Cameron Currey
Sam Nazari
Tim Pruss
December 12, 2008

iCarTunes Project Report

iCarTunes Introduction

Imagine driving in a car and the most amazing song begins playing on the radio. The artist and song name is unknown, but you want to hear the song again as soon as possible. With the press of a button on a steering wheel, your car connects to *iTunes* through a satellite connection and downloads the song directly to your *iPod*. The iCarTunes project makes this possible.

iCarTunes is a project developed to be installed in the dashboard of automobiles. A small display screen on the dashboard shows the user interface the user can interact. The user can control the system with two simple buttons installed on the steering wheel of the car. One button navigates through the menus, and the other button makes selections. The user is able to save the current song title and artist playing on the radio to a download list. The user is able to select one of the saved songs from the download menu. The system will automatically purchase and download from the *iTunes* network through a satellite Internet connection. The user then has the option to load downloaded songs onto an *iPod* connected to the system. There is also an option to sync the newly downloaded songs to an *iTunes* library on a home computer. The song will be transferred through the satellite Internet connection to a home computer connected to the Internet and store the song into *iTunes*.

External Components and Networks

iCarTunes makes use of two external networks explained here. These are described to help the reader understand the functionality of iCarTunes.

- **iTunes**

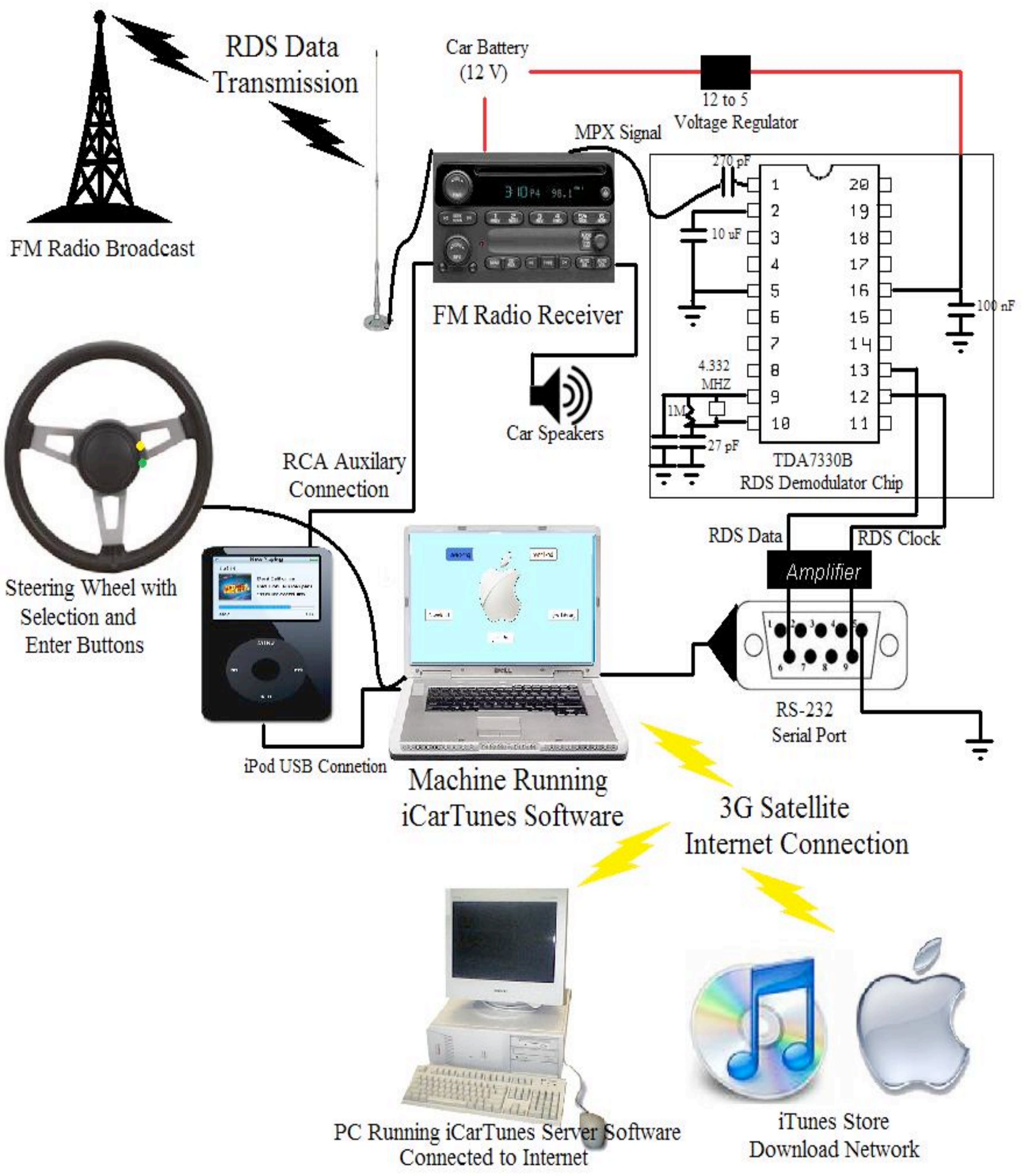
iTunes is a popular music download network developed by *Apple*. Users can download nearly any song in *m4a* format for a small fee. The advent of *iPod* technology has made *iTunes* one of the most useful and popular music networks, which is why iCarTunes interfaces with it to download songs.

- **RDS – Radio Data System**

RDS is is a communications protocol standard for sending small amounts of digital information using conventional FM radiobroadcasts. Several radio stations use RDS to transmit song title and artist information when playing music. iCarTunes utilizes this information being transmitted by using an RDS demodulator chip. The data is extracted from the FM transmission from the bits known as RT (Radio Text) and the software decodes the information and interfaces it with *iTunes*.

Hardware Schematic, Design, and Functionality

Below is a diagram of all the hardware and software components used to develop iCarTunes and all the interconnections. This section describes each component and how it functions in the entire architecture of the project.

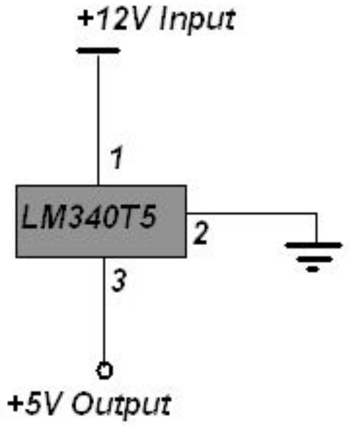

RDS Data Transmission
FM Radio Broadcast
Car Battery (12 V)
12 to 5 Voltage Regulator
MPX Signal
270 pF
10 uF
100 nF
4.332 MHZ
1M
27 pF
TDA7330B RDS Demodulator Chip
FM Radio Receiver
Car Speakers
RCA Auxilary Connection
Steering Wheel with Selection and Enter Buttons
RDS Data
RDS Clock
Amplifier
RS-232 Serial Port
iPod USB Connetion
Machine Running iCarTunes Software
3G Satellite Internet Connection
PC Running iCarTunes Server Software Connected to Internet
iTunes Store Download Network

- **FM Radio Receiver**

The FM radio receiver is a typical car stereo connected to a car's power connection to the battery. The antenna picks up FM transmissions containing RDS data and plays the songs on the connected car speakers. A VRCD220FD radio by VR^3 was used for the iCarTunes project.

