Telephony-based email application
Perdana Darta

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Telephony-based Email Application

By

Perdana Darta

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: Perdana Darta

Student Number: __________________

Thesis Title: Telephony-based Email Application

Project Committee

<table>
<thead>
<tr>
<th>Name</th>
<th>Signature</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prof. Rayno Niemi</td>
<td>8/25/99</td>
<td></td>
</tr>
<tr>
<td>Chair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mr. Robert Hermanns</td>
<td>8/28/99</td>
<td></td>
</tr>
<tr>
<td>Committee Member</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prof. Alec Berenbaum</td>
<td>8/25/99</td>
<td></td>
</tr>
<tr>
<td>Committee Member</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table of Contents

I. Abstract 1

II. Goals of the Project 2

III. Project Overview 3
   III.1. Hardware Requirements 3
       III.1.1. Voice Modem Driver 3
   III.2. Programming Language 3
       III.2.1. TAPI Browser ver. 1.4 4
   III.3. Windows Application Programming Interface (API) 4
       III.3.1. Windows API Text Viewer 5
       III.3.2. Implementations of Windows APIs 5

IV. Functional Design 8
   IV.1. Configuration Steps 10
   IV.2. Place an Outgoing Call 14
   IV.3. Handle an Incoming Call 16
       IV.3.1. Answer an Incoming Call Manually 16
       IV.3.2. Answer an Incoming Call Automatically 16
   IV.4. Send the Caller’s Message to an Email Address 18
   IV.5. Call Administrations 22
       IV.5.1. Playback New / Old Messages or Greeting Messages 22
       IV.5.2. Delete Old Messages 23
       IV.5.3. Change the Announcement (Greeting Message) 23

V. Telephony Application Programming Interface (TAPI) 25
   V.1. Basic Typical TAPI Functions 25
   V.2. Media Stream 26
   V.3. Data Structure 26
   V.4. Initialize TAPI 27
   V.5. Monitor Messages from TAPI 27
   V.6. Open Line / Phone Devices 27
   V.7. Receive Calls 28
   V.8. Monitor Digit DTMF 28
   V.9. Place the Call 28
   V.10. Set Up the Phone Hook Switch 29
V.11. Toll Saver 29
V.12. Drop the Call 29
V.13. Deallocate Call Handles 29
V.14. Close Line / Phone Device 30

VI. Multimedia Control Interface (MCI) 31
VI.1. MCI Command Messages 31
VI.2. WAV Audio File Format 32
VI.3. Detecting a Proper Recording and Playback Modem Device 32
VI.4. Recording a Message 32
VI.5. Saving a Message 33
VI.6. Playback an Announcement (Greeting Message) 33
VI.7. Detect and Display MCI Error Codes 34

VII. Remote Access Service Application Programming Interface (RAS API) 35
VII.1. Starting a Connection 35
VII.2. Terminating a Connection 35
VII.3. Detecting RAS Error Messages 36
VII.4. Timer 36

VIII. Mailer 37
VIII.1. Windows Sockets (Winsocks) Control 37
VIII.1.1. Multipurpose Internet Mail Extensions (MIME) 37
VIII.1.2. Base64 Encoding 38
VIII.1.3 MIME – base64 Encoding 38
VIII.2. Messaging Application Programming Interface (MAPI) as a MAPI Client 39
VIII.2.1. MAPI Client – MAPI Server (Netscape Messenger) 40

IX. Conclusions 41

Citations

Appendix

Attachment “Incoming Call Flow Chart”
I. Abstract

The aim of this project is to provide user to gain access of telephony services such as placing an outgoing calls, answering calls, playing back an announcement to the callers, recording caller's messages, playing back recorded messages and finally establishing a dial up connection to send the recorded message to an e-mail address.

To do that, the application will be implementing and combining some of Windows Application Programming Interface (API) functions as follows:

1. Telephony Programming Interface (TAPI) ver. 1.4.
3. Remote Access Service (RAS API)
II. Goals of the Project

The primary focus of the above integration is to develop a client-side application (using Visual Basic 6) within the Windows 95 operating system that can function like a telephone answering machine where at the end of the call session the caller can send the recorded message out to an email address.
III. Project Overview

II.1. Hardware Requirements

This application needs an analog line telephone device and runs on a workstation with Windows 95. Besides that the application needs a speakerphone voice-data modem supported by Voice Modem driver and a sound card as well (with external speakers and a microphone) to play back and record callers’ messages.

III.1.1. Voice Modem Driver (Unimodem/Voice version 4.10.1343)

There are two common types of modem drivers i.e. Unimodem and Unimodem / Voice. Both are called a TAPI Service provider (TSP) that is basically a driver to allow TAPI to communicate with telephony hardware.

Unimodem driver is a standard TSP provided by Windows that supports any modem when placing data calls. If the TAPI compliant application is using the Unimodem driver to place a voice call, the user has to determine the progress of the call (whether the call completes or not) by invoking a certain button to end the voice call. It shows that not all modems can handle voice calls.

Instead, Unimodem/Voice, a released version of Unimodem to support voice modems under Windows 95, is designed to answer calls and be able to play/record wave files over the phone line and the handset. It also detects a disconnecting signal when the call session ends so the application does not rely on human intervention to press a certain button when the call completes.

TAPI Browser version 1.4 under Windows 95 will be helpful for the developer to observe exactly what a given TSP is capable of, how it works and its behaviors especially using Unimodem/Voice. The behavior of other service providers will vary depending on the capabilities of the hardware being used.

The developers can check the TSP version (especially Unimodem/Voice) being used in their machine by clicking the “version” tab at the properties of the Unimdm.tsp file in directory Windows/Systems. This application is using Unimodem / Voice version 4.10.1343 that can be downloaded from ftp://ftp.microsoft.com/Softlib/MSLFILES/UNIMODV.EXE

III.2. Programming Language

Even though the Windows API based application can be developed using C++, Java, etc., the author feels more knowledgeable using Visual Basic version 6 than those programming languages above.
The VB 6 also provides some controls which directly manage low level Windows’ API functions. By setting properties and invoking methods of the controls, it is much easier to build Windows’ API-based application.

For example, the VB 6’s Multimedia control (MCI) manages recording and play back messages rather than building low level wave form APIs. The VB 6’s MAPI control provides messaging services to allow an application to compose and send an email rather than building low level messaging APIs. Finally the VB 6's Winsocks control will let the application establish and maintain a direct connection to a remote server so the application can exchange data (i.e. email messages) rather than building low level Winsock APIs.

This application also needs Netscape Messenger version 4.5 or newer (one of MAPI compliant mail servers) when sending an email. It functions as a message transport (mailer the mail server) that delivers an email composed by this application (the mail client) to a remote server.

### III.2.1. TAPI Browser ver. 1.4

To observe how TSP works, the TAPI browser version 1.4 for Windows 95 is a tool to query all available TAPI functions so the user will know how to properly manage and use each TAPI function such as TAPI’s constants, data structures in response to the telephony signal sent by a local exchange. The TAPI Browser (included in this project as an attachment) will return all required parameters in each required data structure. By using this, the user can directly get access to telephony services such as how to place an outgoing calls (lineMakeCalls( )), to query the modem capabilities (lineGetDevCaps( )), to detect total modems in associated machine (lineInitialize( )), etc.

This is an example to make some standard TAPI calls using a single modem:

```vbscript
lineInitialize( ), lineOpen( ) /* use device 0 */, lineMakeCall( ), lineDrop( )
lineDeallocateCall( ), lineClose( ), lineShutdown( )
```

This browser can be downloaded from FTP://ftp.microsoft.com/developr/TAPI)

### III.3. Windows Application Programming Interface (API)

The Windows Application Programming Interface (API) is a set of predefined functions built into the DLLs such as TAPI32.DLL, RasApi32.DLL, winmm.dll, etc. that make up the Windows operating system. A DLL (dynamic link library) is compiled code that can be accessed only by other applications at runtime. This standardizes the programming community, as it allows the same code to be used repeatedly. End users can't directly access these functions. However, developers can access the code written in the DLLs through the API and use this code in their programs. This allows developers to use existing code in the DLLs and save time in the programming development cycle.

