5-1998

The FM Radio Program

Andrew Olsen
Utah State University

THE FM RADIO PROGRAM

by

Andrew Olsen

Thesis submitted in partial fulfillment
of the requirements for the degree

of

UNIVERSITY HONORS
WITH DEPARTMENT HONORS

in

Electrical and Computer Engineering

Approved:

_____	_____
Thesis/Project Advisor	Department Honors Advisor

_____
Director of Honors Program

UTAH STATE UNIVERSITY
Logan, UT

1998

Executive Summary

The PC FM Radio program allows a PC user to listen to FM radio stations while using a PC. The application includes controls that will let the PC user select different FM stations, preset a favorite FM station, select a preset station, and record the current station. The FM hardware is an ISA expansion card that provides the audio signal to the PC through the auxiliary input jack in the soundcard. The supporting software runs on Windows 95 machines. A help file is included for simple instructions. The software runs in the background and doesn't use much of the computer resources. The software graphical user interface (GUI) is user-friendly and looks like an FM radio interface. The project met all of the original design specifications and followed the principles and techniques of software engineering.

Table of Contents

List of Figures

1.0 Introduction

The personal computer (PC) has become more than an adding machine or a word processor. Multimedia personal computers are a trend in the current market. These multimedia computers have become a home entertainment system. There are many different features and functions that make a multimedia PC enjoyable.

The ability to listen to an FM station could be a nice feature of a multimedia PC. The digital point-and-click interface would be simple to use and could easily be minimized. A PC user could also record a favorite song or a DJ monologue.

The FM Radio Program has been designed and developed so that it allows a PC user to listen to any local FM station. The FM Radio Program works in the background. This means that listening to the FM radio uses few CPU cycles. The FM audio data is sent straight from the FM PC receiver card to the speakers through the soundcard. The FM receiver card is an 8-bit ISA expansion card and has an antenna external to the computer. The FM Radio Program is a graphical user interface (GUI) that controls the FM PC Receiver card. The GUI allows the user to select an FM station, jump right to a preset favorite FM station, scan for FM stations, or record from an FM station. The GUI is user friendly and looks like an FM tuner on a commercial stereo.

2.0 Functional Requirements

The FM Radio Program is a Graphical User Interface (GUI) that has the ability to control the FM receiver card. The GUI is intuitively easy to use and has the familiar look of an FM tuner. The FM Radio Program allows the user to change FM stations, adjust the speaker output volume, and preset favorite FM stations. An additional feature of the FM Radio Program is that the user can record the current incoming station and save this recording to a (.wav) file. The most important feature of this program is that it runs in the background and doesn't use the CPU to play the FM radio signal. The music signals will come from the FM receiver card through a stereo cord into the auxiliary input port on the soundcard. The soundcard will then play the audio signal directly to the speakers. This approach will bypass the need for the CPU to transfer the music data from the FM Receiver card to the soundcard.

This program was developed using Microsoft Visual C++ 5.0 for the Windows 95 operating system. Existing libraries and object linking and embedded (OLE) controls were used to enhance the user interface of the FM Radio Program. C++ was the language used.

3.0 Design

3.1 Design and Analysis

In order for a software project to be successful, each phase of the software application must be carefully planned. The design and analysis phase uses the Systems Analysis, Design, and Implementation (SADIE) method. The first phase in software engineering is to create a context diagram. A context diagram shows the boundaries of the system; in the other words, it puts the system "in context". The context diagram is shown in Figure 1.

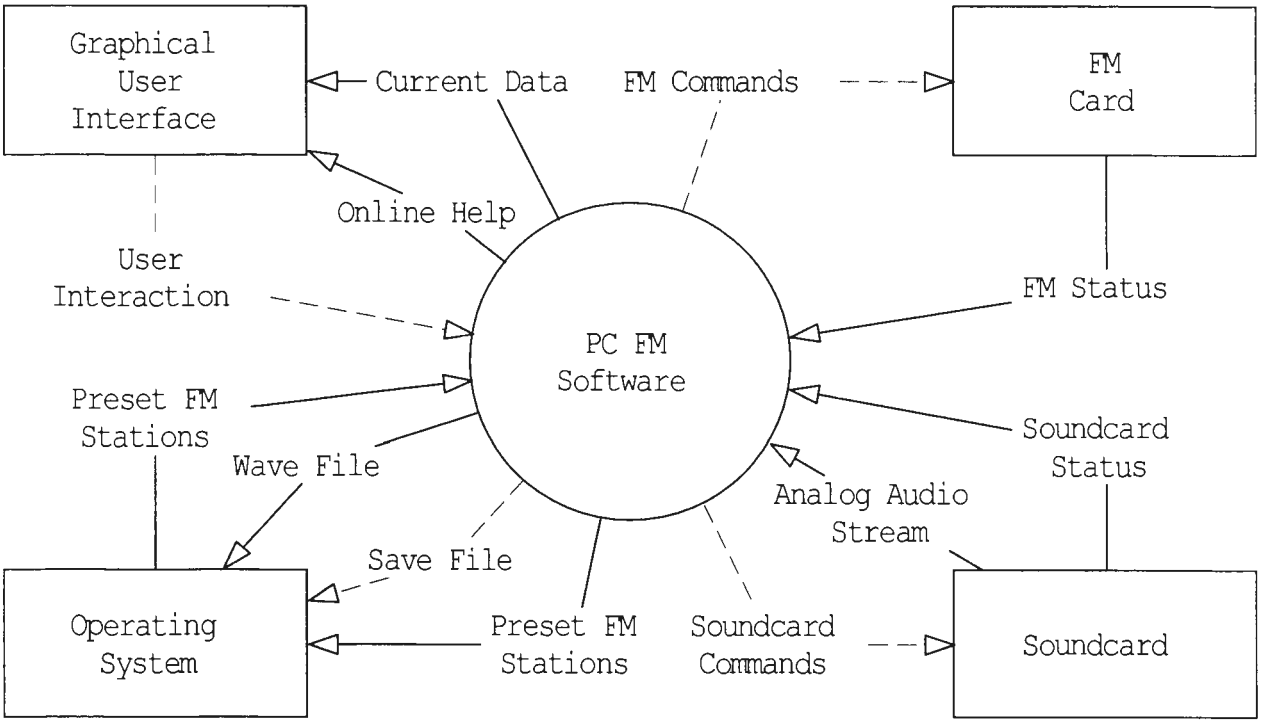

Figure 1: Context Diagram

The context diagram for this application defines the different devices and software with which the application will be interfacing. The graphical user interface is the monitor, keyboard, and mouse interactions. The operating system provides access to the hard drive for

saving different data. Interfacing to the soundcard, the application can control volume and record music. The interface to the FM card controls the FM radio hardware to adjust for different radio stations.

The next step in SADIE method is defining the user interface. This is discussed in further detail in section 3.2, Graphical User Interface.

The level one dataflow diagram is the next step in the SADIE method. The level one dataflow diagram shows the main structure of the application. A circle represents a

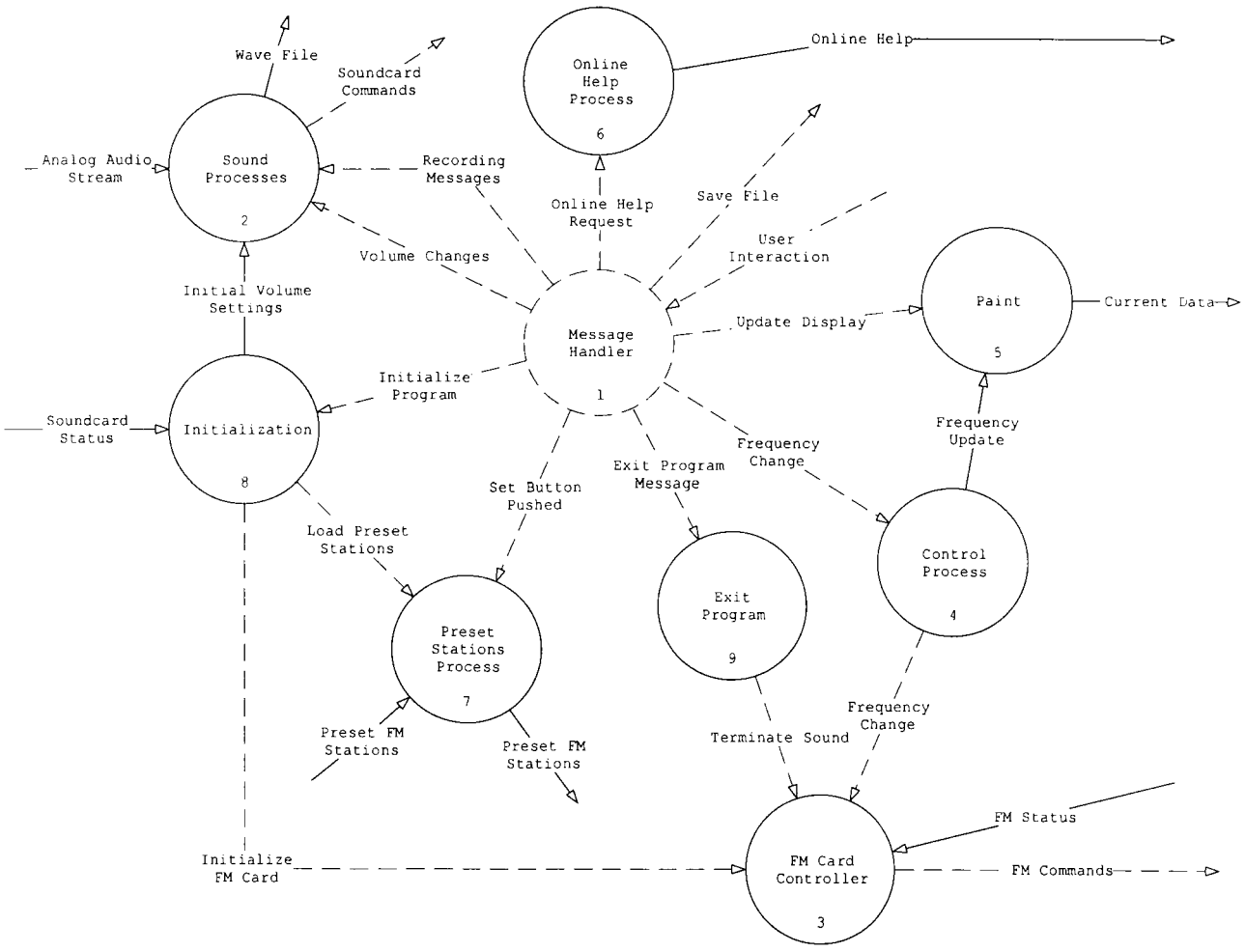

Figure 2: Level 1 Dataflow Diagram

process. Two horizontal lines represent a data store. Dashed lines represent program control. Solid lines represent the flow of data.