- **RDS Data Extraction Circuits**

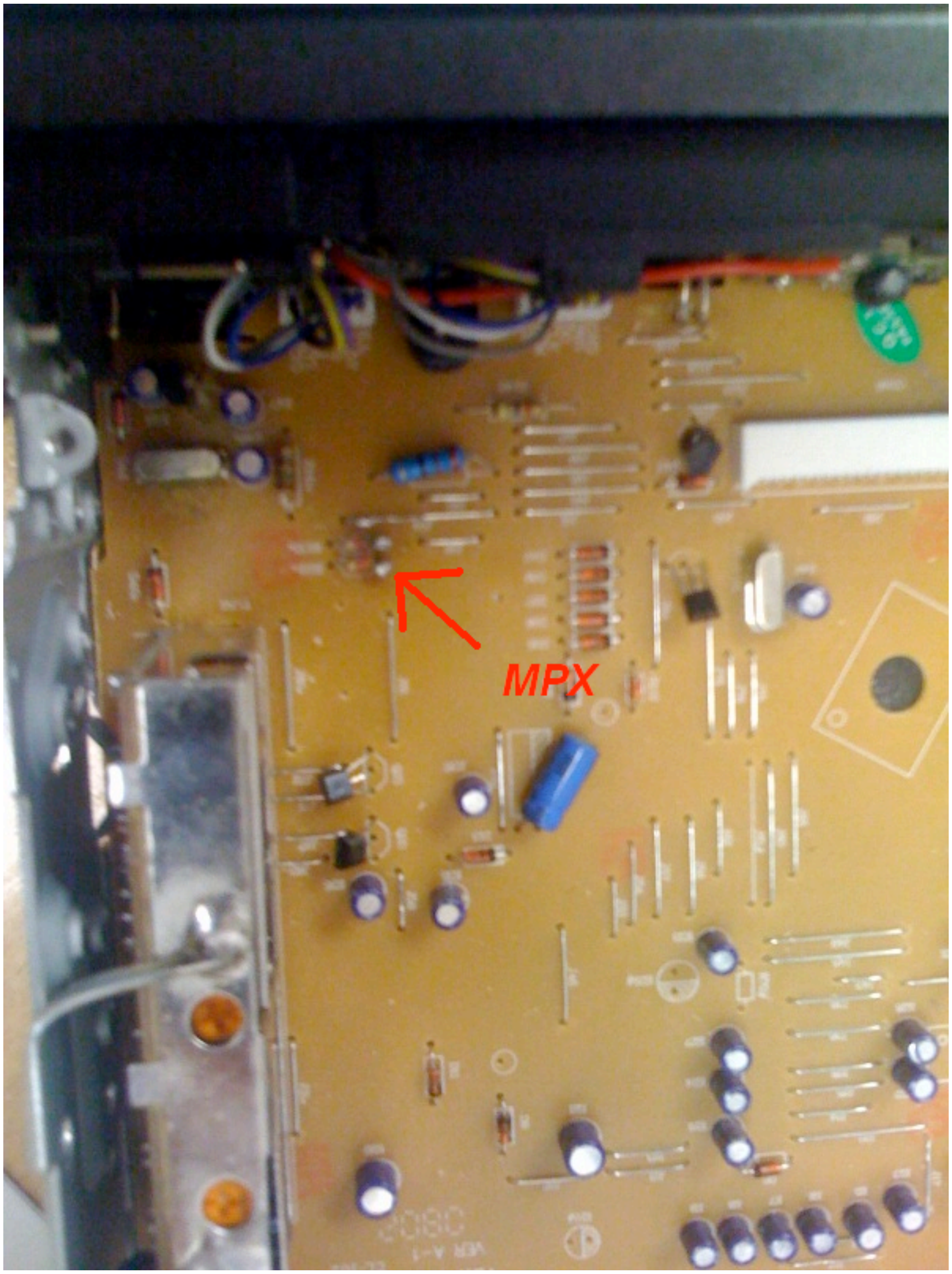
The circuits designed consist of four parts: voltage demodulator, RDS demodulator chip, amplifier circuit, and an inversion circuit. The voltage demodulator is a simple circuit consisting of just one chip, LM340T5.



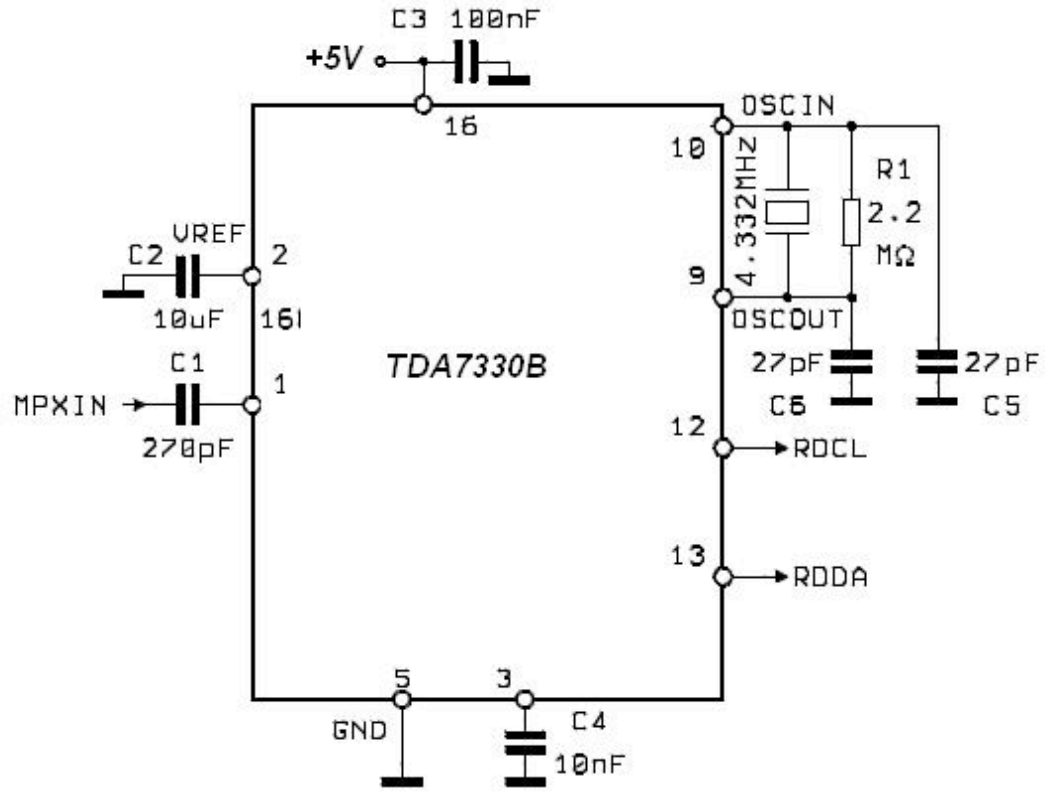
Voltage Demodulator to Provide Voltages for Components

Every car has a 12V battery, but the other three circuits require 5V, which is provided by this demodulator chip.

The RDS chip circuit takes in the MPX signal found in the radio. To get it from the radio the CD player in the console was removed. The following photo shows where the MPX signal is found in the VRCD220FD radio.



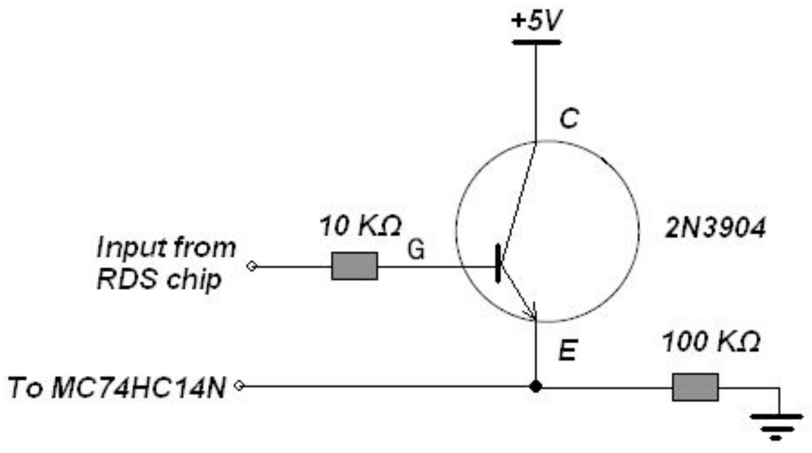
MPX Signal found on the VRCD220FD radio.



RDS Demodulator Circuit – As Shown in the Overall Schematic

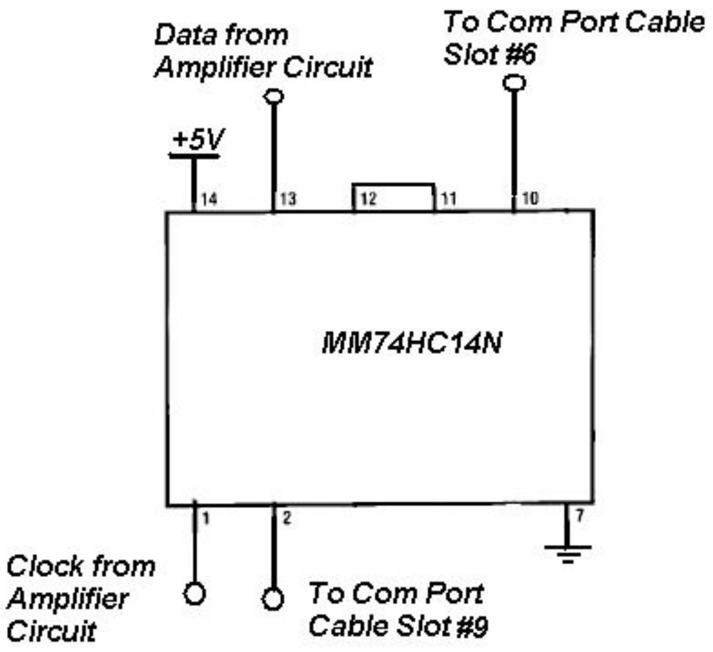
The RDS demodulator chip is the essential component for the software to decode the song currently playing. The RDS demodulator chip is a TDA7330B chip. The RDS demodulator chip extracts the RDS data from the MPX signal found in the FM circuitry on the radio. The chip is powered by a 5-volt source and runs on a 4.332 MHz crystal. Many external capacitors are connected to the chip to reduce the noise in the FM transmissions as shown in the schematic. The chip has two outputs, the clock signal running at 1187.5 Hz and the data signal. Every time the clock runs on the downward slope the data output changes to either a high or low value. The data itself contains the binary information being broadcast by the radio station.