To use and access the Windows APIs in Visual Basic, the developers can use the `Declare statement` in the General Declarations section of a module. For example, one function that the application uses lineMakeCall which places an outgoing calls as follows:

```vbscript
lineMakeCall( )
```
Declare Function lineMakeCall Lib "TAPI32.DLL" (ByVal hLine As Long, lphCall As Long, ByVal lpzDestAddress As String, ByVal dwCountryCode As Long, lpCallParams As Any) As Long.

Syntax :
Declare Function: A keyword indicating an external procedure declaration
LineMakeCall: The name of this particular external procedure
Lib: A keyword indicating the DLL in which to locate the function
hLine, lphCall: An argument variable
lpCallParams: which is a Windows-defined type, CALLPARAMS
As String / As Long, As Any: keywords indicating argument type
Long: A function return type

Windows - defined types / structures

A structure is a group of variables organized under one name. This is used to pass information between the Windows API and the requesting application. When certain variables are grouped, the application can create instances of the type, as the application would create a variable. For example: line Make Call function needs the CALLPARAMS data structure. The developers can find the definition of that Windows-defined types within the API Viewer tool provided by VB or within the documentation of the Windows Software Development Kit (SDK), which comes on the Microsoft Developers Network CD-ROM included in VB.

III.3.1. Windows API Text Viewer

To avoid an error when inadvertently writing parameters of Windows API and to save valuable programming time, the developers can make use of the TAPI Text Viewer shipped which Visual Basic (in "add-ins" menu), which enables developers to cut and paste API functions, constants, and Windows-defined types right into the code.

III.3.2. Implementations of Windows APIs

This project uses five Windows APIs, which are briefly described in the following section:

1. The Telephony API (TAPI):

The TAPI is an application programming interface in the Windows platform that allows an application to provide access to telephony services such as, making a call, answering incoming call, detecting and generating DTMF digit from / to local exchange, etc. The TAPI will work together with a voice modem to let the application access to the telephony services without knowing specifics of the telephone network or connection hardware. Instead, the application only
needs to query the TAPI functions to determine the capabilities of both that are actually available.

2. The Windows Multimedia Control Interface (MCI):

This application uses the Windows Multimedia Control Interface to interact with multimedia hardware. Hardware can consist of any media devices that Windows supports i.e. audio CD players, wave audio, etc. The application issues the MCI's standard commands, works with the specific driver (Unimodem/Voice) to support the voice modem device in use, and then translates these commands to the exact signals required to carry out the action such as play back a WAV file, record a callers' message, and save it.

One way to send commands to the MCI device is to use the function mciSendCommand. Although this function is not a part of VB, this function is part of Windows Application Programming Interface (API), the huge collection of functions that provides most of Windows' capabilities.

The second method for sending MCI commands is using Visual Basic's Multimedia control. When displayed on a form, this control provides an array of buttons for controlling a multimedia device.

3. The Messaging API interface (MAPI):

Visual Basic's MAPI control will allow the application to send a message with or without an attachment to a recipient mail address. This application, called the MAPI client, will work together with a MAPI Server (i.e. Netscape Messenger or other mailer) that has responsibility for message transport (moving messages to and from remote servers and clients), message storage (a mailbox before mail client reads the incoming message) and address book (contains a particular mailing address / addressee).

After the callers press # 3 (to send their message to an email address), the application will be activating the VB’s MAPI control, once an internet connection exists.

4. The Windows Sockets (Winsocks)

As an alternative using the MAPI services to send an email, the application can make use of VB's Winsocks control. It allows the application to create and maintain a connection to a remote server by opening sockets on other computers and accessing Transport Control Protocol (TCP) networks services. Using the connection, both computers can transfer data between themselves using the socket which the application has opened.

The application applies one of conversion method to convert binary file (an attachment file) into ASCII text to deliver a message over the Internet.
5. The Remote Access Service (RAS API):

The RAS API functions will let the applications establish and terminate a dial up connection to the remote server without displaying a Dial Up Networking dialog box. The RAS functions will query a connection information such as the name of the phone-book entry (which modem) used to establish the connection, user name and password to log onto remote server, and also remote server's phone number from “Login” dialog box in the “Mail” menu.
IV. Functional Design

This is the main menu when the application runs.

![Main Menu](image)

Figure 4.1. The Main Menu

These are functions that the users can perform:

1. Place an outgoing call.
2. Answer an incoming call (manually – press “Answer” button or automatically).
3. Playback an announcer with three menus to the callers.
   - Press #1: to hang up the call.
   - Press #2: to playback a caller's message.
   - Press #3: to send the recorded message to recipient mail address.
4. Record caller's message and save it in the same folder where the application resides.
5. Create recorded message file name base on the date and time it's recorded.
6. Playback recorded message directly from the application or from the telephone line by press # 2.
7. Delete old messages
8. If necessary user can change the greeting message by recording the new greeting and overwriting to the old one.
9. Automatically launch dial up connection and end the recorded message to specified email address using either Netscape Messenger or Windows sockets.
10. The application will automatically redial up (2 or 4 dial attempts) if it failed to get an internet connection.
IV.1. Configuration Steps

This flowchart details how the user accesses to the functionality of the application (menus). See the flowchart as follows:

1. enumerate, initialize and display the various modem's capabilities (callLineInitialize(), callPhoneInitialize(), callEnumLineDev(), callEnumPhoneDev() in mdlTAPIFunction.bas).
2. count and display unread message(s) that resides in application's folder (Dir() in frmTAPIfrm).
3. detect and display wave audio device - sound card / voice modem (waveOutGetDevCaps() in frmTAPIfrm).
4. terminate all active dial up connections (callHangUp() in mdlTAPIFunction.bas).
5. close Winsocks TCP connection.
6. set application default: manual answer, disabled mail, length of recordable message = 10 secs, number of rings = 2 rings.

---

Run Application

System ON

1. open line / phone device (callLineOpen(), callPhoneOpen()) to get line (lphLine) / phone (lphPhone) handle to be used for subsequent operations.

System OFF

1. terminate all active dial up connections (callHangUp() in mdlTAPIFunction.bas).
2. close Winsocks TCP connection.
3. close an open MIME base64 encoded file.
4. close an open line / phone device (callLineClose(), callPhoneClose() in mdlTAPIFunction.bas).

Select : line, phone, wave device

1. select modem, mic - speaker phone, wave device of the available modem in associate combo box.
2. If not using the default user has to select mic speaker modem device wave device that allows the application to play back and record greeting / incoming message over the telephone line otherwise all greetings will be played back in the local sound card.

For example:
Select : "Voice Modem ..." to play back greetings and record caller's message to / from over the telephone line.
Select : "Sound Card ...." To play back greetings to local sound card and this won't record caller's message.
if "Auto Answer" is checked:

- the application will automatically answer an incoming call based on number of rings (the default is 2 rings).

  - Play back greeting ("greeting.wav") to the caller.
  - After greeting ends it will automatically record caller's message based on predetermined length of message in "Msg Length" of "Mail" menu (the length message default is 10 secs).
  - An incoming message will be saved with the beginning of the file name is "New_& date & time & p/m.wav" based on the date and time it recorded.
  - After saving the recorded message, it plays back the second greeting ("greeting2.wav") to inform the caller if he wants to play back recorded message (by pressing digit # 2).
  - The caller can hang up the call session by pressing digit # 1.
  - If the caller does not make an action within 40 seconds (tmrHangUp in frmTAPI.frm) after the second greeting message played, the application will automatically terminate the call.

if "Enable Mail" is checked:

- The application will play back "greeting23.wav" to tell the caller that he can play back his recorded message (press digit #2) or directly send it out to the owner application email address (press digit #3).
- The application will send it out using either Netscape Messenger or Windows Sockets based on user's choice in "Set Up" of "Mail" menu.
- It takes 15 secs (after caller presses digit #3) before the application fires up dial up connection (callRasDial()).
- User has to provide user name, password and phone number of the remote server in "Set Up" of "Mail" menu. These parameters will be used for establishing dial up connection.
- Once dial up connection established, either Netscape Messenger or WinSocks control will send it out based on mailer parameters (such as destination / originating email address and server name).
- After sending message, it automatically close MAPI services (if using Netscape Messenger) or TCP connection (if using WinSocks control) and terminates the dial up connection (callHangUp()). So the application will be ready to receive an incoming call or place an outgoing call.