The Level 1 dataflow diagram shows the major components of the software. The flow of control and the flow of data are shown. The heart of any Windows program is the message handler. The message handler receives its messages from the operating system. The message handler is a large case statement that either calls certain functions, passes the message on to child windows, or ignores the message.

The initialization process sets up the program's data members and gets the current system information. The first part of the initialization is to verify that the computer system has a soundcard. Once the soundcard is detected, the abilities of the soundcard are tested. The tests include verifying that an auxiliary input jack exists, that the volume for the auxiliary jack can be modified, and that the soundcard mixer can be opened. If any of these tests fail, then a diagnostics message is printed to the user. This indicates that certain functions will not work on the current computer system.

3.2 Graphical User Interface

The graphical user interface (GUI) provides an easy interface for interacting with the applications capabilities and functions. By using a mouse, the user can easily manipulate the program. The GUI can be overdone and difficult to use. The FM Radio Program provides a fun and simple GUI.

The FM Radio Program was built from an MFC (Microsoft Foundation Classes) dialog class. The Visual C++ AppWizard created the main shell of the dialog GUI. The

AppWizard is a Visual C++ that will create a generic Windows-based application shell. The shell can then be shaped into most applications.

Based on the MFC dialog class, the FM Radio Program consisted of two dialog boxes; the first is the main program, and the second is a supporting dialog box. The main dialog box has multiple buttons, a slider control, and a display. Figure 3 is a screen capture of the main dialog box.

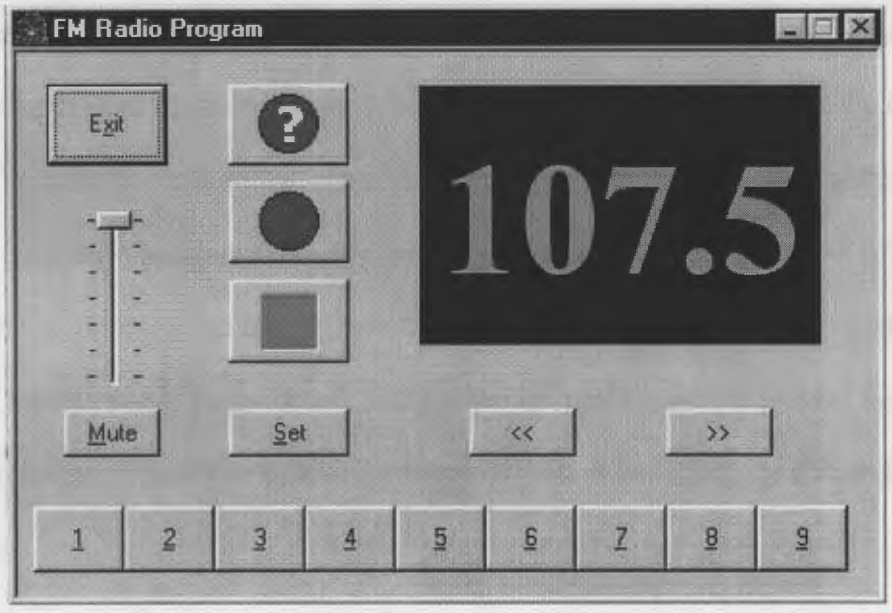

Figure 3: FM Radio Program Graphical User Interface

The Exit button terminates the program. The Mute button will mute the incoming radio music. The slider will adjust the volume level. The button with a "?" will initiate the help program. The button with a red circle will start recording the radio music. The button with a square will stop the recording. The buttons along the bottom are the Preset Station buttons. The black display with the green letters indicates the current FM radio station.

3.3 Sound Functions

To enhance the FM Radio Program's capabilities, several sound functionalities were added. These functionalities include controlling the volume level, muting the radio music, and adjusting the volume slider for outside changes.

To control the volume level for the auxiliary input jack, the Windows 95 soundcard driver must contain support for volume adjustments. If the driver supports volume adjustments, the soundcard device may contain many different auxiliary ports. The FM Radio Program discovers if the capabilities exist for volume control and then finds the auxiliary input port. Once this port is discovered, the program can adjust the volume.

To mute the radio music, the software uses the same routines as for the volume control. The mute function saves the current volume level. The soundcard driver is then told to mute the volume. If the user adjusts the volume while it is muted, the volume slider changes, but the music stays muted. Only when the mute button is pressed again is the volume restored. The volume level is restored to sound level that volume slider currently shows.

3.4 Preset FM Stations

An enhancement to the FM Radio Program is the ability to assign favorite stations to a button. By selecting a button, the application will immediately change the station assigned to that button. The user can then save favorite stations and not have to worry about finding a particular station again.

When the program is installed, each button has a default value of 89.5. To change the station assigned to a particular button, the user presses the Set button. The Set button creates a dialog box for changing the stations assigned to each button as seen in Figure 4.

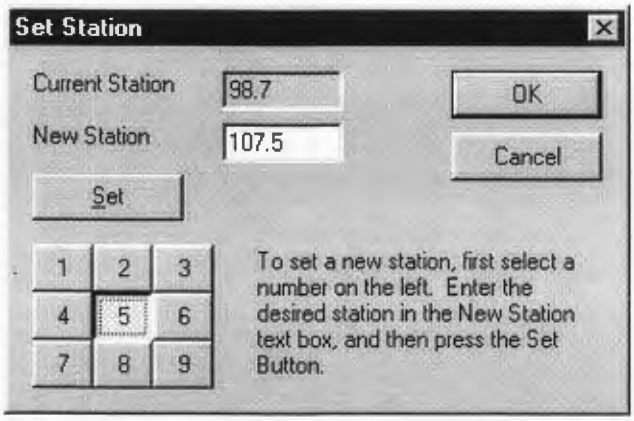

Figure 4: Set Station Dialog Box

When the Set Station dialog box first appears, the current station showing in the main program is in the New Station edit box. To set a new station, the user would first press a number in the number column. The Set Station dialog box shows in the Current Station edit box the current station assigned to that number. The user can modify the station to assign to that preset number by entering the desired station in the New Station edit box. The user then must push the Set button to save that change. The Current Station edit box will reflect that new change. If the user tries to enter in an invalid number, the program will tell the user to use a frequency between 87.9 and 107.9. Once the user has modified the preset stations, the OK button must be pressed. If the Cancel button is pressed, all changes will be lost.

3.5 Recording

When a favorite song is heard on the radio, the user may want to record that song for later listening. By pressing the record button, the software will start recording the incoming

data. This data is written to a temporary file. When the user is done recording, the Stop button is pressed. The application will prompt the user for a file name for the recorded (.wav) file. If the user enters a name that already exists, the program will ask if the user really wants to overwrite this file. If the user decides not to save the file, the Cancel button is pressed. The data written to the hard drive is discarded.

The recording process was the most difficult feature to add to the program. The ability to record is handled by the WIN32 API functions. These functions interface directly to the soundcard. Once the recording is started, it is its own thread.

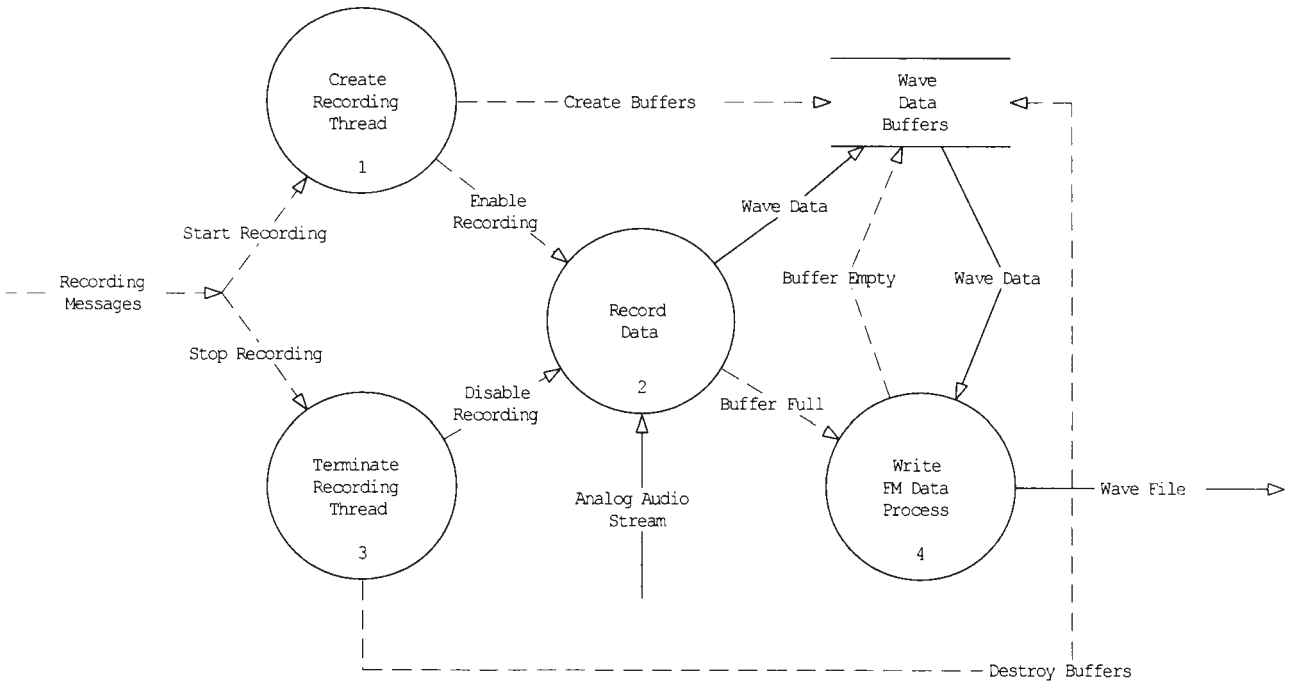

Figure 5: Recording Dataflow Diagram

Figure 5 shows the recording process. The recording process is initiated by the start thread message. The first process creates two buffers for storing the recorded data. The two buffers are added to the recording thread. Once that has been taken care of, the message is

given to enable the recording thread. When the recording thread has finished a buffer, it sends a message back to the main application indicating which buffer was filled. It then starts using the other buffer without any interruptions. The main application dumps the buffer into the temporary file, reinitializes the buffer and adds it back the recording thread. This process continues until the recording thread is disabled. Any remaining data in the buffer is placed into the temporary file.

