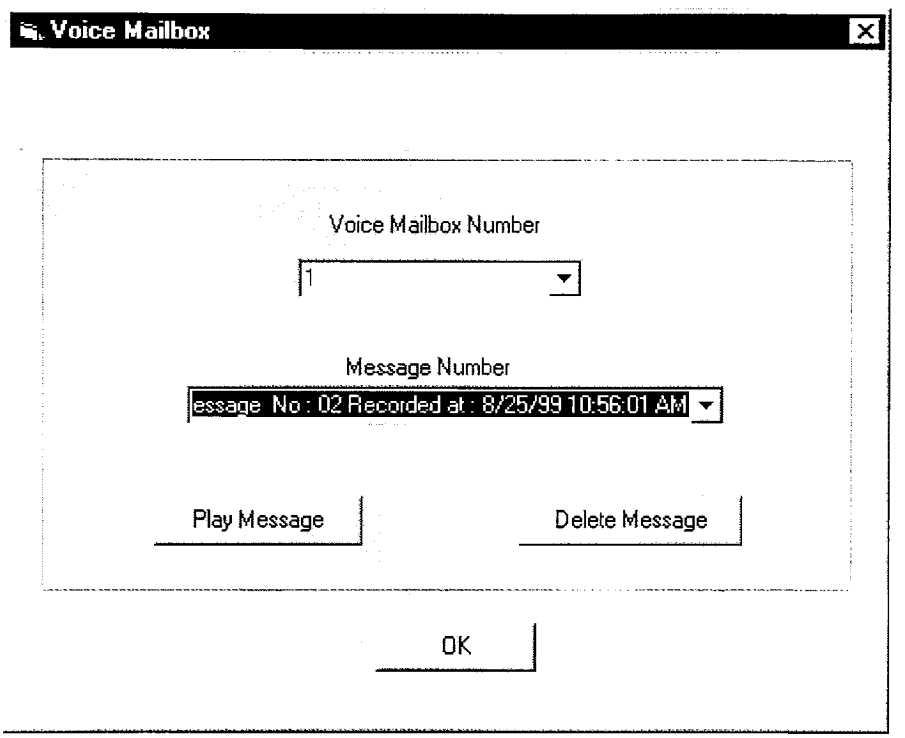

Figure 5.6. Voice Mailbox Windows

5.8. Ending a Call

Call disconnection could be made either by the caller (human) or by the called machine. If the caller disconnects the call, it activates `fnHangup` function. `FnHangup` function calls `lineDrop` function which is used to terminate the end of a call. This does not destroy the call, but it does cause the logical connection between the two ends of the call to be severed. When a call has been successfully dropped, its state will change to idle. This state transition will be detect by the callback function

Once Callback function recognizes the idle state, it will deallocate the call by calling `lineDeallocateCall`. Deallocating the call is the way to free memory that was allocated for the idle call.

5.9. Closing TAPI

In order to close TAPI application, the steps that must be performed are closing the line and shutdown the TAPI itself.

5.9.1. Closing the Line/Phone

Once the call has been deallocated and the line or the phone is no longer needed, the application should close the line or the phone using the `lineClose` or `phoneClose` function.

Once the line or the phone has been closed the line handle or the phone handle is no longer valid and the line call back function will receive no more events for that line.

5.9.2. Shutdowning the TAPI

When the application has no more use for any part of TAPI, the application must call the `lineShutdown or phoneShutdown` function. Parameters needed for these functions are line usage handle or phone usage handle. After the completion of these functions, the usage handles are no longer valid and the application may not use TAPI functions beginning with the "line" or "phone" prefix.

CHAPTER 6

CONCLUSIONS

Conclusions of this project are :

1. Visual Basic can be used to develop telephony application using Windows TAPI because it provides enough functionality to access Windows API. Before using TAPI functions developer must correctly use its' parameters, that is the number and the type of each.