Set the number of rings before the application will automatically answer an incoming call (the default is 2 rings)
From "Mail" menu:

- **Recordable Message Length?**
  Set the length of the recordable message for either 10 secs (154 kb) or 35 secs (544 kb).

From "Set Up" in "Mail" menu:

- **Mailer?**
  The default of the mailer is Windows Sockets.
  1. If the user chooses Winsocks as a mailer,
     The user must provide the remote server name.
     The user doesn’t need to configure Netscape Messenger.

  2. if the user choose Netscape Messenger as a mailer:
     User must configure Netscape Messenger first as follows:
     a. Open Netscape (ver 4.5) -> click "Edit" menu -> click "Preferences" ->
        -> click "Mail & Newsgroups" -> locate "Use Netscape Messenger from MAPI based application" and check it if it’s unchecked -> close Netscape.
     b. Make sure that destination email address is already available in Netscape Address Book, if not try to create it in it.
     . Netscape Messenger as a mail server will take 80 secs (based on predetermined timer #2) to send the recorded message. This timer will give an enough time to allow the Netscape Messenger to send the message to the remote server.

From "Set Up" in "Mail" menu

- User must provide all mailer parameters.
  The defaults are available by alternating "Cancel / Default" button.

This button is enabled when there is an incoming call.
If "Auto Answer" is checked, this button will automatically call a specific function as follows (see at mdlTAPILine.bas)

1. CallphoneSetHookSwitch() to enable modem speaker so the user can hear a caller voice and to disable mic of the modem so the caller can't monitor user voice.
2. CallLineAnswer () to open a connection between application and the caller.
3. CallLineMonDigits() to monitor caller response when pressing a certain digit DTMF
4. CallGreeting() to play greeting ("greeting.wav")
5. CallMCIRecMsg() to record caller message for a certain message length (10 secs as a default)
If "Auto Answer" is unchecked, the application will alert user by sounding a ringing call so the user can press this button to answer an incoming call.

- **Dial Pad**
  To place an outgoing call.
  - This action will launch "Dial Pad" window so the user can enter a letter or numeric digit of the phone number (phone # default is 1800Marinel).
  - All letter entries will be converted to numeric value in mdlSupport.bas (callConvertStringToDigits())

- **Hang Up**
  - This action will hang up the call session by calling calllineDropO (see at mdlTAPIFunction.bas)

- **PlayBack Msg**
  - To play back a greeting file or a caller message "New_ & date & time & a/p. wav" - prefixed file name.
  - It will open Multimedia Control Interface (MCI) control and a certain folder where all recorded message and greeting files reside.
  - It will rename the "New_*”.wav” prefixed file name to "Del_*”.wav” prefixed file name (callRenameMsgO in mdlSupport.bas) and also subtract new message counter.

- **Save Greeting**
  - To save a greeting created by the user (see at frmTAPI.frm).
  - It will open a certain folder where the user can save the greeting by overwriting the old greeting (for ex. "greeting.wav", "greeting2.wav" and "greeting23.wav").

- **Delete Msg**
  - To delete all recorded messages ("Del_*”.wav”) that have been played back (callDeleteMsgO in mdlSupport.bas)

**From "Modem Set Up" in "Tools" menu**

- **Modem Set Up**
  - To configure a particular modem (calllineConfigDialogO in mdlTAPIFunction.bas) by launching configuration dialog window.

**From "Exit" in "File" menu**

- **Exit**
  - To exit from the active application. This action will terminate line / phone device, dial up connection (calllineCloseO, callphoneCloseO, callHangUpO), close TCP connection if it exists and unload all open forms (see at frmTAPI.frm).
IV.2. Place an Outgoing Call

1. Run the application (Fig. 4.1. The Main Menu).
2. Select the available modem to be used to place an outgoing call (if the application has more than one modem installed). Once selected this modem will be answering incoming call (Fig. 4.2. Modem Option).

3. Check the "Microphone Speaker" and "Record - Playback Device" check box as a default instead of selecting the microphone speaker and wave device option (Fig. 4.3.1. and Fig. 4.3.2.). This will let the application automatically specify the associated microphone speaker and wave device based on the selected modem (Fig. 4.3. Microphone Speaker and Wave Device Option).
4. Turn on the application (press "System OFF (On)" button).
5. Press "Dial Pad" button and the application will be launching the "Login" pop up window (4.4. Dial Pad).
6. Enter the destination phone number and press "Dial" button.

IV.3. Handle an Incoming Call

If the “Enable Mail” is unchecked, there are two ways the application can handle an incoming call (see attachment “Incoming Call Flow Chart”):

a. Manually : the “Auto Answer” option is unchecked

b. Automatically : the “Auto Answer” option is checked.

IV.3.1. Answer an Incoming Call Manually

The user doesn’t let the application perform as answering machine (it does not record the caller’s message). It means the user can directly handle an incoming call by pressing “Answer” button as shows in the following steps:

1. Repeat step 1 - 4 in IV.1.
2. If the caller calls in, the application will be ringing the user (play back "Ringin.wav") and enable "Answer" button.
3. Press it and get connected to the caller.
4. Press “Hang Up” button to end the call session.

IV.3.2. Answer an Incoming Call Automatically

The user lets the application perform as an answering machine by checking the “Auto Answer” option (Fig. 4.5. Auto Answer – Number of Rings”). The application will playback the greeting messages and record the caller’s message.
1. Repeat step 1 - 4 in IV.1.
2. Check the "Auto Answer" check box.
3. Choose the number of rings (Fig. 4.5. Auto Answer Number of Rings) before the application automatically activate answering machine (Toll Saver).

Figure 4.5. Auto Answer - Number of Rings

4. If the caller calls in, the application will playback an announcement ("greeting.wav") to the caller. This tells the caller to press # 1 to hang up the call or press # 2 to play back message after the message length (10 - the default or 35 seconds) has been elapsed (Fig. 4.6. Length of the Message).

Figure 4.6. Length of the Message

5. The application will be recording the caller's message for 10 or 35 seconds accordingly.
6. Save it the caller's message in the same folder with the application resides and name the file with "New_" prefix follows by the date and time it was recorded (Fig. 4.7. Playback Message).
4.7. Playback Message

7. Playback the second announcement ("greeting2.wav").
8. Playback the caller's recorded message if the caller presses # 2.
9. Hang up the call session if the caller presses # 1 or does on hook

IV.4. Send the Caller's Message to an Email Address

If the “Auto Answer” and “Enable Mail” options are checked, the application will perform steps IV.3.2. and it also allows the caller to send the message to a predetermined email address.

After the caller presses # 3, the application will get ready to send the caller's message to the recipient mail address. Before firing up a dial up connection (after 15 seconds the caller hang up the connection), it will collect all connection information in the "Login" form from the user.

The user can either select Netscape Messenger or Windows Sockets as a mailer (message transport) to move out the message to the remote server. The mailer's default is Windows Sockets. If using the Netscape Messenger, the user must configure the Netscape Messenger. Please click "Instruction" in the "Help" menu (Fig. 4.8. Help Menu) and will guide to do that.
These are steps to enable the application sending the caller's message:

1. Repeat step 1 - 4 in IV.2. and check "Enable Mail" check box (Fig. 4.9. Enable Mail).

Figure 4.8. Help Menu

Figure 4.9. Enable Mail
2. The application will pop up the "Login" window (Fig. 4.9.1. Login Window).

![Login Window](image)

Figure 4.9.1. Login Window

3. Select the mailer (Netscape Messenger or Windows Sockets - the default)
4. Enter the user name, password and phone number to log on to remote server name.
5. Enter the first digit the text box available if the application is connected to the PBX (Fig. 4.9.2. PBX Digit - Dial Attempts).

![PBX Digit - Dial Attempts](image)

4.9.2. PBX Digit - Dial Attempts

6. Specify the maximum number of tones that the application will attempt to dial up to get an internet connection (Fig. 4.9.2. PBX Digit - Dial Attempts). It would be performed if the application (RAS API) failed to fire up an internet connection to the remote server.
7. Specify a modem to be used to dial up an internet connection (Fig 4.9.3. Modem Option).
8. Specify recipient mail address, remote server name and sender address.

Having sent the message out, the application will automatically terminate the internet connection and get ready to answer incoming calls or place outgoing calls. If using Netscape Messenger, the application takes around 80 seconds (after it got a connection) to send a message out.