3.6 User Interface

The ToolTips feature makes the FM Radio Program easier to use. The ToolTips feature will display a pale yellow text box by a button or a control when the user pauses the mouse over the button. The text within the pale yellow box indicates the function of that button or control.

The ToolTip functionality was added by inserting an MFC component into the application. An MFC Component is an object-oriented module that can be used with any Visual C++ software project. These "off-the-shelf" components allow the programmer to add certain functions to a program without having to code them.

Once the ToolTip component was added, each button and control within the FM Radio Program was assigned text. The text was carefully selected to help the user understand the function of every button and control in the program. The ToolTips feature gives a textual description to buttons that contain only a picture.

Another added feature was the ability to capture outside changes to the volume. If the Volume Control located as a speaker icon on the taskbar is opened and the volume adjusted, the volume changes would be seen with the volume slider in the FM Radio Program.

3.7 Help File

Users new to an application often find the program unfamiliar and awkward to use. Although carefully planning provides for an easy user interface, a new program can be intimidating. A help file provides written instructions that make it easier to understand the application and its different abilities.

The help file is viewed in its own application. A help file is a rich-text format (RTF) file with markers that indicate links to different pages and definitions. Help files are like HTML pages except that the help file is more difficult to create and must be compiled to be viewable by the help application.

The help file for the FM Radio Program included help pages for all of the major functions and a troubleshooting guide. The help file explains how to set a preset station or record a favorite song. The troubleshooting section in the help file explains difficulties with the recording process. One difficulty is that the soundcard may be able to record from only one device. The solution is that the device must be selected for the soundcard to record from it. Another problem is that if the recording volume is too high, the recorded song may sound distorted.

4.0 Test Results

4.1 Component Testing

As each component of the project was built, it was thoroughly tested to make certain that it performed as expected. The first step in the testing each module was stepping through each line of code and verifying that the code was producing the correct results. This works well for algorithm testing. To verify how the component worked as a function of time, the program would run without any breakpoints or debugging interrupts. This verified that the application would work under normal conditions.

This meticulous method of testing proved to be very successful. Once code was added to the project and thoroughly tested, it was accepted as being bug-free code. Any new bugs encountered later on were usually caused by newly added code. This approach to programming and testing allowed the project to move along smoothly. The program had many minor software bugs but only a few major software bugs.

The major software bugs were with the recording process. Recording is an independent thread. This means that the recording thread runs in its own time slices and is not as responsive as a process-oriented task. Communications between the main program and the recording thread needed to be carefully maintained and given a high priority. This was a major difficulty.

All of the Windows 95 GUI components are off-the-shelf Microsoft components. Other necessary functionality was added through the MFC components. There was no need to test these components. The assumption was made that these components had been carefully

tested by Microsoft and would perform according to their specifications. The project never encountered any bugs from the MFC components.

After each component was built and fully tested, the components could be put together and tested with each other. All conflicts between the major components were resolved, and the application was built.

4.2 System Testing

Once all of the major code for the program had been written, the entire application needed to be tested. This testing involves experimenting with different methods that might make the application crash or fail to perform to its capabilities. Sometimes software bugs found in this situation can cause major changes to the code and revisions to the structure of the application. System testing can also identify bugs that weren't previously noted.

The major bug that showed up on system testing was with the recording process. Code that was once working operated smoothly sometimes and failed miserably other times. This required a huge restructuring of the code and modifications on handling the recording process. Once these changes were in place, the recording process became robust and fully functional.

The FM Radio Program was tested on different computers and different operating system platforms. The application worked perfectly on two computers different from the development computer. These three computers were running Windows 95. The program was also tested on a Windows NT 4.0 computer. The program didn't work as well on the Windows NT computer. Most of the problems were with the soundcard device driver and its major differences with the Windows 95 soundcard device drivers. It appears that the FM Radio Program would run on a variety of machines.

5.0 Conclusions

5.1 Software Analysis

The FM Radio Program works as originally designed and meets all of its design specifications. The software was designed carefully to work regardless of any unexpected requests made by the user. Exception handling and error testing was implemented to handle an absent soundcard, bad device drivers, and other run-time errors.

5.2 Commercial Application

The FM Radio Program has been designed so that it could be a commercial product. The software used a generic Windows 95 approach. This means that the software lets the operating system determine which drivers and devices exist and how to interface. This means that the same function calls are used to access a soundcard from company X as wells as a soundcard from company Y. The underlying assumption is that the driver that interfaces between the operating system and the device is accurate and provides the correct information needed.

5.3 Recommendations

A recommendation is to incorporate an FM Card that this software can control and receive audio signals from. The software was validated by using a portable CD player that fed the audio into the soundcard. All of the sound and recording functions were tested this way. The program was structured in such a way that adding the assembly code for the FM Card communications would have been trivial. Having the FM Card built and working would have made this project complete.

6.0 Acknowledgments

Only one name appears on the title page of this report. Missing are the names of all those who gave me support and help in finishing this project and in choosing a career.

Many thanks go to my wife, Amy. She has supported me in all of my schooling endeavors. Although my time here at school took longer that it should have, she has supported me without question. She has tolerated my long hours away from home and my consuming interest in school. Without her support, I wouldn't be where I am today.

I thank Dr. Doran Baker for taking me in under his wing. I was uncertain to about my career choices. His influence helped me to see the fun and excitement of engineering.

I also thank Dr. Paul Wheeler and Dr. Todd Moon for their additional help to make the decision to study Computer Engineering.

I thank Dr. Ben Abbott for teaching me how to work hard and be ambitious. He has shown me what it is like to work in industry, to have real time and money constraints, to work with customers, and to work with a team on large projects. He has also shown me my potential and given me confidence in my work.

I also thank all of the Electrical and Computer Engineering staff and faculty. They have made possible the best education that I could ever get from any university. I thank Utah State University and the College of Engineering for their investments in me through scholarships and grants.

7.0 Vita

Profile

- Eager to participate in the growth of a progressive computer engineering organization.
- Skilled in designing and implementing Windows CE software for embedded systems.
- Experienced Windows 95 software programmer.

Education

Bachelor of Science in Computer Engineering, Utah State University, Logan, Utah (expected June 1998). GPA = 3.96.
Minor in German, Utah State University, Logan, Utah.
Minor in Computer Science, Utah State University, Logan, Utah.

Academic Achievements

- The Institute of Electrical and Electronics Engineer Secretary (IEEE), 1997-98.
- Member of Tau Beta Pi, National Engineering Honor Society, 1996-1998.
- Member of Honors Program, Utah State University, 1991-1998.
- Phi Kappa Phi, Utah State University, 1998.

Course Projects

- Developed Windows 95 software for a PC FM Radio Receiver.
- Designed and synthesized a robot path-finding chip in VHDL.
- Designed and built an 8088 microcomputer.
- Developed and implemented a video baseball game on a FPGA chip.

Computer Skills

Visual C++ for Win 95 & CE	VHDL	Matlab
Window NT	C and C++ Language	Mathematica
UNIX	Assembly Language	PSPICE for Windows

Experience

Computer Engineer. Center for Intelligent Systems, Logan, Utah, September 1997-present.

- Developing a Windows CE-based application for a mapping display for an Army vehicle. The software uses MFC threads and event handling.
- Developed a video camera driver and display for a Windows 95 LEGO project.

Software Test Engineer. IBM, Boulder, Colorado, June 1997-September 1997.

- Tested internal mainframe software for budget analysis.
- Created testing strategies to fully test the functionality of software.
- Tested labor submission software developed for Lotus Notes.

Independent Software Tester. Space Dynamics Lab., Logan, Utah, May 1996-June 1997.

- Independently tested software used for processing data from the SPIRIT III satellite.
- Created and implemented programs used for testing the functionality of software.
- Verified and reported bugs in software.

8.0 References

[1] Michael J. Pont, *Software Engineering with C++ and CASE Tools*, Addison-Wesley, 1996.

[2] David J. Kruglinski, *Inside Visual C++*, Microsoft Press, 1997.

[3] Andrew Evans, *Visual C++ 4 Masterclass*, Wrox Press, 1996.

[4] Richard C. Leinecker, *Microsoft Visual C++ 5 Power Toolkit*, Ventana, 1997.

[5] Ori Gurewich and Nathan Gurewich, *Teach Yourself Visual C++ 4 in 21 Days*, Sams Publishing, 1996.

[6] Kate Gregory, *Using Visual C++ 5*, Que Corp., 1997.