The clock and data coming out of the RDS demodulator chip have amplitudes of 100 mV. An amplitude of 5V is needed for the machine running the software to detect the signal over the serial port. So these signals are piped into an amplifier circuit.



Amplifier Circuit to Amplify RDS Data to 5V

One separate amplifier circuit is required for the data and clock. The output goes to the inversion circuit to invert the clock and to stabilize both signals.


Data from Amplifier Circuit
To Com Port Cable Slot #6
+5V
14
13
12
11
10
MM74HC14N
1
2
7
Clock from Amplifier Circuit
To Com Port Cable Slot #9

Inversion Circuit to Invert the Signal

The inverter circuit consists of one digital inverter chip. The clock signal needs to be inverted for the software, so it goes through the inverter once. The data signal just needs to be stabilized, so it goes through the inverter twice. The outputs connect to the serial cable at the specified slots.

- **Serial Port Connection**

The clock and data signal from the inverter chip are connected to pins 9 and 6 of a standard RS-232 serial port. Pin 5 is connected to ground. The serial port is connected to a *Windows* machine with *.NET* installed.

- **Machine Running iCarTunes Software**

Any computer with the *Windows* operating system and *iTunes* installed is able to run the iCarTunes software. The final project used a *Dell* laptop with a 701 MHz *AMD* processor with 128 MB of RAM. The laptop also had an internal wireless card that was used to connect to the Internet through the University of Utah's wireless access. An actual implementation with proper funding would have used a 3G Internet card to connect to the Internet. This machine was used because it had a serial port input that was necessary for the decoding of the RDS data being input.

Two USB ports were also required. One port was used for the *iPod* USB connection and the other port was used for the steering wheel column connected to the laptop to control the software.

The application software was developed in *C#* and will be described in more detail in the software section.

- **Steering Wheel with Selection and Enter Buttons**

The steering wheel is designed to act as a simple USB mouse. Inside the steering wheel is a basic mouse circuit. The two buttons on the steering wheel (cruise and speed control) were soldered to the corresponding mouse button wires on the USB mouse circuit. When the user pressed either the cruise control or speed control buttons, they would close the circuit on the USB mouse and act as a left or right click. The software was designed to be fully interactive with simple usage of these two buttons. The left click mechanism allowed the user to navigate the software, and the right click allowed for execution of the current selection (more on this later).

- **The iPod**

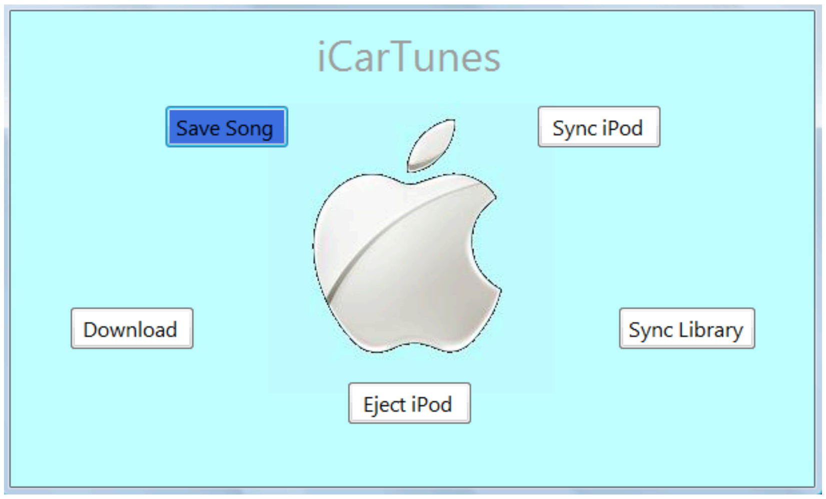
An *80 GB iPod Classic* was used for the project demonstration. The *iPod* was able to receive the songs being downloaded from *iTunes* through the USB connector. The *iPod* itself was connected to an auxiliary RCA port in the FM receiver so songs loaded onto the *iPod* could be played back in the car. This component of the hardware was necessary for the full demonstration of iCarTunes functionality.

- **Home PC Running iCarTunes Server Software**

Any *Windows* machine with an Ethernet connection was capable of running the iCarTunes Server Software. The PCs in the Senior Hardware Lab at the University of Utah were used for the demonstration of the project for receiving songs.

Software and Functionality

The software running on the *Dell* Laptop was central for computing RDS data being input, interfacing with *iTunes* for downloads, and displaying a user interface for the user to interact with. The software was developed in *C#* using *Visual Studio 2005* as a *Windows Application*.

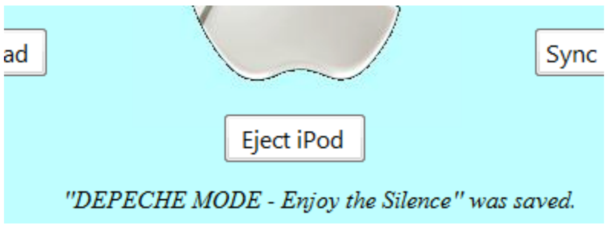


iCarTunes Main Application Interface

The software would run idle and display the screen until the user selected one of the menus with the steering wheel. Once a menu was selected, critical sections of the code were executed. This section describes what each section of the code does and how it functions in the entire design.

- **Save Song**

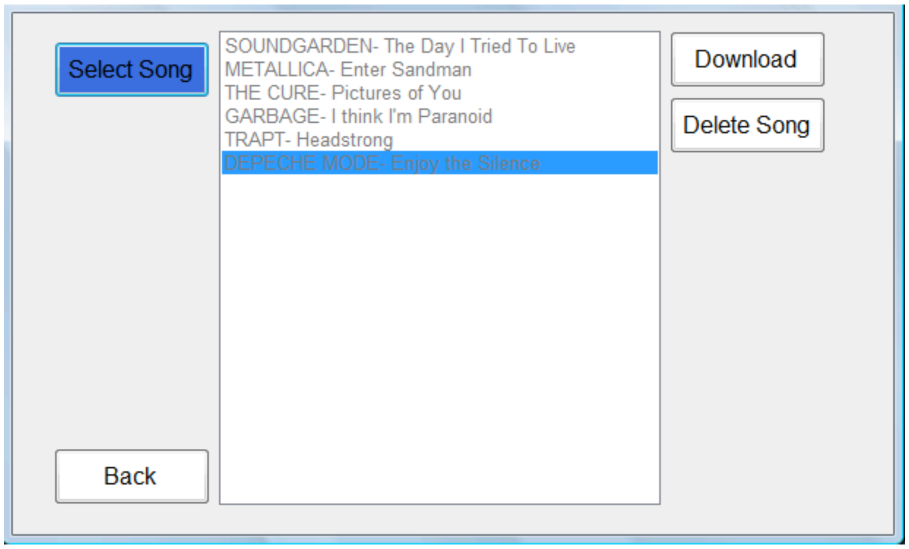
When the user selects the "Save Song" button, the software deciphers the RDS data being input on the serial port. The RDSDX utility package deciphers the data and save the "Radio Text" bits to a file. The software then reads that file and displays the song currently playing and saves the information to a list. The song is displayed on the bottom of the application, or an error is reported if the data could not be decoded.



Song Decoded from RDS and Displayed.

- **Download**

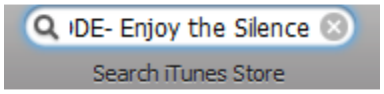
When the user selects "Download", a new menu is displayed. The menu shows all the songs that have been saved and are available to be downloaded. The user is able to select which saved song they want to download then download it by selecting the "Download" button.


Select Song
SOUNDGARDEN- The Day I Tried To Live
METALLICA- Enter Sandman
THE CURE- Pictures of You
GARBAGE- I think I'm Paranoid
TRAPT- Headstrong
DEPECHE MODE- Enjoy the Silence
Download
Delete Song
Back

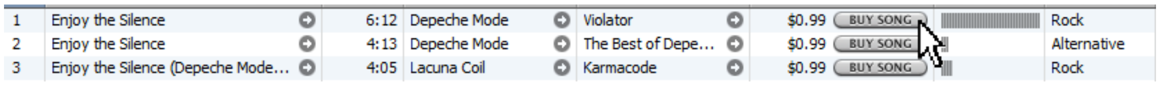
Download Menu Displaying Saved Songs

When "Download" button is selected, an *AutoIt* script is executed to interface with the *iTunes* Store. *AutoIt* is a scripting language that provides automatic commands in *Windows*, such as mouse clicks, window resizing, and typing.