These are some actions performed by the application when handling an incoming call and followed by sending the message to an email address (see attachment “Incoming Call Flow Chart”):

1. When user presses “System Off / On” button, the application will initialize and open the selected modem, set the number of rings and set the speaker phone on.
2. If the “Auto Answer” and “Enable Mail” options are checked, the application will receive an incoming call after 2 or 4 rings.
3. The caller will be prompted to the announcement containing three menus:
   - Press # 1: to hang up the call.
   - Press # 2: to listen to the caller’s message.
   - Press # 3: to send the message to the email address.
   The application will call callGreeting( ) functions in mdlTAPIFunc.bas
4. After playing back the greeting message to the caller, the application starts recording caller’s message for 10 or 35 seconds based on user selection.
   The application will set up the MCI device with the specific recording audio i.e. standard Windows wave file format (PCM, 16 bit resolution with sampling rate 8 kHz). Finally store the recorded message to the specific file name based on the data and time it was recorded (callMCIsaveMsg( )).
5. If the user presses # 3, the application will terminate the call session. Fifteen seconds after the call session was closed, the application will invoke Ras API functions to establish a dial up connection.
6. After collecting the connection information from the “Login” dialog box, the RasDial( ) function tries to establish a dial up connection to the remote server. If it failed, it redials up to 2 or 4 attempts.
7. If the application successfully makes the dial up connection, it starts invoking the Nestcape Messenger or Windows Sockets to send the message out.
If the user selects the Winsock as mailer, the application will perform a conversion of binary format (attachment file) to ASCII text (in module mdlMIME.bas). On the other hand, if the user selects the Netscape Messenger (MAPI), the application will launch the Netscape and doesn’t need to perform a binary format conversion.

8. After successfully sending the message, the application terminates the dial up connection (RasEnumEntries( ), RasHangUp( )).

9. And finally the application invokes lineDrop( ), lineDealloCall( ), lineClose( ), lineShutDown( ), phoneclose( ) and phoneShutDown( ) to close the call session.

IV.5. Call Administrations

The application offers the user to manage the caller’s message and the greeting messages as follows:

1. Playback a New / Old Message or Greeting Message
2. Delete Old Messages
3. Change the Announcement (Greeting Message)

IV.5.1. Playback a New / Old Message and Greeting Message

1. When starting the application, it will display the new message(s) to the user (Fig. 4.10. Playback New / Old Message and Greetings - user has 1 new message)
2. Press "Playback Msg" button (Fig. 4.10. Playback New / Old Message and Greetings).

Figure 4.10. Playback New / Old Message and Greetings

3. The application will pop up common dialog window so the user can select one of new messages to listen to (4.7. Playback Message).

4. After the user is done to listen it, the application will rename the message file name to be "Del_" prefixed file name (4.7. Playback Message). It will differentiate the new message from the message has been listened.
IV.5.2. Delete Old Messages

1. Just press "Delete" button. The application will delete all old messages ("Del_" - prefixed file name).

IV.5.3 Change the Announcement (Greeting Message)

The user could change the greeting messages ("greeting.wav", "greetings.wav", "greeting2.wav", "greeting23.wav").

1. Press VB's MCI record button to record user's greeting (Fig. 4.11. Record Greeting Message).

2. Press "Save Greeting" button to overwrite the old one and save it (Fig. 4.12. Overwrite and Save the Greeting Message).

Figure 4.11. Record Greeting Message

Figure 4.12. Overwrite and Save the Greeting Message
3. Playback the new greeting message if it’s necessary (press "Playback Msg" button).
V. Telephony Application Programming Interface (TAPI)

The Telephony Application Programming Interface (TAPI) is one of the API sets to be released by Microsoft. The TAPI is a single set of function calls that allows an application to manage and manipulate any type of communication link between the PC and the telephone line(s). An application that uses TAPI can generate a full set of dialing tones (answer an incoming call and place an outgoing call) and flash hook functions on / off hook (like that of the simple analog handset).

TAPI defines several data structures that are used to pass information between TAPI and the requesting application. The layout of the structures contains variable as well as fixed data (see at mdlTAPISource.bas module).

TAPI is sub into two subsets of API calls. Where each API subset (see at mdlTAPISource.bas) refers to:
1. Line devices: a model of the physical telephony lines used to send and receive voice and data between locations, such as fax board, a modem, or an ISDN card.
2. Phone devices: a model of the desktop handset used to place and receive calls.

Two of the significant TAPI functions are lineCallBack and phoneCallBack function. These functions keep track any state changes occurring the telephone network and notify the application accordingly by sending a corresponding message (LINECALLSTATE).

V.1. Basic Typical TAPI Functions

These are the important steps that can be used to receive a call, place a call, end a call, close the line, and leave TAPI (this condition is using a non speakerphone modem, so the application doesn't need to call phone device functions such as phoneSetHookSwitch, phoneInitialize, phoneOpen):

1. LineInitialize( ) to open a TAPI session between the application and TAPI calls.
2. LineOpen( ) to open a specific line device to be used for the next call operations.
3. LineSetNumRings( ) to set number of rings before an incoming call is answered’
4. LineMakeCall( ) / lineAnswer( ) to place an outgoing call or receiving an incoming call.
5. LineMonitorDigits( ) to detect digits DTMF pressed by callers.
6. LineDrop( ), to place the call in the IDLE state. The call still exists, and the application still has its handle.
7. LineDeallocateCall( ) to release the call handle for the finished call. The call no longer exists.
8. If the application expects no more calls on the line, it uses lineClose( ) to close the line. At this point, there will be no more incoming or outgoing calls on that line.
9. LineShutdown( ) to end the use of TAPI's functions for the current session.
V.2. Media Stream

TAPI only provides control for line and phone devices and does not give access to the content of the media stream that consists of the information exchanged over a call.

This application uses Win32 API functions, such as the Remote Access Service (RAS API), Media Control Interface (MCI), Messaging Application Programming Interface (MAPI) or Windows Sockets functions to manage the media stream. Although, this application functions like an answering machine and send the recorded message through an email uses the TAPI functions to control and monitor the incoming call, it uses the MCI functions to record and save the message, RAS API functions to launch the dial up connection to remote ISP's server and then finally uses MAPI / Windows Sockets functions to transmit the actual data to an email address.

V.3. Data Structure (mdlTAPIFunction.bas, mdlTAPISource.bas)

Memory for all data structures used by the API must be allocated by the application. The application passes a pointer to the API function that returns the information, and the function fills the data structure with the requested information.

The defined data structures contain fields that indicate the total size needed to fill in all variable data (dwNeededSize) along with the total size used by TAPI when filling in the structure (dwUsedSize). For example:

Type VARSTRING
  dwTotalSize As Long 'size 4 byte
  dwNeededSize As Long 'size 4 byte
  dwUsedSize As Long 'size 4 byte
  dwStringFormat As Long 'size 4 byte
  dwStringSize As Long 'size 4 byte
  dwStringOffset As Long 'size 4 byte
End Type

Global Const VARSTRING_FIXEDSIZE = 24 'total size needed by this data structures (4 x 6)

The dwTotalSize field is first set by the calling application to indicate the total number of bytes allocated for TAPI to return the contents of the structure. If TAPI cannot fill in all values without running out of allocated space, an error is returned.

Some TAPI functions in mdlTAPIFunction.bas module use particular data structures for example:
lineOpen( ) in calllineOpen( ) needs LINECALLPARAMS data structure.
lineMakeCall( ) in calllineMakeCall( ) needs LINECALLPARAMS data structure.
V.4. Initialize TAPI (calllineInitialize( ), callphoneInotialize( ) - mdlTAPIFunction.bas)

Before placing or receiving a call, an application must establish communication link between itself and TAPI. The application must select a telephony event notification mechanism (lineCallBack and phoneCallBack), and specify this in a call to the lineInitialize / phoneInitialize function. One of the values lineInitialize / phoneInitialize returns is the number of line / phone devices available (lpNumLines / lpNumPhones) to the application. In this example, that number is one (if only one modem installed), and the line’s ID in telephony’s zero-based scheme is 0. The application must establish this communication link (with lineInitialize) regardless of the type of call to be placed or received.