[7] Viktor Toth, *Visual C++ 5 Unleashed*, Sams Publishing, 1997.

[8] *MSDN CDROM On-line Library*, Microsoft Corp., 1998.

9.0 Appendix

9.1 FM Radio ProgramDlg.cpp

```
// FM Radio ProgramDlg.cpp : implementation file
//

#include "stdafx.h"
#include "FM Radio Program.h"
#include "FM Radio ProgramDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
public:
   CAboutDlg();

// Dialog Data
   //{{AFX_DATA(CAboutDlg)
   enum { IDD = IDD_ABOUTBOX };
   //}}AFX_DATA

   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CAboutDlg)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
   //}}AFX_VIRTUAL

// Implementation
protected:
   //{{AFX_MSG(CAboutDlg)
   //}}AFX_MSG
   DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
   //}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
```

```
    //{{AFX_DATA_MAP(CAboutDlg)
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
    //{{AFX_MSG_MAP(CAboutDlg)
        // No message handlers
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//
// CFMRadioProgramDlg dialog

CFMRadioProgramDlg::CFMRadioProgramDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CFMRadioProgramDlg::IDD, pParent)
{
    //{{AFX_DATA_INIT(CFMRadioProgramDlg)
    m_bMuteCheck = FALSE;
    //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CFMRadioProgramDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CFMRadioProgramDlg)
    DDX_Check(pDX, IDC_MUTE_CHECK, m_bMuteCheck);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CFMRadioProgramDlg, CDialog)
    //{{AFX_MSG_MAP(CFMRadioProgramDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_DESTROY()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_WM_VSCROLL()
    ON_BN_CLICKED(IDC_EXIT_BUTTON, OnExitButton)
    ON_BN_CLICKED(IDC_RECORD_BUTTON, OnRecordButton)
    ON_BN_CLICKED(IDC_STOP_BUTTON, OnStopButton)
    ON_BN_CLICKED(IDC_UP_BUTTON, OnUpButton)
    ON_BN_CLICKED(IDC_DOWN_BUTTON, OnDownButton)
    ON_BN_CLICKED(IDC_MUTE_CHECK, OnMuteCheck)
    ON_BN_CLICKED(IDC_SET_BUTTON, OnSetButton)
    ON_BN_CLICKED(IDC_STATION1_BUTTON, OnStation1Button)
    ON_BN_CLICKED(IDC_STATION2_BUTTON, OnStation2Button)
    ON_BN_CLICKED(IDC_STATION3_BUTTON, OnStation3Button)
    ON_BN_CLICKED(IDC_STATION4_BUTTON, OnStation4Button)
    ON_BN_CLICKED(IDC_STATION5_BUTTON, OnStation5Button)
    ON_BN_CLICKED(IDC_STATION6_BUTTON, OnStation6Button)
    ON_BN_CLICKED(IDC_STATION7_BUTTON, OnStation7Button)
    ON_BN_CLICKED(IDC_STATION8_BUTTON, OnStation8Button)
    ON_BN_CLICKED(IDC_STATION9_BUTTON, OnStation9Button)
    //}}AFX_MSG_MAP
```

```
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//
// CFMRadioProgramDlg message handlers

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

   // Add "About..." menu item to system menu.

   // IDM_ABOUTBOX must be in the system command range.
   ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
   ASSERT(IDM_ABOUTBOX < 0xF000);

   CMenu* pSysMenu = GetSystemMenu(FALSE);
   if (pSysMenu != NULL)
   {
      CString strAboutMenu;
      strAboutMenu.LoadString(IDS_ABOUTBOX);
      if (!strAboutMenu.IsEmpty())
      {
         pSysMenu->AppendMenu(MF_SEPARATOR);
         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
      }
   }

   // Set the icon for this dialog.  The framework does this automatically
   //  when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);       // Set big icon
   SetIcon(m_hIcon, FALSE);      // Set small icon

   /************************************************************
   **
   ** Code to paint images on buttons
   **
   ************************************************************/

   // Record Button
   CButton *pButton = (CButton *) GetDlgItem(IDC_RECORD_BUTTON);
   int x = m_RecordBitmap.LoadBitmap(SimpleRecord);
   if(x == 0)
      MessageBox("Failed to load Record bitmap.");
   m_hRecordPic = (HBITMAP) m_RecordBitmap.GetSafeHandle();
   pButton->SetBitmap(m_hRecordPic);
   m_bRecording = FALSE;

   // Stop Button
   pButton = (CButton *) GetDlgItem(IDC_STOP_BUTTON);
   x = m_StopBitmap.LoadBitmap(SimpleStop);
   if(x == 0)
      MessageBox("Failed to load Stop bitmap.");
   m_hStopPic = (HBITMAP) m_StopBitmap.GetSafeHandle();
   pButton->SetBitmap(m_hStopPic);
   // Disable Stop Button
   pButton->EnableWindow(FALSE);
```

```
   // Help Button
   pButton = (CButton *) GetDlgItem(ID_HELP);
   x = m_HelpBitmap.LoadBitmap(SimpleHelp);
   if(x == 0)
      MessageBox("Failed to load Help bitmap.");
   m_hHelpPic = (HBITMAP) m_HelpBitmap.GetSafeHandle();
   pButton->SetBitmap(m_hHelpPic);

   // Load Recording Button
   x = m_RecordingBitmap.LoadBitmap(SimpleRecording);
   if(x == 0)
      MessageBox("Failed to load Recording bitmap.");
   m_hRecordingPic = (HBITMAP) m_RecordingBitmap.GetSafeHandle();

   /*****************************************************************
   **
   ** Code to get current auxiliary line in volume and set slider control.
   **
   *****************************************************************/
   MMRESULT aux;
   DWORD auxVolume;
   auxID = GetAuxLineInNum();
   if(auxID != -1)
   {
      aux = auxGetVolume(auxID, &auxVolume);
      if(aux == MMSYSERR_NOERROR)
      {
         m_Volume = (WORD)(0xFFFF - (auxVolume & 0xFFFF));
         CSliderCtrl *pVolume = (CSliderCtrl *) GetDlgItem(IDC_VOL_SLIDER);
         pVolume->SetRange(0, 65535, FALSE);
         pVolume->SetTicFreq(10923);
         pVolume->SetPos(m_Volume);
      }
      else if(aux == MMSYSERR_BADDEVICEID)
      {
         MessageBox("Unable to Control Volume");
         // Disable slider
      }
   }
   else
   {
      MessageBox("Unable to find an auxillary input jack!\nRadio will not
work.");
   }
   m_bMuteCheck = TRUE;

   /*****************************************************************
   **
   ** Code to set up a callback function for outside volume changes.
   **
   *****************************************************************/

   MMRESULT                        mmr;
   UINT                    cMixerDevs;
   HMIXER                     hmx;
   unsigned int               id;
```

```
   cMixerDevs = mixerGetNumDevs();

   if(cMixerDevs == 0)
   {
      MessageBox("Won't be able to intercept outside volume changes.");
   }

   // Need a function to make this handle multiple mixer devices.
   // Currently only handles one device

   mmr = mixerOpen(&hmx, 0, (DWORD)(UINT)GetSafeHwnd(), 0L,
CALLBACK_WINDOW);
    if (MMSYSERR_NOERROR != mmr)
    {
      MessageBox("Failed to open Mixer.");
   }

   mmr = mixerGetID((HMIXEROBJ)hmx, &id, MIXER_OBJECTF_HMIXER);
   if(MMSYSERR_NOERROR != mmr)
   {
      if(mmr & MMSYSERR_BADDEVICEID)
         TRACE("Bad MMSYSERR_BADDEVICEID");
      if(mmr & MMSYSERR_INVALFLAG)
         TRACE("Bad MMSYSERR_INVALFLAG");
      if(mmr & MMSYSERR_INVALHANDLE)
         TRACE("Bad MMSYSERR_INVALHANDLE");
      if(mmr & MMSYSERR_INVALPARAM)
         TRACE("Bad MMSYSERR_INVALPARAM");
      if(mmr & MMSYSERR_NODRIVER)
         TRACE("Bad MMSYSERR_NODRIVER");
   }

   /*****************************************************************
   **
   ** Code to load FM data.
   **
   *****************************************************************/

   m_RadioData.Open("fm.dat",
                CFile::modeReadWrite | CFile::shareDenyWrite,
                &m_FileError);
   if(m_FileError.m_cause == CFileException::none)
   {
      m_RadioData.Read(m_nStations, sizeof(m_nStations));
      m_RadioData.Read(&m_nLastStation, sizeof(m_nLastStation));
   }
   else if(m_FileError.m_cause == CFileException::fileNotFound)
   {
      MessageBox("Failed to find file fm.dat.\nCreating a new file.");
      for(int i = 0; i < 9; i++)
         m_nStations[i] = DEFAULT_STATION;
      m_nLastStation = DEFAULT_STATION;
      m_RadioData.Open("fm.dat",
                  CFile::modeCreate | CFile::modeWrite |
CFile::shareDenyWrite);
      m_RadioData.Write(m_nStations, sizeof(m_nStations));
```

```
      m_RadioData.Write(&m_nLastStation, sizeof(m_nLastStation));

   }
   else
   {
      MessageBox("File fm.dat is missing or damaged.\nCreating a new
file.");
      for(int i = 0; i < 9; i++)
         m_nStations[i] = DEFAULT_STATION;
      m_nLastStation = DEFAULT_STATION;
      m_RadioData.Open("fm.dat",
                  CFile::modeCreate | CFile::modeWrite |
CFile::shareDenyWrite);
      m_RadioData.Write(m_nStations, sizeof(m_nStations));
      m_RadioData.Write(&m_nLastStation, sizeof(m_nLastStation));

   }

   floatToString(m_StringFrequency, m_nLastStation);

   /////////////////////////////////////////////////////////////////////////
   **
   ** Code for the ToolTip component
   **
   /////////////////////////////////////////////////////////////////////////*/
   // CG: The following block was added by the ToolTips component.
   {
      // Create the ToolTip control.
      m_tooltip.Create(this);
      m_tooltip.Activate(TRUE);

      // TODO: Use one of the following forms to add controls:
      m_tooltip.AddTool(GetDlgItem(IDC_EXIT_BUTTON), "Exit Program");
      m_tooltip.AddTool(GetDlgItem(IDC_DOWN_BUTTON), "Decrease Frequency");
      m_tooltip.AddTool(GetDlgItem(IDC_UP_BUTTON), "Increase Frequency");
      m_tooltip.AddTool(GetDlgItem(IDC_MUTE_CHECK), "Mute Volume");
      m_tooltip.AddTool(GetDlgItem(IDC_RECORD_BUTTON), "Record Music");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION1_BUTTON), "Preset Station
1");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION2_BUTTON), "Preset Station
2");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION3_BUTTON), "Preset Station
3");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION4_BUTTON), "Preset Station
4");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION5_BUTTON), "Preset Station
5");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION6_BUTTON), "Preset Station
6");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION7_BUTTON), "Preset Station
7");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION8_BUTTON), "Preset Station
8");
      m_tooltip.AddTool(GetDlgItem(IDC_STATION9_BUTTON), "Preset Station
9");
      m_tooltip.AddTool(GetDlgItem(IDC_VOL_SLIDER), "Volume");
      m_tooltip.AddTool(GetDlgItem(ID_HELP), "Help");
```

```
        m_tooltip.AddTool(GetDlgItem(IDC_STOP_BUTTON), "Stop Recording");
        m_tooltip.AddTool(GetDlgItem(IDC_SET_BUTTON), "Edit Preset
Stations");
    }
    return TRUE;  // return TRUE  unless you set the focus to a control
}

void CFMRadioProgramDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDialog::OnSysCommand(nID, lParam);
    }
}

void CFMRadioProgramDlg::OnDestroy()
{
    WinHelp(0L, HELP_QUIT);
    CDialog::OnDestroy();
}

// If you add a minimize button to your dialog, you will need the code
below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

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

        SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        // Background
        CClientDC dc(this);

        RECT textbox = {200, 17, 400, 147};
        // Display frequency
```

```
        LOGFONT logFont;
        logFont.lfHeight = 90;
        logFont.lfWidth = 0;
        logFont.lfEscapement = 0;
        logFont.lfOrientation = 0;
        logFont.lfWeight = 700;
        logFont.lfItalic = 0;
        logFont.lfUnderline = 0;
        logFont.lfStrikeOut = 0;
        logFont.lfCharSet = ANSI_CHARSET;
        logFont.lfOutPrecision = OUT_DEFAULT_PRECIS;
        logFont.lfClipPrecision = CLIP_DEFAULT_PRECIS;
        logFont.lfQuality = PROOF_QUALITY;
        logFont.lfPitchAndFamily = VARIABLE_PITCH | FF_ROMAN;
        strcpy(logFont.lfFaceName, "Times New Roman");

        CFont font;
        font.CreateFontIndirect(&logFont);
        CFont* oldFont = dc.SelectObject(&font);

        // Save old Text colors
        COLORREF oldTextColor = dc.SetTextColor(RGB(0,255,0));
        COLORREF oldBkColor = dc.SetBkColor(RGB(0,0,0));

        UINT oldAlignment = dc.SetTextAlign(TA_RIGHT);
        dc.ExtTextOut(390, 38, ETO_OPAQUE | ETO_CLIPPED, &textbox,
m_StringFrequency, NULL);

        dc.SelectObject(oldFont);
        dc.SetTextColor(oldTextColor);
        dc.SetBkColor(oldBkColor);
        dc.SetTextAlign(oldAlignment);

        CDialog::OnPaint();
    }
}

// The system calls this to obtain the cursor to display while the user
drags
//  the minimized window.
HCURSOR CFMRadioProgramDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CFMRadioProgramDlg::OnVScroll(UINT nSBCode, UINT nPos, CScrollBar*
pScrollBar)
{

    CSliderCtrl * volume = (CSliderCtrl *) pScrollBar;
    WORD position = volume->GetPos();
    MMRESULT auxControl;
    DWORD lr_volume;
    position = 0xFFFF - position;
    lr_volume = position | (position << 16);
    if(m_bMuteCheck == TRUE)
    {
```

```
      auxControl = auxSetVolume(auxID, lr_volume);

      if(auxControl == MMSYSERR_NOERROR)
      {
         m_Volume = position;
      }
      else
         // Reset scrollbar
         volume->SetPos(m_Volume);
   }
   else
      m_Volume = 0xFFFF - position;
   CDialog::OnVScroll(nSBCode, nPos, pScrollBar);
}

BOOL CFMRadioProgramDlg::OpenWaveOut()
{
   MMRESULT openWaveResult;
   wfmt.wFormatTag      = WAVE_FORMAT_PCM;   // Vanilla PCM
   wfmt.nChannels    = 2;           // Stereo
   wfmt.nSamplesPerSec  = 22050;       // 22.05kHz sample rate
   wfmt.wBitsPerSample = 16;          // Bitwidth = 16 bits
   wfmt.nBlockAlign  = 4;           // (nChannels * wBitsPerSample) / 8
   wfmt.nAvgBytesPerSec = 88200;       // nSamplesPerSec * nBlockAlign
   wfmt.cbSize       = 0;           // No extra data for vanilla PCM

   openWaveResult = waveOutOpen(&hWaveOut,      // Device handle to fill
               WAVE_MAPPER,           // ID of device to open
               &wfmt,                 // WAVEFORMATEX describing wave
               0,                     // Callback target
               0,                     // Callback user data
               0);                 // Open flags
   if(openWaveResult == MMSYSERR_NOERROR)
      return TRUE;
   else
      // Error handling
      return FALSE;
}

void CFMRadioProgramDlg::OnExitButton()
{
   // Write out current data to file.
   m_RadioData.SeekToBegin();
   m_RadioData.Write(m_nStations, sizeof(m_nStations));
   m_RadioData.Write(&m_nLastStation, sizeof(m_nLastStation));
   m_RadioData.Close();

   if(m_bRecording == TRUE)
      OnStopButton();

   // Exit program
   CDialog::OnOK();
}

void CFMRadioProgramDlg::OnRecordButton()
{
   if(m_bRecording == FALSE)
```

```
   {
      UINT numInWave = waveInGetNumDevs();
      if(numInWave > 0)
      {
         CButton *pButton = (CButton *) GetDlgItem(IDC_RECORD_BUTTON);
         pButton->SetBitmap(m_hRecordingPic);

         if(OpenWaveIn() == TRUE)
         {
            // Generate buffers and headers
            MMRESULT mmr;
            m_hFirstData = GlobalAlloc(GMEM_MOVEABLE|GMEM_SHARE,
BUFFERSIZE*nBlockAlign);
            if (!m_hFirstData)            // If we didn't get the memory
            {
               // Error Message
               MessageBox("Unable to Record!");
               return;
            }
            m_lpFirstData = (LPSTR)GlobalLock(m_hFirstData);
            if (!m_lpFirstData)                  // If we didn't get the
lock
            {
               GlobalFree (m_hFirstData);        // free the memory
               m_hFirstData = NULL;              // reset the handle
               // Error Message
               MessageBox("Unable to Record!");
               return;
            }
            m_hLastData = GlobalAlloc(GMEM_MOVEABLE|GMEM_SHARE,
BUFFERSIZE*nBlockAlign);
            if (!m_hLastData)          // If we didn't get the memory
            {
               // Error Message
               MessageBox("Unable to Record!");
               return;
            }
            m_lpLastData = (LPSTR)GlobalLock(m_hLastData);
            if (!m_lpLastData)                  // If we didn't get the
lock
            {
               GlobalFree (m_hLastData);        // free the memory
               m_hFirstData = NULL;              // reset the handle
               // Error Message
               MessageBox("Unable to Record!");
               return;
            }
            m_hFirstBuffer.lpData = m_lpFirstData;
            m_hFirstBuffer.dwBufferLength = BUFFERSIZE*nBlockAlign;
            m_hFirstBuffer.dwFlags = 0L;
            m_hLastBuffer.lpData = m_lpLastData;