First the script disables all input from the user. Mouse clicks are not available while the script is preparing iTunes for downloads. The script then types the search query in the *iTunes* Store.



Once the search completes, if there are any relevant search results, the script selects the most relevant one.



1	Enjoy the Silence	6:12	Depeche Mode	Violator	$0.99	BUY SONG	Rock
2	Enjoy the Silence	4:13	Depeche Mode	The Best of Depe…	$0.99	BUY SONG	Alternative
3	Enjoy the Silence (Depeche Mode…	4:05	Lacuna Coil	Karmacode	$0.99	BUY SONG	Rock

AutoIt Script Making a Purchase from the iTunes Store

iTunes then uses the active Internet connection and to download the song from *iTunes* using the default account information and charges the user for the purchase. Once the download is complete, it minimizes *iTunes* and restores control back to the main application.

- **Sync iPod**

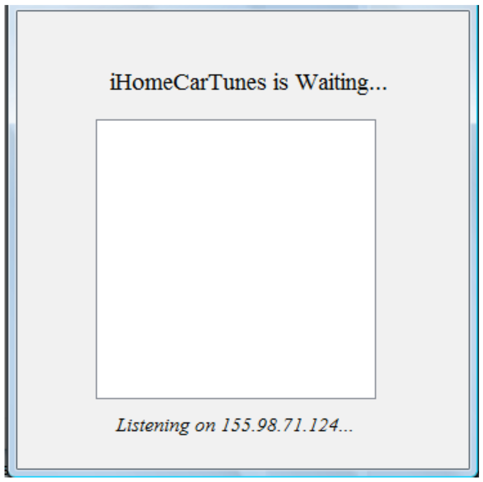
The "Sync iPod" button utilizes the *iTunes Software Development Kit* downloadable from *www.apple.com*. The *iTunes SDK* provides an API for programs to interact with iTunes. When the "Sync iPod" button is selected, the application uses the SDK code for syncing *iPod*s connected to a USB port. If no *iPod* is connected an error message is provided, otherwise all songs that have been downloaded are loaded onto the *iPod*.

- **Eject iPod**

This option only appears when an *iPod* is actually connected. A timer in the code checks every 5 seconds to see if an *iPod* has been connected to display this option.
The "Eject iPod" button is similar to the "Sync iPod" button in functionality. When selected, it calls an iTunes SDK function to eject the *iPod* and allowing the user to disconnect the *iPod* from the machine without loss of any song data.

- **Sync Library**

The Sync Library button transmits all the newly downloaded songs files to a computer connected to the Internet running the application called "iHomeCarTunes".

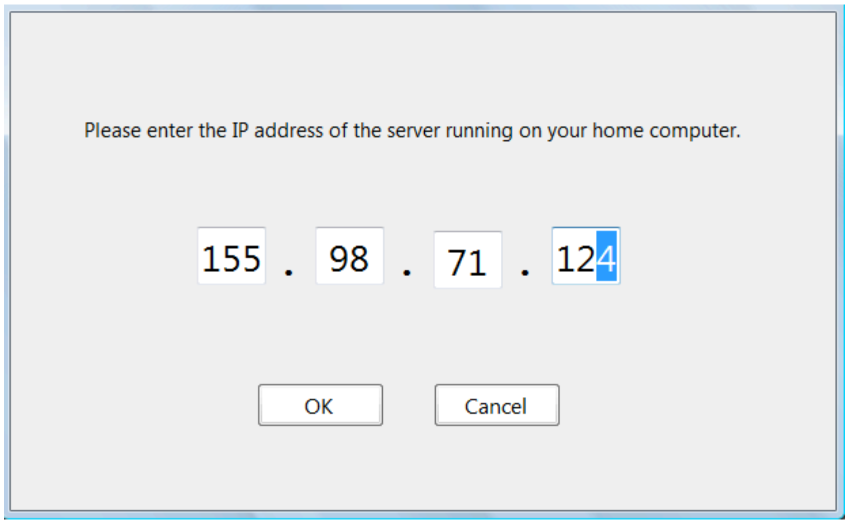


iHomeCarTunes is Waiting...

Listening on 155.98.71.124...

iHomeCarTunes running on a PC connected to 155.98.71.124

Every time a song is downloaded, it is saved to a list. When the “Sync Library” button is pressed it attempts to open a socket connection with the software running on the PC. To specify the IP address of the software, the user selects “Sync Library” and presses and holds the button for 5 seconds to display the following menu.


Please enter the IP address of the server running on your home computer.

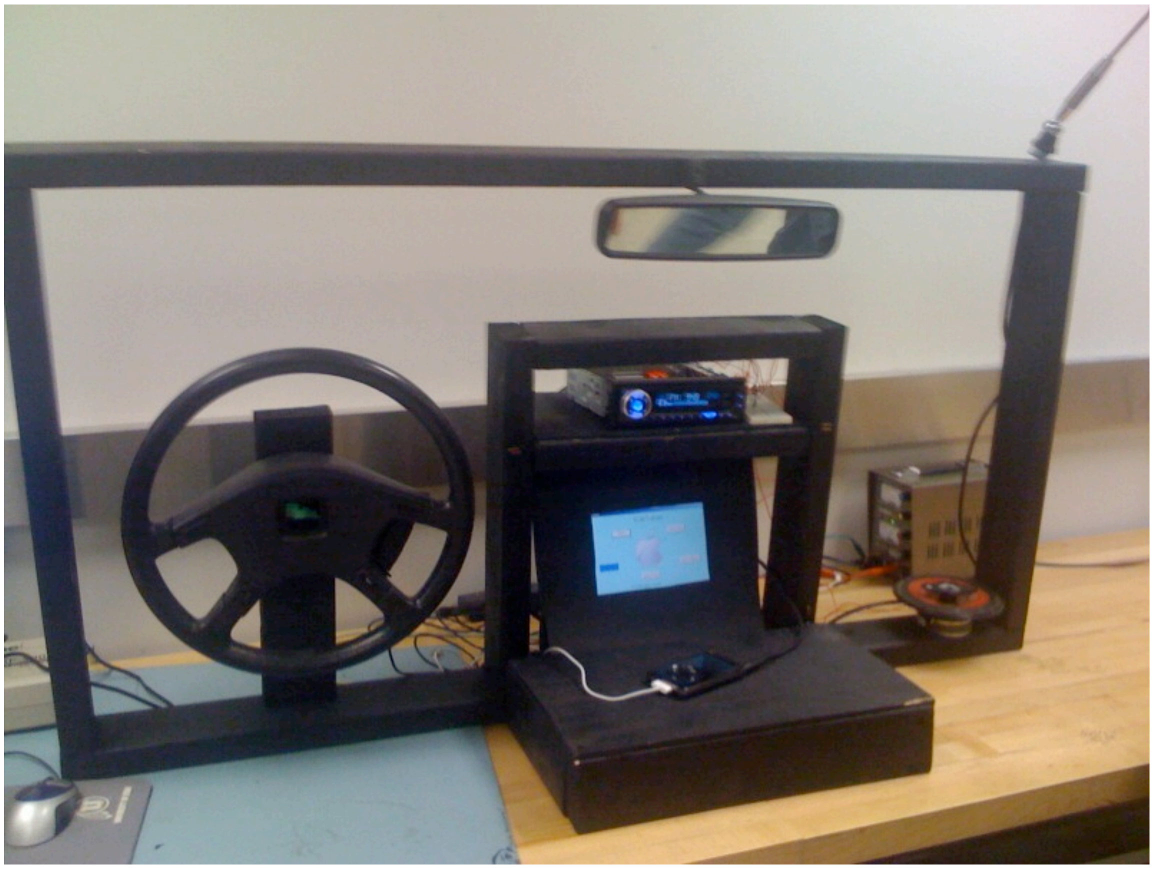
155 . 98 . 71 . 124

User Entering IP Address of Home Computer

Using the steering wheel the user can specify the IP address. Once the IP address is entered, the files downloaded are transferred. First an acknowledgment message is sent over the socket connection. Once iHomeCarTunes receives the message it sends an acknowledgement back to iCarTunes. iCarTunes then begins reading the file and reads enough bytes to send in a packet and a byte to acknowledge if more data is being sent. iHomeCarTunes receives the packet, writes the data to a file, then sends back a response requesting for the next packet. iCarTunes reads the next amount of bytes to fill the packet, sends the data and the process continues until all the bytes in the file have been transmitted. Once iHomeCarTunes has received all the files, it copies the files to the *iTunes* directory and uses the *iTunes SDK* to load them into the library.