V.5. Monitor Messages from TAPI (lineCallBack( ) - mdlTAPILine.bas, phoneCallBack( ) - mdlTAPIPhone.bas)

The LineCallBack( ) (mdlTAPILine.bas) and phoneCallBack( ) (mdlTAPIPhone.bas) functions are a callback scheme through which applications are notified of the success or failure of function calls and other events. TAPI will continue to receive status information and route that to the application through the callback function previously registered in lineInitialize( ). The quality of the status information (dialing, ringing, busy, idle, and so on) is all determined by the hardware vendor and TAPI service provider application. The more sophisticated the hardware, the more accurate the progress information.

For example, when the call session is finished, the application receives a LINE_CALLSTATE message, which informs it that the state of a line device has changed. In this example, a remote disconnect has occurred. The application disconnects the call at the local end with lineDrop (on hook).

V.6. Open Line / Phone Devices (calllineOpen( ), callphoneOpen( ) - mdlTAPIFunction.bas)

The function lineOpen( ) and phoneOpen( ) will activate the specified line or phone device and give the application line (lphLine) / phone (lphPhone) handle that be used for the following operations on the corresponding line / phone device.

Once a line / phone device opened, the application is enable to place an outgoing call or receive an incoming call.
V.7. Receive Calls (calllineAnswer( ) - mdITAPIFunction.bas)

After an application has opened a line device (calllineOpen( )) registered a privilege LINECALLPRIVILEGE_OWNER (allows the application to manipulate call), and a media mode (LINEMEDIAMODE_AUTOMATEDVOICE the call and the voice is locally handled by an automated application), the application is notified when a call arrives on that line.

Specifically, an application that has opened the line with LINECALLPRIVILEGE_OWNER receives a LINE_CALLSTATE message only if it has become an owner of the call. In this notification, TAPI gives the application a handle to the incoming call (lineCallback( ) mdITAPILine.bas), and the application keeps this handle (lhActiveCall) until the application deallocates the call.

This handle will be used by lineAnswer (calllineAnswer( )) to receive an incoming call when the application detects a message (LINEDEVSTATE_RINGING lineCallBack( ) - mdITAPILine.bas) sent by TAPI.

There are two conditions in this state. If "Auto Answer" is enabled the application will automatically activate calllineAnswer( ), play back an announcement, and start recording an incoming message. If not, the user can manually press "Answer" button to receive an incoming call.

V.8. Monitor Digit DTMF (calllineMonitorDigits( ) - mdITAPIFunction.bas)

The application will enable lineMonitorDigits when "Auto Answer" option is checked. Detected DTMF digits cause the application to be notified with the LINE_MONITORDIGITS message (lineCallBack( ) mdITAPILine.bas). This allows caller to press digit # 1 to hang up / disconnect the call, press digit # 2 to play back recorded message and press digit # 3 to send an e-mail. Digit monitoring on a call ends as soon as the call disconnects or goes idle.

The application disables this monitoring digits if the user makes an outgoing call or answers an incoming call.

V.9. Place the Call (calllineMakeCall() - mdITAPIFunction.bas)

Once the application has opened the line / phone device, it places the call with lineMakeCall, specifying the address in the lpszDestAddress parameter and the media mode (LINEMEDIAMODE_AUTOMATEDVOICE) in the lpCallParams parameter. This function returns a positive value if the function will be completed successfully, or a negative error number if an error has occurred. Negative return values describe specific error states. LINEERR_CALLUNAVAIL, for example, means that the line is probably in use (someone else already has an active call). If dialing completes successfully, messages are sent to the application to inform it about the call's progress. One of the message is LINECALLSTATE_CONNECTED.

At the same time, this application also calls callphoneSetHootSwitch( ) to set the hook switch mode (off hook mode).
V.10. Set Up the Phone Hook Switch Mode (callphoneSetHookSwitch() - mdlTAPIFunction.bas)

The user can manipulate the phone hook switch mode to activate the speaker or microphone or both while handling an incoming or outgoing call. The application will set device’s speaker on (mute the microphone PHONEHOOKSWITCHMODE_SPEAKER) when "Auto Answer" is enabled. And it sets both devices active (PHONEHOOKSWITCHMODE_MICSPEAKER ) if the user places an outgoing call or manually answers an incoming call (pressing "Answer" button).

V.11. Toll Saver (lineCallBack() - mdlTAPILine.bas)

The application invokes lineGetNumRings function to determine the number of times an inbound call on the given address should ring before the call is to be answered. Waiting a certain number of rings (2 or 4 rings) allows callers to be spared the charge of a call connection. Application also use the lineSetNumRings function in combination to provide a toll saver mechanism.

The application that receives a LINE_LINEDEVSTATE ringing message should wait a number of rings equal to the number returned by lineGetNumRings before answering the call. The function lineGetNumRings returns the minimum number of rings any application has specified with the function lineSetNumRings.

V.12. Drop the Call (calllinedrop() - mdlTAPIFunction.bas)

To terminate a call, the application uses lineDrop on the call. This has the effect of hanging up on (disconnecting) the call, which makes it possible to make another call on the line. If the remote party disconnects a call, the local application receives a LINE_CALLSTATE message via lineCallBack function (mdlTAPILine.bas) with a call state of disconnected (LINECALLSTATE_DISCONNECTED). If the local application disconnects a call, the call becomes idle (LINECALLSTATE_IDLE), but its handle (lhActiveCall) is not automatically deallocated (the application must call lineDeallocateCall).

V.13. Deallocate Call Handles (calllineDeallocCall() - mdlTAPIFunction.bas)

A call handle (lhActiveCall) remains valid after the call has been dropped. Once an application has received a LINE_CALLSTATE (LINECALLSTATE_IDLE) message, it should call lineDeallocateCall to free system-allocated memory related to the call. If an application is the owner of the call, it can deallocate the call’s handle only if the call is in the idle state. Failure to deallocate call handles in a timely way can result in system failure and lost calls due to unnecessary consumption of memory and other resources.
V.14. Close Line / Phone Device (calllineClose(), callphoneClose() - mdlTAPIFunction.bas)

The application must close all lines / phones it has open before it becomes inactive. After an application is finished using a line / phone device, it must close the device by calling lineClose() / phoneClose() on the line / phone -device handle. After the line / phone has been closed, the application's handle for the line / phone device is no longer valid.
VI. Multimedia Control Interface (MCI)

As mentioned in project overview, 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 Telephony-API line and phone device classes 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 Win32 environments to access or manage these media streams. One of these APIs includes the MCI (Media Control Interface) that provides a high-level generalized interface for controlling media devices.

For example, for line devices, an application can use TAPI to answer a connection from a caller. Once the connection is established, the application can then use the MCI Waveaudio API on the associated device to play back (send) and record (receive - and save) audio data over the connection.

There are two forms of MCI command sets:
1. command strings (communicates using English like text, such as : mmPlayMsgGree.Command = "close",
   mmPlayMsgGree.Command = "open",
   etc (see at "PlayBack Msg" button in frmTAPI.frm).
2. command messages (communicates using messages ID constants via mciSendCommand() functions and also additional parameters (data structures), such as : mciSendCommand frmTAPI.mmRecInMsg.DeviceID, MCI_SAVE, MCI_SAVE_FILE, mcisaveparams ; see at callMCISaveMsg( ) in mdlTAPIFunction.bas)

VI.1. MCI Command Messages

An application will use mciSendCommand( ) functions and MCI command to send messages that set up and control multimedia devices (such as waveform audio, a digital sampling of an analog sound source that can be stored in a file and play back). Waveform audio provides a method of recording and playing back digitally sampled sound.

