Wilson: Homemade Voice Recognition Using C#

By Matthew Mansfield

We've all seen Ironman and have undoubtedly wondered how we could get our own Jarvis, right? Well, with today's operating systems which now include so many helpful features, files and smart code that you can build from to make your very own Jarvis; we are going to explore just how to make your own Jarvis. In this tutorial I will show you how you can have a voice recognition software program in your home or office.

Before we get going too far into this all I would like for you to understand a few things. This program we are about to write has very specific functions that I needed to fill which I could not find in any voice recognition software. Some of the specific features of this software:

- *Runs executable program files and scripts*
- *Provides audible feedback as to whether a task has been carried out*
- *Listens to words as they are spoken instead of waiting for the end of a paragraph*

If these features are what you are interested in then keep reading because this tutorial is for you.

This manual is assuming you have a basic to intermediate amount of experience working knowledge of the C# language as well as Microsoft Access. In this manual I will try my best to cover every step needed in the process but should a step or two be missing it's my hope that you will be able to fill in those gaps. Best of luck in your journey to creating your personalized voice recognition program!

Contents

Section 1 – Origins of Wilson

Section Objective

In this section we will cover:

- The Need
- The Name
- The Uses

The Need

In mid 2008 I was attempting to construct a robotic arm out of spare parts. Sometimes this process became rather laborious and I often found myself wishing I had an assistant. Many times I would be in the middle of soldering on a circuit board or stringing a wire from one place to another and need to run my code. When you run your code it causes electricity to flow through all of your components and if you have wired something incorrectly you may fry a servo or worse an entire microcontroller. As my need for having an assistant remained rather constant I began becoming curious as to how I could make an automated assistant. This curiosity brought me to seeking out the different voice recognition (VR) software solutions that were already on the market. After a lot of research, downloading and trials of software I realized that there were no programs that provided for me the kinds of features I was seeking. The features I needed in order keep a productive flow were to run an application upon voice request, receive voice confirmation that the given application had been run and accept single word commands if necessary.

The Name

Originally, when I first implemented the application in my office I was working on the voice codec training phase when I would often turn to find my wife standing there. She would ask me things like, "You do know it's not real, right?" Of course I knew it was just a computer but speaking to it in natural tones is the best way for it to understand your speech. Over time her visits to my office began growing jokes about how I talk to my computer. Often times when I'm working on a project I become immersed in it and I grow my beard out, forget to eat and sleep. One day, my wife came in and told me I looked like Tom Hanks from Cast Away and my computer was like my Wilson. From there the name stuck and my VR program had been coined as Wilson. The voice of the application has changed several times during the different phases of the process but usually the voice of the application is a British male which adds to the character of Wilson. Of course, you can name your application whatever you would like but since I'm so used to referring to the software solution as Wilson please don't be thrown for a loop if I call it simply Wilson.

The Uses

Once you get your voice recognition system up and running you quickly realize how many things you can attach it to. The original purpose for creating Wilson was to run some of my algorithms through the debugger while I worked on my robotics. Since that time, Wilson has grown leaps and bounds. Now, Wilson controls all of my media and is tied into my cloud server as well. I have also automated many of the electric devices in my office with Wilson and the most recent aspirations have been to tie in VoIP (Voice over Internet protocol) to Wilson.

Section 2 – Beginning Your Project

Section Objective

In this section we will cover:

- System Requirements
- A New Project
- Adding the Forms
- Importing the References
- Creating an Access Database

System Requirements

It has been my experience that this program functions best when certain conditions are met. Part of those conditions include that either Microsoft Windows Experience (XP) or Microsoft Server 2008 R2 be used. Also, I've noticed is in order to get the best DLL's in place you must install a few applications that have the DLL's you'll need. It's an older program, but if you install Microsoft Office 2003 it contains the Microsoft Speech module which will be inserted into your Control Panel. After installing MS Office 2003, you will be able to open your Control Panel, select "Speech" *(or* "Text to Speech" *depending on which OS you are using)* and adjust the voice settings to what you would like. Next, you will need to install Dragon Naturally Speaking Premium 11.0 which contains the remaining DLL's you would need to get things up and running. For this project it is desirable for you to have at least .Net 4 framework installed.