Final Project Demonstration

A wooden display was built to house all the components and demonstrate the functionality of the iCarTunes project. The display was designed to resemble the dashboard of a car. It held a place for the antenna, FM Radio receiver, a screen for the laptop, the steering wheel, and all the interconnections running behind.


iCarTunes Project Display

Applicability and Possibilities for Improvement

iCarTunes had a lot of expensive components: laptops, *iPods*, $0.99 for each download, cables, and connectors. The price of all these components limited the amount able to be implemented. The original design of iCarTunes was to run on a microcomputer to fit easily into a dashboard. The design also included an external display to display the user interface.

Another problem with iCarTunes is its reliance on RDS data transmissions. RDS is not widely used in the United States, and many radio stations are inconsistent on updating broadcast information. The project was demonstrated using 94.9 FM since it is the most reliable radio station for RDS data in the Salt Lake Valley. The project design could easily be extended to support XM radio as originally intended. XM radio has similar functionality of transmitting song information. However, XM radio requires a monthly subscription fee, so the design incorporated the cheaper RDS format. Hopefully radio stations would be more consistent on updating RDS data in radio transmissions if iCarTunes was ever commercially developed.

Another flaw with the design is the *iTunes* Store interface. Much research was involved in figuring out how to actually download from *iTunes*. The *iTunes SDK* provides a lot of functionality to interface with *iTunes*, but almost no support for downloading. In researching this issue online, *Apple* specifically designed it this way to prevent malicious applications from accessing sensitive account information. *Apple* did announce in an article from *www.apple.com* that in 2009 they would release an *iTunes* Store SDK for applications on the popular *iPhone*.

iCarTunes is a great commercial idea. *Apple* has become dominant in the music industry and *iTunes* is the most popular music seller in the world. iCarTunes would be a great addition to the *iTunes* market and increase the potential of when and where *iTunes* users can purchase songs for downloads.

Source Citing and Thanks

AutoIt Scripting Software for Interfacing with iTunes
http://www.autoitscript.com/autoit3/downloads.shtml

RDSDX Software Utility for Decoding RDS Data
http://home.scarlet.be/~wijnherm/rdsdxdownload.htm

iTunes Download for Downloading from the iTunes Store
http://www.apple.com/itunes/download/

iTunes Software Development Kit for Interfacing with the iPod and iTunes
http://developer.apple.com/sdk/

C# Examples for File Transfers
http://www.csharphelp.com/archives2/archive335.html

Appendix I: Bill of Materials

RS-232 Serial Cable	Hardware Lab
Dell Laptop	Tim Pruss Donatation
iPod and USB Connector	Cameron Currey Donation
Car Speaker	Tim Pruss Donatation
Car Radio Model #VRCd220FD	$50
FM Radio Power Connector	$20
Radio Antenna	$20
Resistors, Capacitors, Wire, and Clock	$10
RCA Audio Input Connector	$20
Steering Wheel and Rear View Mirror	$10
2x4 Studs, Screws, and Spray Paint for Display	$20
iTunes purchases during Debugging and Demo	$25
Lessons Learned While Building iCarTunes	*Priceless*
Total	*$175*

Appendix II: iCarTunes C# Source Code (Critical Sections Only)

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using System.IO;
using iTunesLib;
//stuff for file transfers
using System.Net;
using System.Net.Sockets;

//Threading stuff
using System.Threading;

namespace iCarTunes
{
    public partial class MainForm : Form
    {
        //A list of songs that we have saved away and wish to download
        private List<Song> songs;
        //A list of all the songs we have successfully downloaded
        private List<Song> downloads;
        //The connection to the itunes interface
        private iTunesApp itunes;
        //bool to keep track whether an ipod is connected to the
machine, updates every 5 seconds
        private bool ipodConnected;

        //Server IP address
        string serverIP;

        //Creates a new MainForm and new song list.
        public MainForm()
        {
            InitializeComponent();
            songdisplayLabel.Text = "";
            //Reads all the saved songs from the file.
            songs = new List<Song>();
            StreamReader reader = new StreamReader("songs.txt");
            string song = reader.ReadLine();
            while (song != null)
            {
                int ampPos = song.IndexOf("&&");
                string artist = song.Substring(0, ampPos);
                string title = song.Substring(ampPos + 2, song.Length -
ampPos - 2);
                songs.Add(new Song(title, artist));
                song = reader.ReadLine();
            }
            reader.Close();
            //Reads all the saved downloads from the file.
            downloads = new List<Song>();
            StreamReader read = new StreamReader("downloads.txt");
            song = read.ReadLine();
            while (song != null)
            {
                int ampPos = song.IndexOf("&&");
                string artist = song.Substring(0, ampPos);
                string title = song.Substring(ampPos + 2, song.Length - 
ampPos - 2);
                downloads.Add(new Song(title, artist));
                song = read.ReadLine();
            }
            read.Close();

            //Reads the saved away IP address from the file
            serverIP = "";
            StreamReader readIP = new StreamReader("IP.txt");
            serverIP = readIP.ReadLine();
            readIP.Close();

            //Set up itunes application
            itunes = new iTunesApp();
            try
            {
                if (itunes.Sources.Count == 2)
                {
                    ipodConnected = true;
                    ejectButton.Visible = true;
                }
                else
                    ipodConnected = false;
            }
            catch (Exception)
            {
            }
        }

        //Executed when Save Song button is pressed. Extracts data from
        //the hardware serial cable and adds the Song to the list.
        private void saveSongButton_Click(object sender, EventArgs e)
        {

            //Delete all files from previous screen dumps
            DirectoryInfo directory = new 
DirectoryInfo("RDSDX\\ScreenDumps");
            FileInfo[] files = directory.GetFiles();
            foreach (FileInfo fi in files)
            {
                fi.Delete();
            }


            String artist = "";
            String title = "";

            saveSongButton.BackColor = System.Drawing.Color.Gray;
            //Execute the song data gathering script
            System.Diagnostics.Process script =
System.Diagnostics.Process.Start("getsongscript.au3");



            //Run idle until the script finishes execution
            while (!script.HasExited)
            {
                System.Threading.Thread.Sleep(2000);
            }

            //Read the Title and artist from the file
            try
            {
                DirectoryInfo foo = new
DirectoryInfo("RDSDX\\ScreenDumps");
                FileInfo[] playing = foo.GetFiles();
                StreamReader read = new
StreamReader(playing[0].OpenRead());

                string readline = "";
                //Read until we get to the line containing PI ÿ
                while (!readline.Contains("PI □"))
                {
                    readline = read.ReadLine();
                }
                //Read two more lines
                readline = read.ReadLine();
                //readline now contains the line with the song data.
                readline = read.ReadLine();

                int index = readline.IndexOf("94.9 Z-Rock - ");
                //Readline now has the radio station off
                readline = readline.Substring(index + 14,
readline.Length - 15 - index);
                index = readline.IndexOf(" - ");
                title = readline.Substring(0,index);

                //readline = readline.Substring(
                readline = readline.Substring(index + 3,
readline.Length - 4 - index);
                artist = readline.Trim();
                read.Close();


            }
            catch (Exception)
            {
                //If we fail for some reason, report an error.
                songdisplayLabel.Text = "Invalid song playing. Please
try later.";
                songdisplayLabel.ForeColor = System.Drawing.Color.Red;
                saveSongButton.BackColor =
System.Drawing.Color.RoyalBlue;
                return;
            }
            //Add the song to the saved-song-list if it is not already
there
            //and update display
            bool inSongList = false;
            foreach (Song s in songs)
            {
                if (s.artist.Equals(artist) && s.title.Equals(title))
                    inSongList = true;
            }
            if (inSongList)
            {
                songdisplayLabel.Text = "\"" + artist + "- " + title +
"\" already saved on list.";
                songdisplayLabel.ForeColor = System.Drawing.Color.Red;
            }
            else
            {
                songdisplayLabel.Text = "\"" + artist + "- " + title +
"\" saved.";
                songdisplayLabel.ForeColor =
System.Drawing.Color.Black;
                songs.Add(new Song(title, artist));
            }

            saveSongButton.BackColor = System.Drawing.Color.RoyalBlue;
        }

        //Executed when Download button is pressed. Displays the
download menu
        //And updates the list of songs if they were edited.
        private void downloadButton_Click(object sender, EventArgs e)
        {
            DownloadMenu menu = new DownloadMenu(songs, downloads,
itunes);
            menu.ShowDialog();
            songs = menu.songs;
            downloads = menu.downloads;
        }

        //Executed when Sync iPod button is pressed. Puts the
application
        //to sleep until the syncing is complete.
        private void ipodButton_Click(object sender, EventArgs e)
        {
            //Determine if an iPod is connected to this machine
            if (!ipodConnected)
            {
                songdisplayLabel.ForeColor = System.Drawing.Color.Red;
                songdisplayLabel.Text = "   Sync Failed, no iPod
connected!";
            }
            else
            {
                //Update the display before syncing
                songdisplayLabel.ForeColor =
System.Drawing.Color.Black;
```