This application uses two MCI controls (mmPlayMsgGree and mmRecInMsg) to play back a message and the other one to record an incoming message. The basic strategy to record and playback waveform audio is:
1. Locate the waveform audio device.
   The user must choose a suitable waveform audio recording / play back device in the "Choose Wave Device" combo box to be used to play back and record waveform audio over to / from the phone connection.
   The user must select modem wave device ("Voice Modem ..." not local sound card ("Sound Card") in that combo box, otherwise the WAV file will be played back over the local sound card and the application won't record an incoming message.
2. Open the waveform audio device.
   The application will open that proper waveform audio device using command string
   (mmPlayMsgGree.Command = "open" or frmTAPI.mmRecInMsg.Command = "open")
3. Call mciSendCommand( ) function with one of the parameters below:
   - MCI_SET to set recording audio format (PCM, 16-bit, 8 kHz, mono).
   - MCI_RECORD to record a message based on recording audio format.
   - MCI_STOP and MCI_SAVE to stop recording process and then to save the
     recorded message in a WAV file format.
   - MCI_PLAY to play back the recorded message over the telephone line.
4. Close an open wave device, after a particular function finishes the particular task
   (recording and play back). So the wave device can be used for another operation.
   Besides above, the mmPlayMsgGree - MCI control allows user to record a new greeting
   (press record button) and overwrite the old one ("greeting.wav"). It also provides user to access
   all new or old incoming messages stored in the application's folder.

VI.2. WAV Audio File Format

   To play back and record a message over the telephone line, this application using analog-
   digital conversion process (Pulse Code Modulation) using sampling rates 8 kHz with digital
   resolution 16-bit for one channel (mono).
   Audio recording based on this format would require 16 kB per second. Thus for 10
   seconds wave file, the size would be 160 KB and 35 seconds wave file, the size should be 560
   kB. The higher sample rates the more disk space is required to store the digital audio.

VI.3. Detecting a Proper Recording and Playback Modem Wave Device
   (callLineGetID( ) - mdlTAPIFunction.bas)

   To playback (a greeting or a recorded message) and record (a caller’s message) to or from
   the telephone line, the application will automatically detect the proper modem wave device by
   invoking lineGetID function (the application’s default). This function will result a related device
   ID being used by the selected modem. The application will use this ID to control modem wave
   device instead of using an available sound card.
   When the user selects a sound card as a wave device (by unchecking the default in the
   check box), the greeting or recorded messages will be played in the local sound card not over the
   telephone line.

VI.4. Recording a Message (callMCIRecMsg( ) - mdlTAPIFunction.bas)

   To limit the length of each recording so that one voice file doesn’t inadvertently take up
   too much disk space, the application uses Timer control (Timer1) to specify the record time. The
user can select the recording time in "Mail". The maximum length of the voice file is 35 second and the default is 10 second.

Once the user specifies "Auto Answer" option is active and right after the application plays back a greeting to the caller, it will call callMCIRecMsg( ) to start recording a message for 10 or 35 second.

This function will close an open wave device (MCI control) first and then open it for subsequent recording operations. It will invoke a command message - MCI_SET to set recording audio format and also what voice modem (from the "Wave Device" combo box) will be used (parms.wInput = frmTAPI.cmbWaveRecord.ListIndex). Finally followed by the other command message - MCI_RECORD to record the incoming message.

At the same time, this function also activates the timer control (frmTAPI.Timer1.Enabled = True) to keep track the length of the message.

VI.5. Saving a Message (callMCISaveMsg( ) - mdITAPIFunction.bas)

Once the Timer1 has elapsed, the application will call callMCISaveMSG( ) to stop the recording operation (MCI_STOP) and store the recorded message in application's folder. Before saving the recorded audio to the application's folder, it call CreateFileName( ) (mcisaveparams.lpFileName = CreateFileName) to specify the name of the file in which the recorded data will be stored. This file name will be named based on the date and time it recorded ("New_ & date & time & a/p.wav - 4.10. Overwrite and Save the Greeting Message).

This function also be called, everytime the caller hang up (press digit # 1) the connection (see at lineCallBack( ) in mdITAPILine.bas).

VI.6. Playback an Announcement - Greeting (callGreeting( ) - mdITAPIFunction.bas)

The application calls callGreeting( ) when "Auto Answer" is chosen. It will do some tasks below:

- Close MCI device first and open it (frmTAPI.mmPlayMsgGree.Command = "open").
- Set data structure (Dim mciwavesetparms As MCI_WAVE_SET_PARMS) required before applying command message MCI_PLAY. One of the significant parameters has to be provided is voice modem device where the message will be play back to (mciwavesetparms.wOutput = frmTAPI.cmbWaveRecord.ListIndex).
- Play back greeting (after 2 or 4 rings) or recorded message (caller pressing digit # 2) to the caller.
- Close MCI control after playing back greeting or recorded message done.
VI.7. Detect and Display MCI Error Codes (mdITAPIMCI.bas)

An application has to determine the content of MCI_WAVE_SET_PARMS data structure before using mciGetErrorString() function to detect and display a message. The length of buffer to store the error message returned by lpszErrorText parameter is 300 characters. To extract the error message from series of "0" in the predetermined buffer, this function calls GetRidof0 (mdITAPIPhone.bas) to get rid of all 0s from that buffer.
VII. Remote Access Service Application Programming Interface (RAS API)

To allow this application to send the recorded message via an email, this application must establish a connection to the remote ISP's server first before invoking a message transport provider such as Messaging Application Programming Interface MAPI (Netscape Messenger) or Windows Sockets to send it out.

In Windows 95 environment, this can be done by use of Remote Access Service (RAS API) that lets user to make a connection to the remote server without launching "Dial-Up Networking" dialog box up. This service also provide the way to enumerate all active connections first before terminating them.

The main RAS functions used in this application are (mdlTAPIRas.bas):
1. RasEnumEntries( ) to enumerate and display the phone book entry available by which the RasDial function will establish a connection or.
2. RasDial( ) to establish the connection to the remote server.
3. RasEnumConnections( ) to enumerate and display the active connection before terminating it.
4. RasHangUp( ) to end the active connection.

VII.1. Starting a Connection (callRasDial( ) - mdlRAS_API.bas)

Fifteen (15) seconds (tmrcallDirectDialUp) after the caller terminates a call, this application will call RasEnumEntries function (callRasEnumEntries( )). The application must wait 15 seconds or more to let the system to properly close the previously opened port. If the application performs RasDial( ) function less than 15 seconds the application will fail to fire up the dial up connection.

This function will collect a user predetermed modem information (from frmLogin.ComboBox1.AddItem strEntryName) and some connection parameters to log onto the remote server, i.e. user name, password (both are members of RASDIALPARAMS) and remote server phone number (see at frmLogin.frm). The RASDIALPARAMS.szPhoneNumber, a member of RASDIALPARAMS, must contain the server’s phone number to be called otherwise the connection will fail.

A RAS client application then uses the RasDial function (callrasDial( )) to start the connection operation (carried out by the Remote Access Connection Manager). The Remote Access Connection Manager is a service that handles the details of establishing the connection to the remote server by sending the connection parameters above. These connection parameters will be authenticated by remote server. Once the dial up connection is established, the application can call MAPI or Winsocks to start sending an email.

VII.2. Terminating a Connection (callRasHangUp( ) - mdlRAS_API.bas)

After MAPI or Winsocks sends an email to the remote server, this application needs to end the active connection. First, this application will call RasEnumConnections function to
enumerate if there is more than one active connection established. After this function returns a RASCONN structure containing the hRasConn connection handle of the each active connection, the application can call RasDialHangUp function to end the all available connections.

By releasing the connection, the application is ready to receive an incoming call or to place an outgoing call.

VII.3. Detecting RAS Error Messages

Another significant RAS function is RasGetErrorString function that can get a display string of the error. This will be helpful to determine the size of the structure in byte and also to detect if the user name or password of the logged user is invalid. For example, if the determined size is too small this function will return "The buffer is invalid" message. By detecting this error the user can determine the exact buffer size of the required data structure.

If the user enters inappropriate user name, password or phone number that causes the RasDial function fail to establish a connection, an error trapping function (see at callRasDial( )) will handle this problem by automatically terminating the connection was previously established so that the modem port will be open and ready to fire up a new connection again.

VII.4. Timer

There are five VB’s Timer controls will be used in this project to perform some tasks in response to the TAPI, MCI, RAS API, Winsocks and MAPI calls.

1. timer2 and timer3 (frmTAPI.frm) will let the MAPI server (Netscape Messenger) send a WAV file to the remote ISP’s server for 80 seconds. The two timers will be active whenever the user chooses Netscape Messenger as a mailer to transport an email and once the application had the dial up connection using RAS API calls (in mdlRAS_API.bas). After successfully sending the e-mail, the two timers will be disabled again.

2. timer1 is used to set the length of recordable message for 10 or 35 seconds (see at frmTAPI.frm).