            m_hLastBuffer.dwBufferLength = BUFFERSIZE*nBlockAlign;
            m_hLastBuffer.dwFlags = 0L;
            mmr = waveInPrepareHeader(m_hWaveIn, &m_hFirstBuffer,
sizeof(m_hFirstBuffer));
            if(mmr != MMSYSERR_NOERROR)
            {
```

```
                MessageBox("Unable to Record!");
                return;
            }
            mmr = waveInPrepareHeader(m_hWaveIn, &m_hLastBuffer,
sizeof(m_hLastBuffer));
            if(mmr != MMSYSERR_NOERROR)
            {
                MessageBox("Unable to Record!");
                return;
            }
            mmr = waveInAddBuffer(m_hWaveIn, &m_hFirstBuffer,
sizeof(m_hFirstBuffer));
            if(mmr != MMSYSERR_NOERROR)
            {
                MessageBox("Unable to Record!");
                return;
            }
            GlobalUnlock(m_hFirstBuffer.lpData);

            mmr = waveInAddBuffer(m_hWaveIn, &m_hLastBuffer,
sizeof(m_hLastBuffer));

            if(mmr != MMSYSERR_NOERROR)
            {
                MessageBox("Unable to Record!");
                return;
            }
            GlobalUnlock(m_hLastBuffer.lpData);

            if(InitializeOutput("temp.dat") == FALSE)
            {
                MessageBox("Recording Failed\nInitializeOutput");
                CButton *pButton = (CButton *)
GetDlgItem(IDC_RECORD_BUTTON);
                pButton->SetBitmap(m_hRecordPic);
            }
            else
            {
                mmr = waveInStart(m_hWaveIn);
                if(mmr != MMSYSERR_NOERROR)
                {
                    MessageBox("Unable to Record!\nwaveInStart");
                    return;
                }
                else
                {
                    m_bRecording = TRUE;
                    CButton *pStopButton = (CButton *)
GetDlgItem(IDC_STOP_BUTTON);
                    pStopButton->EnableWindow(TRUE);
                }

            }

        }
        else
        {
```

```
            MessageBox("Unable to record.\nWave device busy.");
            CButton *pButton = (CButton *) GetDlgItem(IDC_RECORD_BUTTON);
            pButton->SetBitmap(m_hRecordPic);
         }
      }

      else
      {
         MessageBox("Cannot Record FM Signal.\nNo wave recording
capabilities found.");

      }
   }
}

void CFMRadioProgramDlg::OnStopButton()
{
   MMRESULT results;
   CButton *pButton = (CButton *) GetDlgItem(IDC_RECORD_BUTTON);
   pButton->SetBitmap(m_hRecordPic);
   if(m_bRecording == TRUE)
   {
      m_bRecording = FALSE;
      results = waveInStop(m_hWaveIn);
      if(results != MMSYSERR_NOERROR)
      {
         MessageBox("Problem stopping recording");
      }

      results = waveInReset(m_hWaveIn);
      while(results != MMSYSERR_NOERROR)
      {
         results = waveInReset(m_hWaveIn);
      }
      if(m_hFirstBuffer.dwFlags & WHDR_DONE)
      {
         BufferFull(&m_hFirstBuffer);
      }
      if(m_hLastBuffer.dwFlags & WHDR_DONE)
      {
         BufferFull(&m_hLastBuffer);
      }

      //
      // Ascend from subchunk
      //
      if (mmioAscend (hmmio,&mmckinfoSubChunk,0) != 0)
      {
         mmioClose (hmmio,0);
         MessageBox("Error writing out data");
         return;
      }
      //
      // Ascend from  RIFF chunk
      //
      if (mmioAscend (hmmio,&mmckinfo,0) != 0)
      {
```

```
            mmioClose (hmmio,0);
            MessageBox("Error writing out data");
            return;
        }

        //
        // Close file
        //

        mmioClose (hmmio,0);

        waveInUnprepareHeader(m_hWaveIn, &m_hFirstBuffer,
sizeof(m_hFirstBuffer));
        waveInUnprepareHeader(m_hWaveIn, &m_hLastBuffer,
sizeof(m_hLastBuffer));

        results = waveInClose(m_hWaveIn);
        if(results != MMSYSERR_NOERROR)
        {
            if(results & MMSYSERR_INVALHANDLE)
            {
                TRACE("Appears that wave device has been closed.\n");
            }
            else if(results & WAVERR_STILLPLAYING)
            {
                TRACE("Problem is the buffer is still being used.\n");
                results = waveInReset(m_hWaveIn);
                if(results & MMSYSERR_INVALHANDLE)
                    TRACE("Appears that wave device has been closed.\n");
                else if(results & MMSYSERR_NOMEM)
                    TRACE("Problem with memory.\n");
            }
            else if(results & MMSYSERR_NOMEM)
                TRACE("Problem with memory.\n");
        }
        // Free Memory
        GlobalUnlock(m_hFirstData);
        GlobalFree(m_hFirstData);
        GlobalUnlock(m_hLastData);
        GlobalFree(m_hFirstData);

        // Allow user to save data or discard it.
        CFileDialog dlg(FALSE, "wav", "*.wav",
                        OFN_HIDEREADONLY | OFN_LONGNAMES | OFN_OVERWRITEPROMPT |
OFN_NOCHANGEDIR,
                        "Wave File  (*.wav)|*.wav|");
        if(dlg.DoModal() == IDOK)
        {

            // Aggressive and overwrites files
            ::CopyFile("temp.dat", dlg.GetPathName(), FALSE);
            ::DeleteFile("temp.dat");
        }
        else
        {
            ::DeleteFile("temp.dat");
        }
```

```
    }
    CButton *pStopButton = (CButton *) GetDlgItem(IDC_STOP_BUTTON);
    pStopButton->EnableWindow(FALSE);
}

void CFMRadioProgramDlg::OnUpButton()
{
    if(m_nLastStation <= 1077)
        m_nLastStation += 2;
    else
        m_nLastStation = 879;

    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnDownButton()
{
    if(m_nLastStation >= 881)
        m_nLastStation -= 2;
    else
        m_nLastStation = 1079;

    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnMuteCheck()
{
    WORD volume;
    if(m_bMuteCheck == TRUE)
        m_bMuteCheck = FALSE;
    else
        m_bMuteCheck = TRUE;

    MMRESULT auxControl;
    volume = 0xFFFF - m_Volume;
    DWORD lr_volume = volume | (volume << 16);
    if(m_bMuteCheck == TRUE)
    {
        auxControl = auxSetVolume(auxID, lr_volume);
    }
    else
    {
        lr_volume = 0L;
        auxControl = auxSetVolume(auxID, lr_volume);
    }
}

void CFMRadioProgramDlg::OnSetButton()
{
    CSetStation dlg;
    for(int i = 0; i < 9; i++)
```

```
    {
        dlg.m_nStations[i] = m_nStations[i];
    }
    dlg.m_NewStation = (double)(m_nLastStation)/10.0;
    int ret = dlg.DoModal();
    if(ret == IDOK)
    {
        for(int i = 0; i < 9; i++)
        {
            if(dlg.m_nStations[i] >= 879 && dlg.m_nStations[i] <= 1079)
                m_nStations[i] = dlg.m_nStations[i];
        }
    }
}

void CFMRadioProgramDlg::OnStation1Button()
{
    m_nLastStation = m_nStations[0];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation2Button()
{
    m_nLastStation = m_nStations[1];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation3Button()
{
    m_nLastStation = m_nStations[2];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation4Button()
{
    m_nLastStation = m_nStations[3];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation5Button()
{
    m_nLastStation = m_nStations[4];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
```

```
    OnPaint();
}

void CFMRadioProgramDlg::OnStation6Button()
{
    m_nLastStation = m_nStations[5];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation7Button()
{
    m_nLastStation = m_nStations[6];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation8Button()
{
    m_nLastStation = m_nStations[7];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::OnStation9Button()
{
    m_nLastStation = m_nStations[8];
    floatToString(m_StringFrequency, m_nLastStation);
    //
    // Add code here to change frequency on FM card
    OnPaint();
}

void CFMRadioProgramDlg::floatToString(CString &str, int value)
{
    int decimal;
    char char_temp[5];
    str.Empty();
    decimal = value % 10;
    value /= 10;
    itoa(value, char_temp, 10);
    str = char_temp;
    str += ".";
    itoa(decimal, char_temp, 10);
    str += char_temp;
}


int CFMRadioProgramDlg::GetAuxLineInNum()
{
    AUXCAPS auxcaps;
```

```
    CString details;
    int numDevices = auxGetNumDevs();
    if(numDevices == 0)
       // Failed to find an auxillary input jack
       return(-1);
    for(int i = 0; i < numDevices; i++)
    {
       auxGetDevCaps(i, &auxcaps, sizeof(auxcaps));
       details = (LPSTR)auxcaps.szPname;
       if(details.Find("Line") != -1)
       {
          if(details.Find("In") != -1)
             return(i);
          else if(details.Find("in") != -1)
             return(i);
          else if(details.Find("IN") != -1)
             return(i);
       }
       else if(details.Find("line") != -1)
       {
          if(details.Find("In") != -1)
             return(i);
          else if(details.Find("in") != -1)
             return(i);
          else if(details.Find("IN") != -1)
             return(i);
       }
       if(details.Find("LINE") != -1)
       {
          if(details.Find("In") != -1)
             return(i);
          else if(details.Find("in") != -1)
             return(i);
          else if(details.Find("IN") != -1)
             return(i);
       }

    }
    return(0);
}

BOOL CFMRadioProgramDlg::OpenWaveIn()
{
    MMRESULT openWaveResult;
    m_wfmt.wFormatTag     = WAVE_FORMAT_PCM;    // Vanilla PCM
    m_wfmt.nChannels      = 1;             // Stereo
    m_wfmt.nSamplesPerSec    = 22050;         // 22.05kHz sample rate
    m_wfmt.wBitsPerSample    = 16;            // Bitwidth = 16 bits
    m_wfmt.nBlockAlign       = nBlockAlign;     // (nChannels *
wBitsPerSample) / 8
    m_wfmt.nAvgBytesPerSec   = 44100;         // nSamplesPerSec * nBlockAlign
    m_wfmt.cbSize         = 0;             // No extra data for vanilla PCM

    openWaveResult = waveInOpen(&m_hWaveIn,       // Device handle to fill
                 WAVE_MAPPER,             // ID of device to open
                 &m_wfmt,                 // WAVEFORMATEX describing wave
                 (DWORD)GetSafeHwnd(),    // Callback target
```

```
                0L,                    // Callback user data
                CALLBACK_WINDOW);          // Open flags
   if(openWaveResult == MMSYSERR_NOERROR)
   {
      return TRUE;
   }
   else
   {
      if(openWaveResult&WAVERR_BADFORMAT)
         TRACE("Unsupported Format\n");
      // Error handling
      return FALSE;
   }
}

LRESULT CFMRadioProgramDlg::WindowProc(UINT message, WPARAM wParam, LPARAM
lParam)
{
   // Handle the full buffer
   if(message == MM_WIM_DATA)
   {
      WAVEHDR *ptr;
      ptr = (WAVEHDR *)lParam;
      BufferFull(ptr);
   }

   else if(message == MM_MIXM_CONTROL_CHANGE)
   {
      LineChange(GetSafeHwnd(), wParam, lParam);
   }
   return CDialog::WindowProc(message, wParam, lParam);
}

void CFMRadioProgramDlg::LineChange(HWND hwnd, WPARAM wParam, LPARAM
lParam)
{
   MMRESULT aux;
   DWORD auxVolume;
   WORD tempVolume;
   if(auxID != -1 && m_bMuteCheck == TRUE)
   {
      aux = auxGetVolume(auxID, &auxVolume);
      if(aux == MMSYSERR_NOERROR)
      {
         tempVolume = (WORD)(0xFFFF - (auxVolume & 0xFFFF));
         if(tempVolume != m_Volume)
         {
            m_Volume = tempVolume;
            CSliderCtrl *pVolume = (CSliderCtrl *)
GetDlgItem(IDC_VOL_SLIDER);
            pVolume->SetPos(m_Volume);
         }
         else
         {
            return;
         }
      }
```

```
      else if(aux == MMSYSERR_BADDEVICEID)
      {
         MessageBox("Unable to Control Volume");
         // Disable slider
      }
   }

}


void CFMRadioProgramDlg::BufferFull(WAVEHDR* buffHand)
{
   DWORD bytesWritten;
   MMRESULT mmr;
   HPSTR temp;

   if(hmmio != NULL && ((m_hFirstBuffer.dwFlags & WHDR_DONE) ||
(m_hLastBuffer.dwFlags & WHDR_DONE)))
   {
      if(&m_hFirstBuffer == buffHand)
      {
         temp = (HPSTR)GlobalLock(m_hFirstBuffer.lpData);
         bytesWritten = mmioWrite(hmmio,
                              temp,
                              m_hFirstBuffer.dwBytesRecorded);
         if(bytesWritten != m_hFirstBuffer.dwBytesRecorded)
            MessageBox("File corruption may have occurred!");

         waveInUnprepareHeader(m_hWaveIn, &m_hFirstBuffer,
sizeof(m_hFirstBuffer));
         GlobalUnlock(m_hFirstBuffer.lpData);
         m_hFirstBuffer.lpData = (LPSTR)GlobalLock(m_hFirstData);
         if(m_hFirstBuffer.lpData == NULL)
            MessageBox("Failed to lock first buffer down.");
         m_hFirstBuffer.dwBufferLength = BUFFERSIZE*nBlockAlign;
         m_hFirstBuffer.dwFlags = 0L;

         mmr = waveInPrepareHeader(m_hWaveIn, &m_hFirstBuffer,
sizeof(m_hFirstBuffer));
         if(mmr != MMSYSERR_NOERROR)
         {
            MessageBox("Unable to Record!");
            return;
         }

         if(m_bRecording == TRUE)
         {
            mmr = waveInAddBuffer(m_hWaveIn, &m_hFirstBuffer,
sizeof(m_hFirstBuffer));
            if(mmr != MMSYSERR_NOERROR)
            {
               MessageBox("Unable to finish recording!");
               OnStopButton();
            }
         }
         GlobalUnlock(m_hFirstBuffer.lpData);
      }
```

```
      else if(&m_hLastBuffer == buffHand)
      {
         temp = (HPSTR)GlobalLock(m_hLastBuffer.lpData);
         bytesWritten = mmioWrite(hmmio,
                              temp,
                              m_hLastBuffer.dwBytesRecorded);
         if(bytesWritten != m_hLastBuffer.dwBytesRecorded)
            MessageBox("File corruption may have occurred!");

         waveInUnprepareHeader(m_hWaveIn, &m_hLastBuffer,
sizeof(m_hLastBuffer));
         GlobalUnlock(m_hLastBuffer.lpData);
         m_hLastBuffer.lpData = (LPSTR)GlobalLock(m_hLastData);
         m_hLastBuffer.dwBufferLength = BUFFERSIZE*nBlockAlign;
         m_hLastBuffer.dwFlags = 0L;
         mmr = waveInPrepareHeader(m_hWaveIn, &m_hLastBuffer,
sizeof(m_hLastBuffer));
         if(mmr != MMSYSERR_NOERROR)
         {
            MessageBox("Unable to Record!");
            return;
         }
         if(m_bRecording == TRUE)
         {
            mmr = waveInAddBuffer(m_hWaveIn, &m_hLastBuffer,
sizeof(m_hLastBuffer));
            if(mmr != MMSYSERR_NOERROR)
            {
               MessageBox("Unable to finish recording!");
               OnStopButton();
            }
         }
         GlobalUnlock(m_hLastBuffer.lpData);
      }
   }
}

BOOL CFMRadioProgramDlg::InitializeOutput(const CString & fname)
{
      //=====================================================
   //
   //  RIFF File I/O:

   //
   // Open the .WAV file for writing
   //

   if (!(hmmio = mmioOpen((LPSTR)(const char*)fname, NULL,
            MMIO_WRITE | MMIO_CREATE | MMIO_ALLOCBUF )))
   {
      MessageBox("Failed to mmioOpen");
      return FALSE;
   }

   //
   // Create a RIFF chunk whose form type is 'WAVE'
   //
```

```
mmckinfo.fccType = mmioFOURCC('W','A','V','E');

if (mmioCreateChunk (hmmio, &mmckinfo, MMIO_CREATERIFF) != 0)
{
   MessageBox("Failed on mmioCreateChunk WAVE");
   mmioClose (hmmio,0);
   return FALSE;
}

//
// Create a subchunk whose ID is 'fmt '
//

mmckinfoSubChunk.ckid = mmioFOURCC('f','m','t',' ');

if (mmioCreateChunk (hmmio, &mmckinfoSubChunk, 0) != 0)
{
   MessageBox("Failed on mmioCreateChunk FMT");
   mmioClose (hmmio,0);
   return FALSE;
}

//
// Write out format info
//

if (mmioWrite (hmmio,(HPSTR)&m_wfmt, sizeof (PCMWAVEFORMAT)) == -1)
{
   MessageBox("Failed on mmioWrite");
   mmioClose (hmmio,0);
   return FALSE;
}

//
// Ascend from subchunk
//
MMRESULT mm;
mm = mmioAscend(hmmio,&mmckinfoSubChunk,0);
if (mm != 0)
{

   if(MMIOERR_CANNOTSEEK & mm)
      MessageBox("Failed on mmioAscend & CANNOTSEEK");
   if(MMIOERR_CANNOTWRITE & mm)
      MessageBox("Failed on mmioAscend & CANNOTWRITE");
   mmioClose (hmmio,0);
   return FALSE;
}

//
// Create a subchunk whose ID is 'data'
//

mmckinfoSubChunk.ckid = mmioFOURCC('d','a','t','a');

if (mmioCreateChunk (hmmio, &mmckinfoSubChunk, 0) != 0)
```

```
    {
        MessageBox("Failed on mmioCreateChunk DATA");
        mmioClose (hmmio,0);
        return FALSE;
    }

    return TRUE;
}

BOOL CFMRadioProgramDlg::PreTranslateMessage(MSG* pMsg)
{
    // CG: The following block was added by the ToolTips component.
    {
        // Let the ToolTip process this message.
        m_tooltip.RelayEvent(pMsg);
    }
    return CDialog::PreTranslateMessage(pMsg);
    // CG: This was added by the ToolTips component.
}

void CFMRadioProgramDlg::OnOK()
{
}

void CFMRadioProgramDlg::OnCancel()
{
    OnExitButton();
}
```