```
                songdisplayLabel.Text = "       Syncing iPod. Please 
wait...";
                ipodButton.BackColor = System.Drawing.Color.Gray;
                this.Update();

                //Put the application to sleep and Sync the iPod 
connected.
                itunes.UpdateIPod();

                System.Threading.Thread.Sleep(5000);

                //This is to make the "Apple" Logo to reappear so the 
script works again
                itunes.OpenURL("hi");
                System.Threading.Thread.Sleep(5000);
                if (itunes.CurrentTrack.Name.Equals("hi"))
                    itunes.CurrentTrack.Delete();

                //Restore the display after syncing completes.
                songdisplayLabel.ForeColor = System.Drawing.Color.Red;
                songdisplayLabel.Text = "             Sync complete!";
                ipodButton.BackColor = System.Drawing.Color.RoyalBlue;
            }

        }

        //Executed when Eject iPod button is pressed. Allows the ipod
        //to be connected.
        private void ejectButton_Click(object sender, EventArgs e)
        {
            //Gets an enumerator to move 2 down to eject that source.
            IEnumerator travel = itunes.Sources.GetEnumerator();
            travel.MoveNext();
            travel.MoveNext();
            IITIPodSource foo = (IITIPodSource)travel.Current;

            //Update display before ejecting
            ejectButton.BackColor = System.Drawing.Color.Gray;

            songdisplayLabel.ForeColor = System.Drawing.Color.Black;
            songdisplayLabel.Text = "       Ejecting iPod. Please 
wait...";
            this.Update();

            //Eject the iPod
            foo.EjectIPod();

            System.Threading.Thread.Sleep(37000);

            //Restore the display
            ejectButton.BackColor = System.Drawing.Color.RoyalBlue;
            songdisplayLabel.ForeColor = System.Drawing.Color.Black;
            songdisplayLabel.Text = "iPod ejected. It is safe to 
disconnect.";

        }
        //selection keeps track of which button is highlighted
        //0 = saveSongButton
        //1 = downloadButton
        //2 = ipodButton
        //3 = syncButton
        //4 = ejectButton
        private int selection = 0;
        private void MainForm_Click(object sender, EventArgs e)
        {
            MouseEventArgs f = (MouseEventArgs)e;
            if (f.Button == MouseButtons.Left)
            {
                selection = selection + 1;
                if (selection == 4 && !ipodConnected)
                    selection = 0;
                else if (selection == 4 && ipodConnected)
                    selection = 4;
                else if (selection == 5)
                    selection = 0;

                if (selection == 0)
                {
                    saveSongButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    downloadButton.BackColor =
System.Drawing.Color.White;
                    ipodButton.BackColor = System.Drawing.Color.White;
                    syncButton.BackColor = System.Drawing.Color.White;
                    ejectButton.BackColor = System.Drawing.Color.White;
                    songdisplayLabel.Text = "";
                }
                if (selection == 1)
                {
                    saveSongButton.BackColor =
System.Drawing.Color.White;
                    downloadButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    ipodButton.BackColor = System.Drawing.Color.White;
                    syncButton.BackColor = System.Drawing.Color.White;
                    ejectButton.BackColor = System.Drawing.Color.White;
                }
                if (selection == 2)
                {
                    saveSongButton.BackColor =
System.Drawing.Color.White;
                    downloadButton.BackColor =
System.Drawing.Color.White;
                    ipodButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    syncButton.BackColor = System.Drawing.Color.White;
                    ejectButton.BackColor = System.Drawing.Color.White;
                }
                if (selection == 3)
                {
                    saveSongButton.BackColor =
System.Drawing.Color.White;
```

```
                    downloadButton.BackColor =
System.Drawing.Color.White;
                    ipodButton.BackColor = System.Drawing.Color.White;
                    syncButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    ejectButton.BackColor = System.Drawing.Color.White;
                    songdisplayLabel.ForeColor =
System.Drawing.Color.Black;
                    songdisplayLabel.Text = "Press and hold to
configure synchronization.";
                }
                if (selection == 4)
                {
                    saveSongButton.BackColor =
System.Drawing.Color.White;
                    downloadButton.BackColor =
System.Drawing.Color.White;
                    ipodButton.BackColor = System.Drawing.Color.White;
                    syncButton.BackColor = System.Drawing.Color.White;
                    ejectButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    songdisplayLabel.Text = "";
                }
            }
            else
            {
                //if a right click, call the callback for the
appropriate button press
                if (selection == 0)
                    saveSongButton_Click(null, null);
                if (selection == 1)
                    downloadButton_Click(null, null);
                if (selection == 2)
                    ipodButton_Click(null, null);
                if (selection == 3)
                    syncButton_Click(null, null);
                if (selection == 4)
                    ejectButton_Click(null, null);
            }
        }

        //If a double click occurs, treat it as a single click
        private void MainForm_DoubleClick(object sender, EventArgs e)
        {
            MainForm_Click(sender, e);
        }

        //Keeps track of whether the mouse is currently up or down
(used for the timer)
        private bool mouseDown = false;
        private void MainForm_MouseDown(object sender, MouseEventArgs
e)
        {
            mouseDown = true;
        }

        private void MainForm_MouseUp(object sender, MouseEventArgs e)
        {
            mouseDown = false;
        }

        //This "tick" is called every 5 seconds
        private bool downLastTick = false;
        private int count = 0;
        private void connectionTimer_Tick(object sender, EventArgs e)
        {
            //Determine if an ipod is connected to the machine, we can
tell
            //by if there are 2 sources available in itunes.
            try
            {
                if (itunes.Sources.Count == 2)
                {
                    ipodConnected = true;
                    ejectButton.Visible = true;

                    if(count < 2)
                    {

                        System.Diagnostics.Process script =
System.Diagnostics.Process.Start("minimizescript.au3");

                        //Run idle until the script finishes execution
                        while (!script.HasExited)
                        {
                            System.Threading.Thread.Sleep(2000);
                        }
                        count = count + 1;
                    }

                }
                else
                {
                    ipodConnected = false;
                    ejectButton.Visible = false;
                    count = 0;
                }
            }
            catch { }

            //The other thing we do is determine if the right click has
been held
            //down for 5 seconds. If it has, we activate the IP Menu
            if (mouseDown)
                downLastTick = true;
            else
                downLastTick = false;

            if (downLastTick && selection == 3)
            {
                mouseDown = false;
                IPForm ipform = new IPForm(serverIP);
                ipform.ShowDialog();
                serverIP = ipform.IP;
            }

        }

        //This occurs anytime the program is shutdown. Before it exits
        //It writes all the saved songs to a file as well as the server
IP.
        private void MainForm_FormClosing(object sender,
FormClosingEventArgs e)
        {
            StreamWriter writer = new StreamWriter("songs.txt");
            foreach(Song s in songs)
            {
                writer.WriteLine(s.artist + "&&" + s.title);
            }
            writer.Close();

            StreamWriter write = new StreamWriter("downloads.txt");
            foreach (Song s in downloads)
            {
                write.WriteLine(s.artist + "&&" + s.title);
            }
            write.Close();


            StreamWriter writeIP = new StreamWriter("IP.txt");
            writeIP.WriteLine(serverIP);
            writeIP.Close();

            //Close the itunes connection
            //itunes.Quit(); Commented to make debugging faster

        }

        //All threading and Network communication stuff goes here
        Socket serversocket;
        private System.Windows.Forms.Label label1;
        Socket clientsock = null;
        string[] cmdList = null;
        string[] DFSCommandList;

        private void syncButton_Click(object sender, EventArgs e)
        {
            songdisplayLabel.ForeColor = System.Drawing.Color.Red;
            try
            {
                serversocket = new Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                serversocket.Blocking = true;

                IPHostEntry IPHost = Dns.Resolve(serverIP);
                IPAddress[] addr = IPHost.AddressList;

                IPEndPoint ipepServer = new IPEndPoint(addr[0], 8090);
                serversocket.Connect(ipepServer);
                clientsock = serversocket;
                Thread MainThread = new Thread(new
ThreadStart(listenclient));
                MainThread.Start();

            }
            catch (SocketException se)
            {
                //Console.WriteLine(se.Message);
            }
            catch (Exception eee)
            {
                MessageBox.Show("Socket Connect Error.\n\n" +
eee.Message + "\nPossible Cause: Server Already running. Check the
tasklist for running processes", "Startup Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                Application.Exit();
            }
        }


        void listenclient()
            {
                  Socket sock = clientsock ;

            //Send a message with the file we want to transfer
            string cmd = "C:\\TEMP\\foo.m4a";
            FileInfo fi = new FileInfo(cmd);
                  byte[] sender =
System.Text.Encoding.ASCII.GetBytes("FILE " + cmd + " " +
fi.Length.ToString()) ;
                  sock.Send(sender, sender.Length,0) ;

                  try
                  {
                        while ( sock != null )
                        {
                              cmd = "" ;
                              byte[] recs = new byte[32767];
                              int rcount =
sock.Receive(recs,recs.Length,0) ;
                              string clientmessage =
System.Text.Encoding.ASCII.GetString(recs) ;
                              clientmessage =
clientmessage.Substring(0,rcount);