Side Step: I know what you may be thinking, "Why would I write my own program that requires me to have to install all of these other software programs, one of which is a VR program?" Great question and I like the way you are thinking. I've done a lot of research in VR and have gone through many software solutions in my search. One of my personal goals was to create a program that executed various tasks and provided me audible feedback once it was done completing that task. Dragon does not do this. Some of Dragon's applications do allow you to execute applications upon command but the voice codec that's utilized works in a funny way. Instead of Dragon executing what you speak immediately; it listens for a long time to "see" if you are done speaking. This is because Dragon listens to paragraphs of speech and that waiting makes your program lag or respond really slowly. I was never able to get Dragon to listen and execute my commands quickly and this created a lot of frustration. Finally, I was able to locate Tazti which is a wonderful program but currently only allows 75 custom commands. This causes a huge issue because you can ask for the same thing many different ways, for example:

"Computer, open My Documents"

"Computer, open My Documents please"

"Please open My Documents"

"Open My Documents"

"Open My Documents please"

"My Documents"

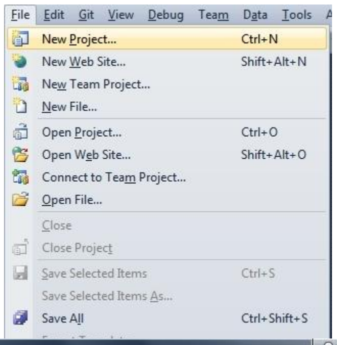
"My Documents please"

Needless to say 75 commands goes extremely quickly. This application we are about to write will allow you to have upwards of 16 million commands which should be plenty of room for growth.

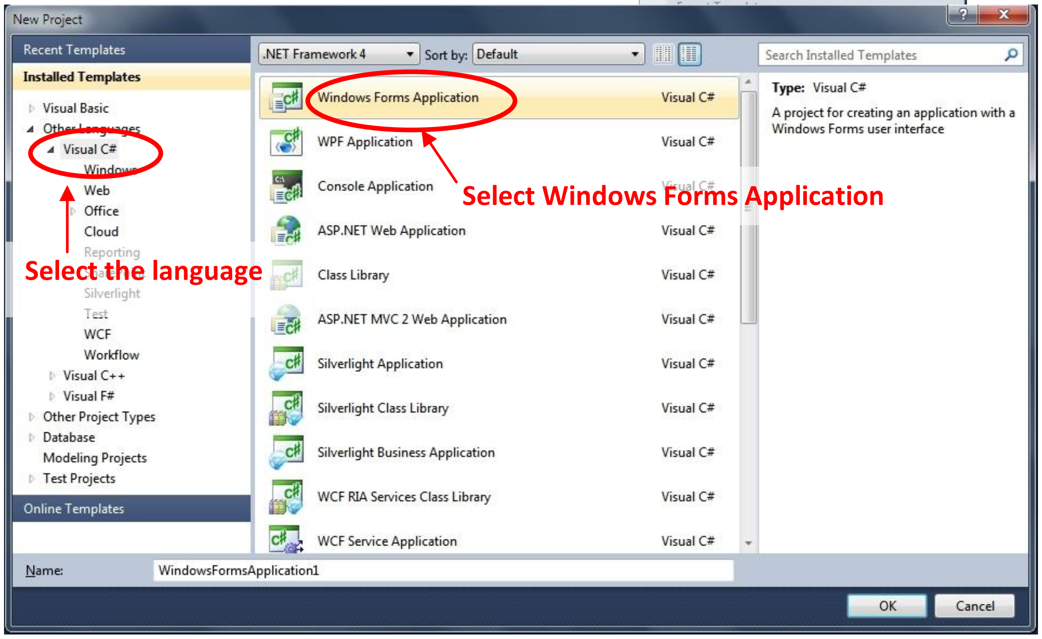
For a little bit of variety you can purchase some premium voices to add some more personality to your application and enhance the experience of speaking to your computer. Cepstral makes some really nice voices for an affordable rate. Also, some more nice voices can be found through NeoSpeech *(my favorite is Julie)*.