3. tmrCcallDirectDialUp (in frmTAPI.frm) will let the application release an open modem’s port so the RAS API functions can use this port to establish a dial up connection. 15 seconds after caller pressing digit # 3 to send a WAV file, the RAS API will try getting a connection to a remote server by gathering the username and password supplied by the user in the "Login" dialog pop-up window (frmLogin.frm).

4. If the application can not detect a disconnecting signal from the local exchange when a caller directly hangs up the call (rather than pressing digit # 1), it will be depending on the tmrHangUp (in frmTAPI.frm) to end the call session by calling lineDrop( ) and lineDeallocateCall( ) function (in mdlTAPIFunction). After 45 seconds, the two function will close and dealloclate the handle of the call (lhActive constant) so the application is ready to do the next call operations.
VIII. Mailer

This project implements two methods to send an email with .WAV file (binary formatted file) as an attachment.
IV.1. Windows Sockets (Winsocks Control).

VIII.1. Windows Sockets (Winsocks) Control

Rather than sending the an email with large data (.WAV file) over one or more intermediary servers, a better solution is using Windows Sockets to transmit directly to the remote user. This means that all transmissions and communications will be in real time.

A WinSocks control allows an application to connect to a remote machine and exchange data using Transmission Control Protocol (TCP). This particular protocol will allow the remote server to return a message when data has been sent, received or error detected (Winsock1_DataArrival( ) and Winsock1_Error( ) - in frmLogin.frm). It also supports sending a large data (audio) file and ensures that the all of the data can be sent successfully once a connections to remote server established.

This project implements this control as a client application that requires an explicit connection first and collects user information (such as server name, local and remote port number, destination / originating email address where the data is going to be sent to) before sending or receiving data to / from an ISP’s remote server.

But the problem using Winsocks is that internet e-mail was designed for plain text messages. As such, many systems expect the messages to only contain printable characters from the 7-bit (first bit of the 8-bit byte is always zero) ASCII character set. These programs can have problems if the message includes extended 8-bit (the first bit is a one) characters, such as the various accented letters. This also poses a problem for sending files, such as images, sound, video, spreadsheets and programs which can contain any combination of 8-bit binary data.

The way around this limitation is to encode the binary data (attachment) into ASCII characters before sending. To the mail system that the messages travels through, the file is just so much text. At the receiving end, the message is decoded back into the original file. There are many different standards. One of the ways to encode the binary data is using MIME base 64 encoding.

VIII.1.1. Multipurpose Internet Mail Extension (MIME) - overall structure of a message

MIME (Multipurpose Internet Mail Extensions) is not actually a method for encoding attachments. It deals with the overall structure of a message. A message using MIME doesn’t necessarily include attachments. If it does include attachments, they most often use base64 encoding.
The main advantage of MIME is that it provides a consistent way for the sending program to describe the message contents to the receiving program. This system lets users attach binary files to e-mail, such as images (.gif, .jpg, etc.), audio (.au, .wav, etc.), or video (.mov, .avi, etc.). The MIME is specified by the MIME headers and can be seen in the message source, for example: "MIME-Version: 1.0".

**VIII.1.2. Base64 Encoding**

Base64 is the preferred encoding method for attachments in messages using MIME. It is a way to encode 8-bit binary data as 7-bit (the 1st bit is zero) ASCII files. This encoding is designed to make binary data survive transport through transport layers that are not 8-bit clean, such as mail bodies.

Base64-encoded data contains uppercase, lowercase, numbers, '+', '/' and '='. It takes the encoded stuff in groups of 4 characters and turn each character into a code 0 to 63:

A means of labeling the content of mail messages:
- A-Z map to 0 to 25
- a-z map to 26 to 51
- 0-9 map to 52 to 61
- + maps to 62
- / maps to 63

**VIII.1.3. MIME - base64 encoding - mdlMIME.bas**

The process of MIME base64 encoding takes place in mdlMIME.bas. The following section will outline how the encoding is accomplished. First of all, open and read the attachment file as an ASCII code and write it to a certain parameter (IConvertFile). By using LOF (Length of File) function to determine the length the created file, read IConvertFile into three variable bin(). These variables are converted to the arbitrary binary data first by calling callEncodeBase64( ) function and it will return it into one or more lines of base64 encoded data (strings - varResultBase64). These results will be sent directly by Winsocks to the remote server.

For example:
Three characters ("AaA") are from each bin( ) variable that has been read from the attached file (txtAttachFileName). These characters are converted to binary digit as follows
- char "A" has character code 65 => 0100 0001
- char "a" has character code 97 => 0110 0001
- char "A" has character code 65 => 0100 0001

These then are combined and mapped to three real bytes formed as follows:
01000001 01100001 01000001

The next step is to convert those three bytes formed above to the 4 sets of 6 bits and extracted it into the 64 x 8 bits by calling callEncodeBase64( ) function.

Final results: 010000 010110 000101 000001 or "QWFB"

Those are ASCII texts that are ready for transmitting as an attachment via Windows Sockets.
VIII.2. Messaging Application Programming Interface (MAPI) as a MAPI client

MAPI compliant documents including an attachment file can be mailed via Netscape Messenger functioning as a MAPI server (message transport). Unlike Winsocks that has to be aware of binary attachment file and convert it to ASCII text, the Netscape Messenger will directly handle MIME base 64 encoding or some other binary transfer protocol. So if the user selects Netscape Messenger as a mailer (message transport), the application doesn't need to encode binary attachment into ASCII texts anymore in the mdlMAPI.bas module.

In addition to acting as a message transport (moving messages to and from remote servers and clients), Netscape Messenger has the responsibility of managing the storage and addressing of messages from any number of client applications (in this case, this project functions as a mail client application. The destination mail address must be available in the Netscape Address Book before the MAPI client can send an email to that email address. That will let the MAPI client invoke "resolvename" event (see at mdlMAPI.bas module).

When a mail client application sends a message, the MAPI32.dll file responds to the MAPI function call. The MAPI routes the messages to the message storage and address book Netscape Messenger (to resolve the mail address). If the application marks a particular message for sending, the MAPI mail server will check the message's address to determine the message transport to be used.

To gain access of MAPI server, Netscape Messenger need to be configured as follows:

For Netscape Messenger with version older than ver. 4.5
  . Select Preferences from the Edit/Preferences menu
  . Select Mail Servers
  . Click on the "More Options" button
  . Select the option "Use Netscape Messenger from MAPI base applications" and click OK.

For newer version of Netscape Messenger
  . Select Preferences from the Edit/Preferences menu
  . Select Mail & Newsgrups
  . Select the option "Use Netscape Messenger from MAPI base applications" and click OK
VIII.2.1. MAPI client - mdlMAPI.bas; MAPI server - Netscape Messenger

This part is responsible for composing (frmTAPI.MAPIMessages1.Compose) and sending a .wav audio file format as an attachment over the net (frmTAPI.MAPIMessages1.Send). The Nestcape Messenger as a one of MAPI compliant Mail Servers will transport it (frmTAPI.MAPISession1.SignOn log onto MAPI server).

The user must provide a destination email address to which an recorded message will be delivered and it must be available in MAPI mail server Address Book first.

After sending "New & date & time.wav" prefixed recorded message file to mail server, it automatically fires up timer (timer2 and timer3) for 80 secs to give an enough time for Nestcape Messenger to send it out to remote ISP’s server.

If an email address is invalid or not available in Address Book, error trapping (On Error GoTo TerminateMAPI) will automatically terminate the dial up connection and close the Nestcape Messenger window (MAPISession1.SignOff) so the application will be ready to receive the next incoming call.

The defaults below can be changed in "Mail" -> "Set Up" menu or in this module mdlMAPI.bas:

frmTAPI.MAPIMessages1.RecipAddress = frmLogin.txtEmailAddress.Text (pyd0023@rit.edu)
lookup up address book of MAPI mail server (Netscape, MExchange, etc.)
frmTAPI.MAPIMessages1.ResolveName
frmTAPI.MAPIMessages1.MsgSubject = “sending recorded message”
frmTAPI.MAPIMessages1.MsgNoteText = " WAV Audio File Format "
frmTAPI.MAPIMessages1.AttachmentName = mcisaveparams.lpFileName
frmTAPI.MAPIMessages1.AttachmentPathName = App.Path & "\" & mcisaveparams.lpFileName
IX. Conclusion

TAPI helps insulate the developers who are creating an application related to the telephony services from the complex underlying hardware and telephone network.