2. Not all voice modems can be used to develop telephony application using Windows TAPI but TAPI-compliant voice modems can be used

3. No changes were needed by the application when it was tried with more than one brand of TAPI-compliant voice modem, Zoltrix and Supra Express. It proves that TAPI is device-independent.

4. By using Telephony Service Provider provided by Microsoft, UnimodemV, TAPI compliant voice modem only able to handle waveform operation for one format (mono, 16 bit, 8 Khz)

CHAPTER 7

FUTURE ENHANCEMENTS

Based on the experience received from the application, it will give better functionality with following enhancement :

1. Implementation of callback function for waveform audio playing or recording. By implementing call back function, caller could interrupt the announcement playing or message recording by pressing the telephone button. Therefore, the caller does not need to wait for the whole waveform audio process.

2. To improve the quality of waveform audio data, the application should implement better audio digitizing and compressing.

3 Add possibility to change digit option easily. With this new feature the owner of application could add the menu option or even totally change it without having any trouble with hard-coded digit option.

4 Add more setup facility. With this new feature the application will provide choices to the owner regarding the number of lines devices, the number of telephony devices, or the types of connection between computer and local exchange.

APPENDIX

TAPI Functions That Have Been Used :

Function's name	Description
lineInitialize	Initialize Telephony API line abstraction for use by the invoking application
lineGetDevCaps	Returns the capabilities of a given line device
lineNegotiateAPIVersion	Allows an application to negotiate an API version to use
lineOpen	Opens a specified line device for providing subsequent monitoring and/or control of the line
lineDrop	Disconnects a call, or abandons a call attempt in progress
lineDeallocateCall	De-allocates the specified call handle
lineClose	Closes a specified opened line device
lineShutdown	Shuts down the application's use of the Telephony API line
lineMakeCall	Makes an outbond call and returns a call handle for it
lineGenerateDigits	Generates digits on a call
lineGetID	Retrieves a device ID associated with the specified open line, address or call.
LineSetNumRings	Indicates the number of rings after which inbound calls are to be answered
LineGetNumRings	This function returns the minimum number of rings requested with lineSetNumRings.
LineAnswer	Answers an inbound call
LineMonitorDigits	Enables or disables digit detection notification on a specified call
phoneInitialize	Initializes the Telephony API phone device for use by the invoking application

phoneGetDevCaps	Returns the capabilities of a given phone device
phoneNegotiateAPIVersion	Allows an applicatin to negotiate an API version to use
phoneOpen	Opens the specified phone device, giving the application either owner or monitor privileges
phoneClose	Closes a specified open phone device
phoneGetHookSwitch	Queries the hookswitch mode of a hookswitch device of an open phone device
phoneSetHookSwitch	Sets the hookswitch mode of one or more of the hookswitch device of an open phone device.

REFERENCES

Amundsen, M. (1996). MAPI, SAPI, and TAPI developer's guide. Indianapolis, IN. : Sams Publishing

Appleman, D. (1997). Dan Appleman's Visual Basic 5.0 programmer's guide to the Win32 API. : Que.

Appleman, D. (1999). Dan Appleman's Win32 API puzzle book and tutorial for Visual Basic programmers. San Francisco, CA. : Apress.

Coppola, C. D., Jarol, S., & Potts, A. Visual Basic 5 web & multimedia adventure set Scottsdale, AZ. : The Coriolis Group, Inc.

Davis, T., Eaton, J, Goertz, R. M., & Simon, R. J. (1996). Windows 95 multimedia & ODBC API bible. Corte Madera, CA: Waite Group Press.

Microsoft. Unimodem V Drivers for Windows 95. http://www.microsoft.com/hwdev/modem/#uDDK

Pennypacker, B. My unofficial TAPI FAQ. http://members.tripod.com/~tapifaq/

Sells, C (1998). Windows telephony programming : a Developer's guide to TAPI. Reading, MA. : Addison Wesley.