A new Project

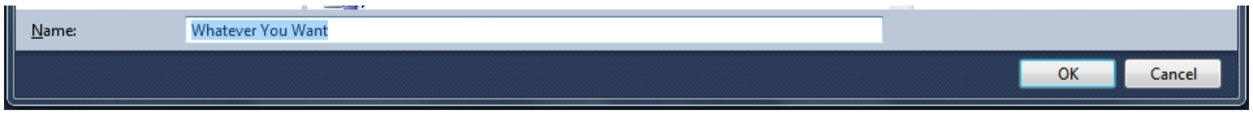
My favorite programming suite is Microsoft's Visual Studio 2010 Ultimate. For this tutorial I will be utilizing that as my application of choice. To begin let's start a new project by selecting from the File dropdown menu and clicking on New Project.


File Edit Git View Debug Team Data Tools
New Project... Ctrl+N
New Web Site... Shift+Alt+N
New Team Project...
New File...
Open Project... Ctrl+O
Open Web Site... Shift+Alt+O
Connect to Team Project...
Open File...
Close
Close Project
Save Selected Items Ctrl+S
Save Selected Items As...
Save All Ctrl+Shift+S

Next, we'll select the language we want to program in. We will be using C# for our purposes. Then we will name the project whatever you want.


New Project
Recent Templates
Installed Templates
Visual Basic
Other Languages
Visual C#
Windows
Web
Office
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Visual C++
Visual F#
Other Project Types
Database
Modeling Projects
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
ASP.NET Web Application Visual C#
Class Library Visual C#
ASP.NET MVC 2 Web Application Visual C#
Silverlight Application Visual C#
Silverlight Class Library Visual C#
Silverlight Business Application Visual C#
WCF RIA Services Class Library Visual C#
WCF Service Application Visual C#
Type: Visual C#
A project for creating an application with a Windows Forms user interface
Select Windows Forms Application
Select the language
OK
Cancel

Name: WindowsFormsApplication1



Name: Whatever You Want

Once you click OK, you will see your new form ready for use.

Adding the Forms

So, you should be looking at your form which defaults to the name Form1. You can name your project whatever you'd like but I will be just leaving the name as is. Next, open your Solution Explorer and right-click your project go to Add>New Item and select Windows Form. Again, name this form what you would like but I will be calling mine Form4 *(the default should be Form2.cs).*

Importing the References

Here I will simply provide you with a list of imports to add to your project. To import a reference you must open Solution Explorer, right-click on References then Add Reference...>COM tab and select the reference you would like to have included in your project and then click on OK. The references you will need before the project is completed are:

- **ADODB**
- **DAO**
- **Interop.SpeechLib**
- **LumiSoft.Net**
- Microsoft.CSharp
- **Microsoft.Office.Core**
- **Microsoft.Office.Interop.Access**
- **Microsoft.VisualBasic**
- **Microsoft.VisualBasic.PowerPacks.Vs**
- **MouseKeyboardLibrary**
- System
- System.Core
- System.Data
- System.Data.DataSetExtensions
- System.Deployment
- **System.Design**
- System.Drawing
- **System.Speech**
- System.Windows.Forms
- System.Xml
- System.Xml.Linq
- **Vallelunga.HomeAutomation**
- **VBIDE**

The ones in the list above which are marked in red are references that are not present when you first open your new project. Many of these references to direct link library (DLL) files will be added in at different stages of your writing process of your application. Automatically adding references is a feature of Visual Studio. However, if for some reason you are missing or cannot find one of the DLL's that's not normally added as you continue to create your program you may download the additional DLL's HERE and then be sure they are all there before you are finished. That link is to a zip file which I have provided for you to easily download and use as needed.