TAPI is Microsoft's technology for the integration of telephone equipment with the powerful capabilities of PCs and Windows. TAPI provides a standardization mechanism which allows a single application to work with different vendor telephony hardware. TAPI accomplishes this goal by using two layers. The first layer is the API layer which the applications are written to. The second, and lower layer, is the Telephony Service Provider Interface (TSPI) layer, sometimes referred to as the device-driver layer. It is responsible for translating the API layer requests into the hardware level requests necessary for each device supported.

The following issues are not carried out in the application yet, so there are a number of improvement can be made in the future:

Hardware independence:
Basically, Microsoft and Intel have developed TAPI to provide a standard interface across telephony platforms voice boards, fax / modems, PBXs, cellular phones, and other telephony devices. This means that any TAPI compliant application can be used with any telephony device or PBX with a TAPI driver. Since not all TAPI device support all TAPI features, the application should have included TAPI examiner function to analyze telephony device in use and then determine which features of TAPI device supports.

Telephone service:
The application should have contained a proper function to identify the types of telephone service used to transmit voice signals such as
- Analog POTS (Plain Old Telephone Service)-for general voice-grade transmissions and for data transmission up to 28.8kbps speed
- Digital line - for dedicated high-speed voice and/or data services (56kbps and above)
- ISDN (Integrated Services Digital Network) for high-speed multichannel simultaneous voice and data services

WAV file format:
To reduce the size of audio files the application should used the Windows low level multimedia wave API instead of the Multimedia Control Interface (MCI), which forces the Windows to use the appropriate codec to do conversion the wav format being used (PCM) into wave format compression supported by the modem. It also allows the application to support many formats of audio files that depend on TSP being used. For example, if the modem supports Adaptive Differential Pulse Code Modulation (ADPCM) compression (7200 samples / second, 4 bits / samples) on the voice path, the application will keep the recorded files much smaller.
Standard ADPCM compressed audio formats are now widely used by vocal mail and answering machines, as well as by the multi-media CD-ROM.

**Controlling the greeting messages:**

The application should implement MCI notification. When the caller press a certain digit while the greeting message is running, it will stop it and directly respond it without waiting until the greeting message end.

**Future Enhancements**

The combination between the TAPI – compliant application and some emerging technologies below will benefit for developers to create an application that can be used in business sectors such as telemarketing (mail orders), phone banking, etc. Those combinations will also help the user especially the disability people to access various telephony services based applications easily.

The application can handle more than one telephone line.

TAPI allows the application to access the telephone lines available simultaneously.

**Combine with text to speech process.**

The combination the TAPI and Text to Speech technology (construct phrases from ASCII text and speak them over a phone line) will let the developer create speaking email messages.

**Combine speech recognition process.**

By combining the TAPI and speech to text recognition process, developers can build an application that is able to recognize spoken letters, words, and numbers from pre-defined or custom vocabularies which would allow callers to control voice mail, spell names, request extensions, and order products, all without entering touch-tones.
Citations

7. “The TAPI FAQ Frequently asked questions about TAPI”,
   http://cbr.nc.us.mensa.org/homepages/schenck/tapifaq.txt
8. “Windows Telephony Overview”
   http://www.sellsbrothers.com/intro2tapi/media.htm
10. “My unofficial TAPI FAQ”
    http://members.tripod.com/~tapifaq/
Appendix

Definitions:

SMTP is a protocol or language used to send email to a server on the Internet. It has been in use for years and is one of the core Internet services.

Kill statement
Deletes files from a disk.

TAPI functions:

Idle state: This corresponds to the "null" state: No activity exists on the call, which means that no call is currently active.
Connected state: Information is being exchanged over the call.
Disconnected state: The call has been disconnected by the remote party

a. line device functions

The lineGetID function returns a device ID for the specified device class associated with the selected line, address, or call.
lineInitialize: Initializes the Telephony API line abstraction for use by the invoking application.
lineShutdown: Shuts down the application's use of the line Telephony API.
LineNegotiateAPIVersion: Allows an application to negotiate an API version Version
lineGetDevCaps: Returns the capabilities of a given line device.
lineSetStatusMessages: Specifies the status changes for which the application wants to be notified.
lineConfigDialog: Causes the provider of the specified line device to display a dialog box that allows the user to configure parameters related to the line device.
lineOpen: Opens a specified line device for providing subsequent monitoring and/or control of the line.
lineClose: Closes a specified opened line device
lineMakeCall: Makes an outbound call and returns a call handle for it.
lineAnswer: Answers an inbound call.
lineSetNumRings: Indicates the number of rings after which inbound calls are to be answered.
lineGetNumRings: Returns the minimum number of rings requested with lineSetNumRings.
lineDrop: Disconnects a call, or abandons a call attempt in progress.
LineGenerateDigits: The lineGenerateDigits function initiates the generation of the specified digits on the specified call as inband tones using the specified signaling mode.
lineDeallocateCall: Deallocates the specified call handle.
b. phone devices function

**phoneOpen**
The phoneOpen function opens the specified phone device. A phone device can be opened using either owner privilege or monitor privilege. An application that opens the phone with owner privilege can control the phone's lamps, display, ringer, and hookswitch or hookswitches. An application that opens the phone device with monitor privilege is notified only about events that occur at the phone, such as hookswitch changes or button presses.

**phoneClose**
The phoneClose function closes the specified open phone device.

**phonSetHookS**witch**
The phoneSetHookS**witch function sets the hook state of the specified open phone's hookswitch devices to the specified mode. Only the hookswitch state of the hookswitch devices listed is affected.

**phoneShutdown**
The phoneShutdown function shuts down the application's usage of TAPI's phone abstraction.

**phoneInitialize**
The phoneInitialize function is to initialize phone device.

**phoneGetDevCaps**
The phoneGetDevCaps function queries a specified phone device to determine its telephony capabilities.

**MCI functions :**

**MCI** (Media Control Interface) provides standard commands for playing multimedia devices and recording multimedia resource files. These commands are a generic interface to nearly every kind of multimedia device.

**MCI Commands**
The mciSendCommand function sends a command message to the specified MCI device.

**mciGetErrorString**
The mciGetErrorString function retrieves a string that describes the specified MCI error code.

**waveOutGetDevCaps**
The waveOutGetDevCaps function retrieves the capabilities of a given waveform-audio output device.

**waveOutGetNumDevs**
The waveOutGetNumDevs function retrieves the number of waveform-audio output devices present in the system.

**RAS API functions :**

**RasDial**
The RasDial function establishes a RAS connection between a RAS client and a RAS server. The connection data includes callback and user-authentication information.
RasHangUp
The RasHangUp function terminates a remote access connection. The connection is specified with a RAS connection handle. The function releases all RASAPI32.DLL resources associated with the handle.

RasEnumEntries
The RasEnumEntries function lists all entry names in a remote access phone book.

RasEnumConnections
The RasEnumConnections function lists all active RAS connections. It returns each connection’s handle and phone-book entry name.

RasGetErrorString
The RasGetErrorString function obtains an error message string for a specified RAS error value.

lstrcpy
The lstrcpy function copies a string to a buffer.
Incoming Call Flow Chart

An incoming call

- LineInitialize() / PhoneInitialize()
- LineOpen() / PhoneOpen()
- LineSetNumRings()
- LineSetHookSwitch()

If "Auto Answer" then
- Answer after 2 / 4 rings
else
- press "Answer" button, user and caller got conversation
end if

Menu:
- Press # 1 to Hang up call.
- Press # 2 to Play back Msg.
If "Enable Mail" then
- Press # 3 to send email
End if

If the caller doesn't end the call after 40 secs the record time elapsed, the application will automatically terminate the call

Play an Announcement (greeting.wav)

- Callgreeting()

Automatically Record caller message for 10 or 35 secs and Save it based on date and time it recorded

- CallMCIRecMsg()
- CallMCISaveMsg()

- Hangs up and ready for receiving an incoming call or placing an outgoing call

Send E-mail:
- Using mailer either
- Netscape Messenger or
- Windows Sockets
To e-mail address:
pyd0023@rit.edu

- RasDialUp()