9.2 FM Radio ProgramDlg.h

```
// FM Radio ProgramDlg.h : header file
//

#include <mmsystem.h>
#include "setstation.h"
#define DEFAULT_STATION 879
const WORD nBlockAlign = 2;
const DWORD BUFFERSIZE = 0xFFFF;

#ifndef SIZEOF_WAVEFORMATEX
#define SIZEOF_WAVEFORMATEX(pwfx)    ((WAVE_FORMAT_PCM==(pwfx)-
>wFormatTag)?sizeof(PCMWAVEFORMAT):(sizeof(WAVEFORMATEX)+(pwfx)->cbSize))
#endif

#if
!defined(AFX_FMRADIOPROGRAMDLG_H__8234217A_8B64_11D1_924E_0020C5E37344__INC
LUDED_)
#define
AFX_FMRADIOPROGRAMDLG_H__8234217A_8B64_11D1_924E_0020C5E37344__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000

/////////////////////////////////////////////////////////////////////////////
//
// CFMRadioProgramDlg dialog

class CFMRadioProgramDlg : public CDialog
{
// Construction
public:
   void floatToString(CString &str, int value);
   CFMRadioProgramDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
   //{{AFX_DATA(CFMRadioProgramDlg)
   enum { IDD = IDD_FMRADIOPROGRAM_DIALOG };
   BOOL  m_bMuteCheck;
   //}}AFX_DATA

   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CFMRadioProgramDlg)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
   virtual LRESULT WindowProc(UINT message, WPARAM wParam, LPARAM lParam);
   //}}AFX_VIRTUAL

// Implementation
private:
   void BufferFull(WAVEHDR *buffHand);
   BOOL AcmAppFileNew(HWND hwnd);
   BOOL OpenWaveIn(void);
```

```
    int GetAuxLineInNum();
    int m_nLastStation;
    int m_nStations[9];
    CFileException m_FileError;
    CFile m_RadioData;
    CString m_StringFrequency;
    BOOL OpenWaveOut(void);

    // Button members
    HBITMAP m_hRecordPic;
    HBITMAP m_hRecordingPic;
    HBITMAP m_hStopPic;
    HBITMAP m_hHelpPic;

    CBitmap m_RecordBitmap;
    CBitmap m_RecordingBitmap;
    CBitmap m_StopBitmap;
    CBitmap m_HelpBitmap;

    // Frequency Members
// float frequency;

    // Volume members
    WAVEFORMATEX wfmt;
    HWAVEOUT hWaveOut;
    WORD m_Volume;
    int auxID;
    DWORD dwControlID;

    // Recording capabilities
    WAVEFORMATEX m_wfmt;
    HWAVEIN m_hWaveIn;
    WAVEHDR m_hFirstBuffer;
    WAVEHDR m_hLastBuffer;
    HANDLE   m_hFirstData;
    HANDLE   m_hLastData;
    LPSTR m_lpFirstData;
    LPSTR m_lpLastData;
    BOOL  m_bRecording;
    HMMIO hmmio;
    MMCKINFO mmckinfo;      // Chunk info structure
    MMCKINFO mmckinfoSubChunk; // Chunk info structure

public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
    void LineChange(HWND hwnd, WPARAM wParam, LPARAM lParam);

protected:
    CToolTipCtrl m_tooltip;
    BOOL InitializeOutput(const CString& fname);
    HICON m_hIcon;

    // Generated message map functions
    //{{AFX_MSG(CFMRadioProgramDlg)
    virtual BOOL OnInitDialog();
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
    afx_msg void OnDestroy();
```

```
    afx_msg void OnPaint();
    afx_msg HCURSOR OnQueryDragIcon();
    afx_msg void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    afx_msg void OnExitButton();
    afx_msg void OnRecordButton();
    afx_msg void OnStopButton();
    afx_msg void OnUpButton();
    afx_msg void OnDownButton();
    afx_msg void OnMuteCheck();
    afx_msg void OnSetButton();
    afx_msg void OnStation1Button();
    afx_msg void OnStation2Button();
    afx_msg void OnStation3Button();
    afx_msg void OnStation4Button();
    afx_msg void OnStation5Button();
    afx_msg void OnStation6Button();
    afx_msg void OnStation7Button();
    afx_msg void OnStation8Button();
    afx_msg void OnStation9Button();
    virtual void OnOK();
    virtual void OnCancel();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_FMRADIOPROGRAMDLG_H__8234217A_8B64_11D1_924E_0020C5E37344__INC
LUDED_)
```