                              string smk = clientmessage ;

                              cmdList = null ;
                              cmdList = clientmessage.Split(' ');
                              string execmd = cmdList[0];

                              sender = null ;
                              sender = new Byte[32767];

                    //Send the file name to the server

                    //We connected to the server just fine, so
                    //Let's just begin sending the file now...
                    /*if (execmd == "SERVER")
                    {
                        SERVER_DOWNLOADING(cmdList, sock);
                    }*/
                        }
                  }
                  catch(Exception Se)
                  {
                        string s = Se.Message;
                        Console.WriteLine(s);
                  }
        }

        void SERVER_DOWNLOADING(string[] cmdList, Socket s)
        {
            DFSCommandList = cmdList;
            //The number of files we are downloading.
            int cnt = 3;
            string[] mUploading = new String[5];
            Socket sock = s;

            string parm1 = "";
            string parm2 = "";

            for (int i = 1; i < DFSCommandList.Length - 1; i++)
                parm1 = parm1 + " " + DFSCommandList[i];
            parm1 = parm1.Trim();
            parm2 = DFSCommandList[DFSCommandList.Length - 1];

            try
            {
                FileInfo ftemp = new FileInfo(parm1);
                long total = ftemp.Length;
                long rdby = 0;
                int len = 0;
                string login_client_machine = "";

                mUploading[0] = parm1;
                mUploading[1] = total.ToString();
                mUploading[2] = "0";
                mUploading[3] = "";
                mUploading[4] = login_client_machine;

                //listView4.Items.Add(new ListViewItem(mUploading));

                byte[] buffed = new byte[1024];
                //Open the file requested for download
                FileStream fin = new FileStream(parm1, FileMode.Open,
FileAccess.Read);
                //One way of transfer over sockets is Using a
NetworkStream
                //It provides some useful ways to transfer data
                NetworkStream nfs = new NetworkStream(sock);

                //lock the Thread here
                //                        lock(this)
                while (rdby < total && nfs.CanWrite)
                {
                    //Read from the File (len contains the number of
bytes read)
                    len = fin.Read(buffed, 0, buffed.Length);
                    //Write the Bytes on the Socket
                    nfs.Write(buffed, 0, len);
                    //Increase the bytes Read counter
                    rdby = rdby + len;

                }
                //Display a Message Showing Sucessful File Transfer
                fin.Close();
            }
            catch (Exception ed)
            {
                Console.WriteLine("A Exception occured in transfer" +
ed.ToString());
                MessageBox.Show(ed.Message);
            }
        }

        //This handles the receiving of input over the serial port
        //previous is the time from the last received pin changed
        private long previous = 0;
        private List<int> incomingData = new List<int>();
        private void serialPort_PinChanged(object sender,
System.IO.Ports.SerialPinChangedEventArgs e)
        {
            long currentTime = 0; //Get the currentTime
            //the elapsed time (in seconds) of the last pin change.
            double elapsed = (currentTime - previous)/1000;

            //We are only interested in Pin 9 on the serial port.
            //If we read 00000000, then we know the value was low.
            //If we read 10000000, then we know the value was high.
            //dataread will have either a value of 1 or 0.
            int dataread = serialPort.ReadByte() >> 7;

            //Now we need to figure out how many 1s or 0s we've seen
over the elapse.
            //We know the RDS signal runs at 1186 Hz.
            int numberOfBits = (int)elapsed * 1186;

            //Now we store the data for later decoding. An RDS signal
is 120 bits.
            //So we keep running until we have reached a maximum of 360
bits for error
            //detection and finding the beginning and end of the
signal.
            for (int i = 0; i < numberOfBits; i++)
                incomingData.Add(dataread);

            //If we have gotten all the data we need, close the port
and stop reading
            //and decode the data.
            if (incomingData.Count > 360)
            {
                serialPort.Close();
                DecodeData();
            }

            previous = currentTime;
        }

        //Decodes the RDS signal and updates the iCarTunes display
        string[] charMap = {"A", "B", "C", "D", "E", "F", "G", "H",
"I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V",
"W", "X", "Y", "Z",
                            "a", "b", "c", "d", "e", "f", "g", "h",
"i", "j", "k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v",
"w", "x", "y", "z",
                            "1", "2", "3", "4", "5", "6", "7", "8",
"9", "0", "/", " "};
        private void DecodeData()
        {
            //First thing we need to do is to find the start and stop
of the signal.
            //We know we are listening to x96, so we loop through all
the bits
            //and find the 1011 DIS bits.
            for (int i = 0; i < incomingData.Count; i++)
            {
                //If this is the DIS bits we found it. Crop the data so
we're just dealing with one signal
                if ((incomingData[i] == 1) && (incomingData[i+1] == 0)
&& (incomingData[i+2] == 1) && (incomingData[i+3] == 1))
                {
                    if ((incomingData[i + 120] == 1) && (incomingData[i
+ 121] == 0) && (incomingData[i + 122] == 1) && (incomingData[i + 123]
== 1))
                    {
                        //Also remove the first block and error code
                        incomingData.RemoveRange(0, i + 24);
                        incomingData.RemoveRange(103,
incomingData.Count - 1);
                    }
                }
            }


            //Now let's begin decoding
            string artist = "";
            string title = "";

            //X96s protocol has the artist name on bits on Blocks 2 and
3
            int lookup = (int)(incomingData[0] * Math.Pow(2, 6) +
incomingData[1] * Math.Pow(2, 5) +
                              incomingData[2] * Math.Pow(2, 4) +
incomingData[3] * Math.Pow(2, 3) +
                              incomingData[4] * Math.Pow(2, 2) +
incomingData[5] * Math.Pow(2, 1) +
                              incomingData[5] * Math.Pow(2, 0));
            //1st character, success
            artist = charMap[lookup];

            //The remaining bits are found in blocks 2 and 3, so l
            for(int i = 6; i < 32; i = i + 2)
            {
                lookup = decodeBinary(i, i + 1) + 26;
                artist = artist + charMap[lookup];
            }

            //Blocks 4 and 5 contain the title data
            //Let's clip everything to 0 so we don't have to worry
about it anymore
            incomingData.RemoveRange(0, 48);

            lookup = (int)(incomingData[0] * Math.Pow(2, 6) +
incomingData[1] * Math.Pow(2, 5) +
                              incomingData[2] * Math.Pow(2, 4) +
incomingData[3] * Math.Pow(2, 3) +
                              incomingData[4] * Math.Pow(2, 2) +
incomingData[5] * Math.Pow(2, 1) +
                              incomingData[5] * Math.Pow(2, 0));

            title = charMap[lookup];

            //The reaminging bits are looped through to decode the
artist song
            for (int i = 6; i < 44; i = i + 2)
            {
                lookup = decodeBinary(i, i + 1) + 26;
                artist = artist + charMap[lookup];
            }

            //Update the display with the decoded information
            songdisplayLabel.Text = artist + " - " + title;