9.3 SetStation.cpp

```
// SetStation.cpp : implementation file
//

#include "stdafx.h"
#include "FM Radio Program.h"
#include "SetStation.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
//
// CSetStation dialog


CSetStation::CSetStation(CWnd* pParent /*=NULL*/)
   : CDialog(CSetStation::IDD, pParent)
{
   //{{AFX_DATA_INIT(CSetStation)
   m_NewStation = 0.0;
   m_StationSelector = -1;
   m_CurrentStation = _T("");
   //}}AFX_DATA_INIT
}


void CSetStation::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CSetStation)
   DDX_Text(pDX, IDC_NEW_EDIT, m_NewStation);
   DDV_MinMaxDouble(pDX, m_NewStation, 87.9, 107.9);
   DDX_Radio(pDX, IDC_RADIO1, m_StationSelector);
   DDX_Text(pDX, IDC_CURRENT_STATIC, m_CurrentStation);
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSetStation, CDialog)
   //{{AFX_MSG_MAP(CSetStation)
   ON_EN_CHANGE(IDC_NEW_EDIT, OnChangeNewEdit)
   ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
   ON_BN_CLICKED(IDC_RADIO2, OnRadio2)
   ON_BN_CLICKED(IDC_RADIO3, OnRadio3)
   ON_BN_CLICKED(IDC_RADIO4, OnRadio4)
   ON_BN_CLICKED(IDC_RADIO5, OnRadio5)
   ON_BN_CLICKED(IDC_RADIO6, OnRadio6)
   ON_BN_CLICKED(IDC_RADIO7, OnRadio7)
   ON_BN_CLICKED(IDC_RADIO8, OnRadio8)
   ON_BN_CLICKED(IDC_RADIO9, OnRadio9)
```

```
    ON_BN_CLICKED(IDC_SET_BUTTON, OnSetButton)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
//
// CSetStation message handlers

void CSetStation::OnCancel()
{
    // TODO: Add extra cleanup here

    CDialog::OnCancel();
}

void CSetStation::OnOK()
{
    // TODO: Add extra validation here

    CDialog::OnOK();
}

void CSetStation::OnChangeNewEdit()
{
    // TODO: Add your control notification handler code here

}

void CSetStation::OnRadio1()
{
    floatToString(m_StationString, m_nStations[0]);
    SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio2()
{
    floatToString(m_StationString, m_nStations[1]);
    SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio3()
{
    floatToString(m_StationString, m_nStations[2]);
    SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio4()
{
    floatToString(m_StationString, m_nStations[3]);
    SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio5()
{
    floatToString(m_StationString, m_nStations[4]);
    SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}
```

```
void CSetStation::OnRadio6()
{
   floatToString(m_StationString, m_nStations[5]);
   SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio7()
{
   floatToString(m_StationString, m_nStations[6]);
   SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio8()
{
   floatToString(m_StationString, m_nStations[7]);
   SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnRadio9()
{
   floatToString(m_StationString, m_nStations[8]);
   SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::OnSetButton()
{
   UpdateData(TRUE);
   if(m_NewStation >= 87.9 && m_NewStation <= 107.9)
   {
      switch(m_StationSelector)
      {
         case 0:
         {
            m_nStations[0] = (int)(m_NewStation*10);
            break;
         }
         case 1:
         {
            m_nStations[1] = (int)(m_NewStation*10);
            break;
         }
         case 2:
         {
            m_nStations[2] = (int)(m_NewStation*10);
            break;
         }
         case 3:
         {
            m_nStations[3] = (int)(m_NewStation*10);
            break;
         }
         case 4:
         {
            m_nStations[4] = (int)(m_NewStation*10);
            break;
         }
```

```
            case 5:
            {
               m_nStations[5] = (int)(m_NewStation*10);
               break;
            }
            case 6:
            {
               m_nStations[6] = (int)(m_NewStation*10);
               break;
            }
            case 7:
            {
               m_nStations[7] = (int)(m_NewStation*10);
               break;
            }
            case 8:
            {
               m_nStations[8] = (int)(m_NewStation*10);
               break;
            }
            default:
               break;
         }
      }
      floatToString(m_StationString, m_nStations[m_StationSelector]);
      SetDlgItemText(IDC_CURRENT_STATIC, m_StationString);
}

void CSetStation::floatToString(CString &str, int value)
{
      int decimal;
      char char_temp[5];
      str.Empty();
      decimal = value % 10;
      value /= 10;
      itoa(value, char_temp, 10);
      str = char_temp;
      str += ".";
      itoa(decimal, char_temp, 10);
      str += char_temp;
}

BOOL CSetStation::PreTranslateMessage(MSG* pMsg)
{
      // CG: The following block was added by the ToolTips component.
      {
         // Let the ToolTip process this message.
         m_tooltip.RelayEvent(pMsg);
      }
      return CDialog::PreTranslateMessage(pMsg);    // CG: This was added by
the ToolTips component.
}

BOOL CSetStation::OnInitDialog()
{
      CDialog::OnInitDialog();    // CG: This was added by the ToolTips
component.
```

```
    // CG: The following block was added by the ToolTips component.
    {
        // Create the ToolTip control.
        m_tooltip.Create(this);
        m_tooltip.Activate(TRUE);

        // TODO: Use one of the following forms to add controls:
        m_tooltip.AddTool(GetDlgItem(IDC_SET_BUTTON), "Set New Station");
        m_tooltip.AddTool(GetDlgItem(IDOK), "OK to Exit");
        m_tooltip.AddTool(GetDlgItem(IDCANCEL), "Cancel Any Changes");
        m_tooltip.AddTool(GetDlgItem(IDC_NEW_EDIT), "Enter New Station
Here");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO1), "Preset Station 1");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO2), "Preset Station 2");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO3), "Preset Station 3");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO4), "Preset Station 4");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO5), "Preset Station 5");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO6), "Preset Station 6");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO7), "Preset Station 7");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO8), "Preset Station 8");
        m_tooltip.AddTool(GetDlgItem(IDC_RADIO9), "Preset Station 9");
        m_tooltip.AddTool(GetDlgItem(IDC_CURRENT_STATIC), "Current Station
for Preset Number");
    }
    return TRUE;    // CG: This was added by the ToolTips component.
}
```

9.4 SetStation.h

```
#if
!defined(AFX_SETSTATION_H__986B3BA0_9292_11D1_924E_0020C5E37344__INCLUDED_)
#define AFX_SETSTATION_H__986B3BA0_9292_11D1_924E_0020C5E37344__INCLUDED_

#if _MSC_VER >= 1000
#pragma once
#endif // _MSC_VER >= 1000
// SetStation.h : header file
//

/////////////////////////////////////////////////////////////////////////////
//
// CSetStation dialog

class CSetStation : public CDialog
{
// Construction
public:
   int m_nStations[9];
   CSetStation(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
   //{{AFX_DATA(CSetStation)
   enum { IDD = IDD_SETSTATION };
   double   m_NewStation;
   int      m_StationSelector;
   CString  m_CurrentStation;
   //}}AFX_DATA


// Overrides
   // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CSetStation)
   protected:
   virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
   //}}AFX_VIRTUAL

// Implementation
public:
   virtual BOOL PreTranslateMessage(MSG* pMsg);
   CString m_StationString;
   void floatToString(CString &str, int value);
protected:
   CToolTipCtrl m_tooltip;
   virtual BOOL OnInitDialog();
   // Generated message map functions
   //{{AFX_MSG(CSetStation)
   virtual void OnCancel();
   virtual void OnOK();
   afx_msg void OnChangeNewEdit();
   afx_msg void OnRadio1();
   afx_msg void OnRadio2();
   afx_msg void OnRadio3();
```

```
    afx_msg void OnRadio4();
    afx_msg void OnRadio5();
    afx_msg void OnRadio6();
    afx_msg void OnRadio7();
    afx_msg void OnRadio8();
    afx_msg void OnRadio9();
    afx_msg void OnSetButton();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Developer Studio will insert additional declarations
immediately before the previous line.

#endif //
!defined(AFX_SETSTATION_H__986B3BA0_9292_11D1_924E_0020C5E37344__INCLUDED_)
```