        }

        //Returns the int value of the 2 digit binary sequence in rds
signals
        private int decodeBinary(int first, int second)
        {
            if (first == 0 && second == 0)
                return 0;
            if (first == 0 && second == 1)
                return 1;
            if (first == 1 && second == 0)
                return 2;
            else
                return 3;
        }
    }
}
```

Download Menu Source Code:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Collections;
using iTunesLib;

using System.IO;

namespace iCarTunes
{
    public partial class DownloadMenu : Form
    {
        //The list of songs.
        public List<Song> songs;
        //The list of downloaded songs
        public List<Song> downloads;
        //The itunes application interface.
        private iTunesApp itunes;

        //Initializes the gui with the list of songs
        public DownloadMenu(List<Song> songs, List<Song> downloads,
iTunesApp itunes)
        {
            InitializeComponent();

            this.itunes = itunes;
            this.songs = songs;
            this.downloads = downloads;
            //Add all the songs to the listbox
            foreach (Song s in songs)
            {
                songList.Items.Add(s);
            }
            if (songList.Items.Count > 0)
                songList.SelectedIndex = 0;
        }

        //Keeps track of which "button" is highlighted
        private int selection = 0;
        //Determines whether selecting buttons or a song from the list
        private bool inSelectMode = false;

        //This is executed any time the menu is clicked.
        private void DownloadMenu_Click(object sender, EventArgs e)
        {
            //If we are selecting a song from the listbox
            if (inSelectMode)
            {
                MouseEventArgs f = (MouseEventArgs)e;
                if (f.Button == MouseButtons.Left)
                {
                    if (songList.SelectedIndex == songList.Items.Count
- 1)
                        songList.SelectedIndex = 0;
                    else
                        songList.SelectedIndex = songList.SelectedIndex
+ 1;
                }
                else
                {
                    selectButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    inSelectMode = false;
                }

            }
            else
            {
                //If we are selecting a button
                MouseEventArgs f = (MouseEventArgs)e;
                if (f.Button == MouseButtons.Left)
                {
                    selection = selection + 1;
                    if (selection == 4)
                        selection = 0;

                    if (selection == 0)
                    {
                        selectButton.BackColor =
System.Drawing.Color.RoyalBlue;
                        downloadButton.BackColor =
System.Drawing.Color.White;
                        deleteButton.BackColor =
System.Drawing.Color.White;
                        exitButton.BackColor =
System.Drawing.Color.White;
                    }
                    if (selection == 1)
                    {
                        selectButton.BackColor =
System.Drawing.Color.White;
                        downloadButton.BackColor =
System.Drawing.Color.RoyalBlue;
                        deleteButton.BackColor =
System.Drawing.Color.White;
                        exitButton.BackColor =
System.Drawing.Color.White;
                    }
                    if (selection == 2)
                    {
                        selectButton.BackColor =
System.Drawing.Color.White;
                        downloadButton.BackColor =
System.Drawing.Color.White;
                        deleteButton.BackColor =
System.Drawing.Color.RoyalBlue;
```

```
                        exitButton.BackColor =
System.Drawing.Color.White;
                    }
                    if (selection == 3)
                    {
                        selectButton.BackColor =
System.Drawing.Color.White;
                        downloadButton.BackColor =
System.Drawing.Color.White;
                        deleteButton.BackColor =
System.Drawing.Color.White;
                        exitButton.BackColor =
System.Drawing.Color.RoyalBlue;
                    }
                }
                else
                {
                    //If a right click, call the callback for the
selected button
                    if (selection == 0)
                        selectButton_Click(null, null);
                    if (selection == 1)
                        downloadButton_Click(null, null);
                    if (selection == 2)
                        deleteButton_Click(null, null);
                    if (selection == 3)
                        exitButton_Click(null, null);
                }
            }
        }


        //Executes when the "Select Song" button is pressed
        //Disables button presses, and allows to scroll through songs
        private void selectButton_Click(object sender, EventArgs e)
        {
            //If there is nothing in the songlist, just return
            if (songList.Items.Count == 0)
                return;

            //Signify we are now in select mode and update display
            inSelectMode = true;
            selectButton.BackColor = System.Drawing.Color.Gray;
        }

        //Executes when the "Delete Song" button is pressed.
        //Removes the selected song from the GUI and the list of songs
        //and updates the display to highlight something else.
        private void deleteButton_Click(object sender, EventArgs e)
        {
            if (songList.SelectedIndex < 0)
                return;
            songs.RemoveAt(songList.SelectedIndex);
            int index = songList.SelectedIndex - 1;
            songList.Items.Remove(songList.SelectedItem);
            songList.SelectedIndex = index;
            if (songList.Items.Count > 0 && songList.SelectedIndex == -
1)
                songList.SelectedIndex = 0;

        }

        //Executes when the "Back" button is clicked, closes the form
        private void exitButton_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        //Executes when the "Download Song" button is pressed.
        private void downloadButton_Click(object sender, EventArgs e)
        {
            //Make the button grey while we do our work
            downloadButton.BackColor = System.Drawing.Color.Gray;

            //Reads the entire download script and saves away the
targetline we're after
            List<String> fileContents = new List<String>();
            int targetline = 0;
            int count = 0;
            StreamReader reader = new
StreamReader("downloadscript.au3");
            String currentLine = reader.ReadLine();
            while(currentLine != null)
            {
                //Check each line as they are read. Save away the line
number after the "marker" ;; is found
                if (currentLine.Length > 2)
                    if (currentLine.Substring(0, 2).Equals(";;"))
                        targetline = count + 1;

                fileContents.Add(currentLine);
                currentLine = reader.ReadLine();
                count = count + 1;
            }
            reader.Close();

            //Change the targetline to the search query for the song
we're looking up
            string foo = songList.SelectedItem.ToString();
            fileContents[targetline] = "Send(\"" +
songList.SelectedItem.ToString() + "\")";

            //Write the updated script back into the file
            StreamWriter writer = new
StreamWriter("downloadscript.au3");
            foreach (String line in fileContents)
            {
                writer.WriteLine(line);
            }
            writer.Close();

            //Now we can open iTunes, and run the script to begin
downloading
```

```
            itunes.GotoMusicStoreHomePage();
            System.Threading.Thread.Sleep(5000);
            System.Diagnostics.Process script =
System.Diagnostics.Process.Start("downloadscript.au3");

            //Run idle until the script finishes execution
            while (!script.HasExited)
            {
                System.Threading.Thread.Sleep(2000);
            }

            //If the download completed successfully
            if(script.ExitCode == 0)
            {
                //Save the song we just downloaded to the download list
                downloads.Add((Song)songList.SelectedItem);

                //It successfully downloaded
                //remove song from the list, and save to the download
list
                deleteButton_Click(null, null);
            }

            //If we had an error and the download did not complete
            //Keep song in the list

            //Return the button to original color
            downloadButton.BackColor = System.Drawing.Color.RoyalBlue;
        }

        //If a double click occurs, treat it like a single click
        private void DownloadMenu_DoubleClick(object sender, EventArgs
e)
        {
            DownloadMenu_Click(sender, e);
        }
        //If the listbox is clicked, just treat it like a normal click
        private void songList_Click(object sender, EventArgs e)
        {
            DownloadMenu_Click(sender, e);
        }
        //If a double click occurs on the listbox, just treat it like a
normal click
        private void songList_DoubleClick(object sender, EventArgs e)
        {
            DownloadMenu_Click(sender, e);
        }
    }
}
```

Appendix III: AutoIt Scripting Code for iTunes Downloads

```
;This script will download a song from the itunes store


;This code is used for debugging purposes, keep it here
;Sleep(3000)
;$pos = MouseGetPos()
;$color = PixelGetColor($pos[0], $pos[1])
;MsgBox(4, "Color at:", "Color = " & $color & " X,Y = " & $pos[0] & ", " & $pos[1])

BlockInput(1)
Sleep(3000)

;Click search clear button, then click the search box, then begin typing our search query
MouseClick("left", 1241, 41)
Sleep(500)
MouseClick("left", 1126, 42)
;;Marker
Send("Jack's Mannequin - Annie Use Your Telescope")
Send("{ENTER}")

;Wait 15 seconds for the search to complete, if it doesn't, Exit
$seconds = 0
Do
        Sleep(1000)
        $seconds = $seconds + 1
        if($seconds = 15) Then
                MsgBox(4, "Connection Timed Out", "The iTunes store is unable to
connect. Please try again later.", 5)
                ;Set the mouse back to the iCartunes App
                MouseMove(937, 571)
                WinSetState("iTunes", "", @SW_MINIMIZE)
                BlockInput(0)
                Exit 10
        EndIf
Until PixelGetColor(646, 54) = 4934212

Sleep(1000)
;Once our search is complete, 1 of 4 things can happen
;If there was no result found, exit
if(PixelGetColor(386, 144) = 16777215) Then
        MsgBox(4, "No Search Results Found", "The iTunes does not have your desired
song. Sorry!", 5)
        ;Set the mouse back to the iCartunes App
        MouseMove(937, 571)
```

```
		WinSetState("iTunes", "", @SW_MINIMIZE)
		BlockInput(0)
		Exit 10
	EndIf

;If there is just one line on blue bar, click the song in the right location
if(PixelGetColor(1062, 414) = 12829635) Then
	MouseClick("left", 826, 429)
Else
	;If there is a podcast bar on the bottom, click the right space
	if(PixelGetColor(1229, 562) = 13158600) Then
		MouseClick("left", 843, 579)
	Else
		;Else click the bottom button for downloads
		MouseClick("left", 843, 558)
	EndIf
EndIf

;Wait 20 seconds for things to get squared away, and exit erroroneously or exit
$seconds = 0
Do
	Sleep(1000)
	$seconds = $seconds + 1
	if($seconds = 20) Then
		MsgBox(4, "Connection Timed Out", "The iTunes store is unable to
connect. Please try again later.", 5)
		WinSetState("iTunes", "", @SW_MINIMIZE)
		BlockInput(0)
		Exit 10
	EndIf
Until PixelGetColor(639, 42) = 5197128

;Minimize iTunes, display success message, and let control be handed back to our C#
code
WinSetState("iTunes", "", @SW_MINIMIZE)
MsgBox(4, "Download Successful", "Your song is downloading.", 5)

;Set the mouse back to the iCartunes App
MouseMove(937, 571)
BlockInput(0)
```