Creating an Access Database

Here is the part where basic to intermediate experience with MS Access comes in handy because we will not go into every detail on this process. I generally use MS Access 2007 so my instruction will be geared toward that version. To begin, select the Windows icon in the upper left then click on New. Name the database whatever you would like; for this tutorial I will be naming mine VR.accdb. Create a database that your project can use. I named mine CustomCommands and I included the following fields:

- ID
- CommonField
- Command
- Result

Save that database and so we can use it during the next step.

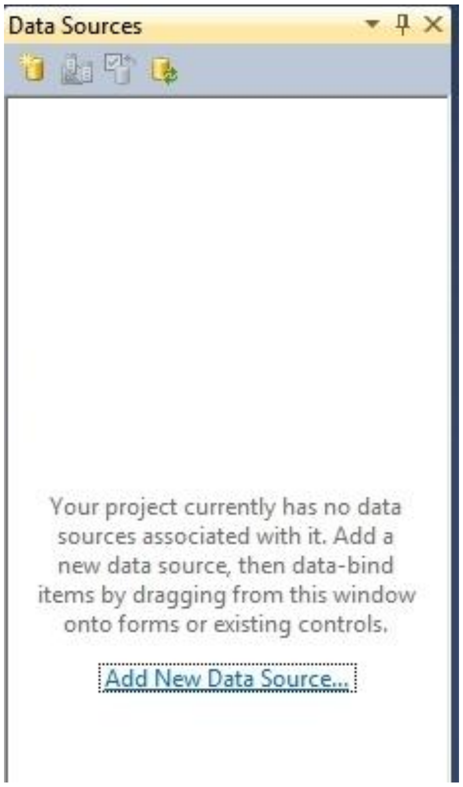
Section 3 – Connecting the Pieces

Section Objective

In this section we will cover:

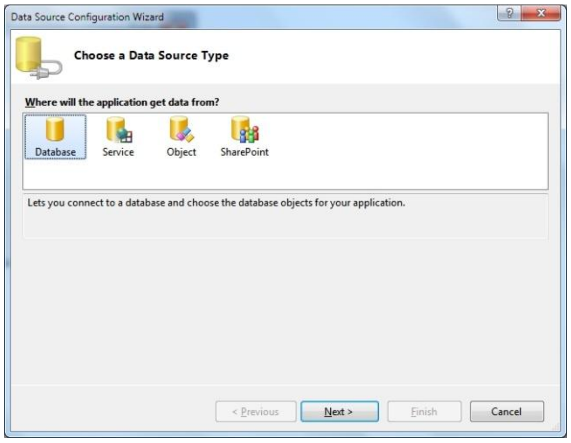
- Connecting the Database to Your Form
- Creating Data Grids
- Creating Text Boxes and Buttons

Connecting the Database to Your Form



Your project currently has no data sources associated with it. Add a new data source, then data-bind items by dragging from this window onto forms or existing controls.

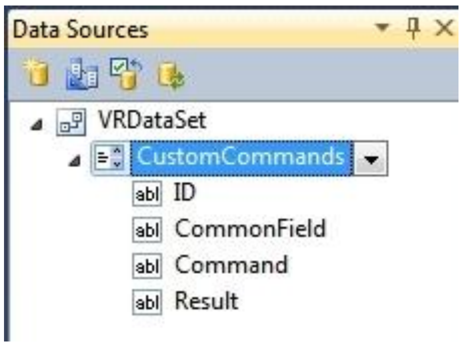
Thanks to MS Visual Studio you can drag and drop a lot of things and then it will add in a lot of the connection strings for you. Let's go ahead and connect the database to your forms now. To begin, go to Data Sources and Add New Data Source... this will open a Data Source Configuration Wizard which will walk you through the steps to add your database connection.



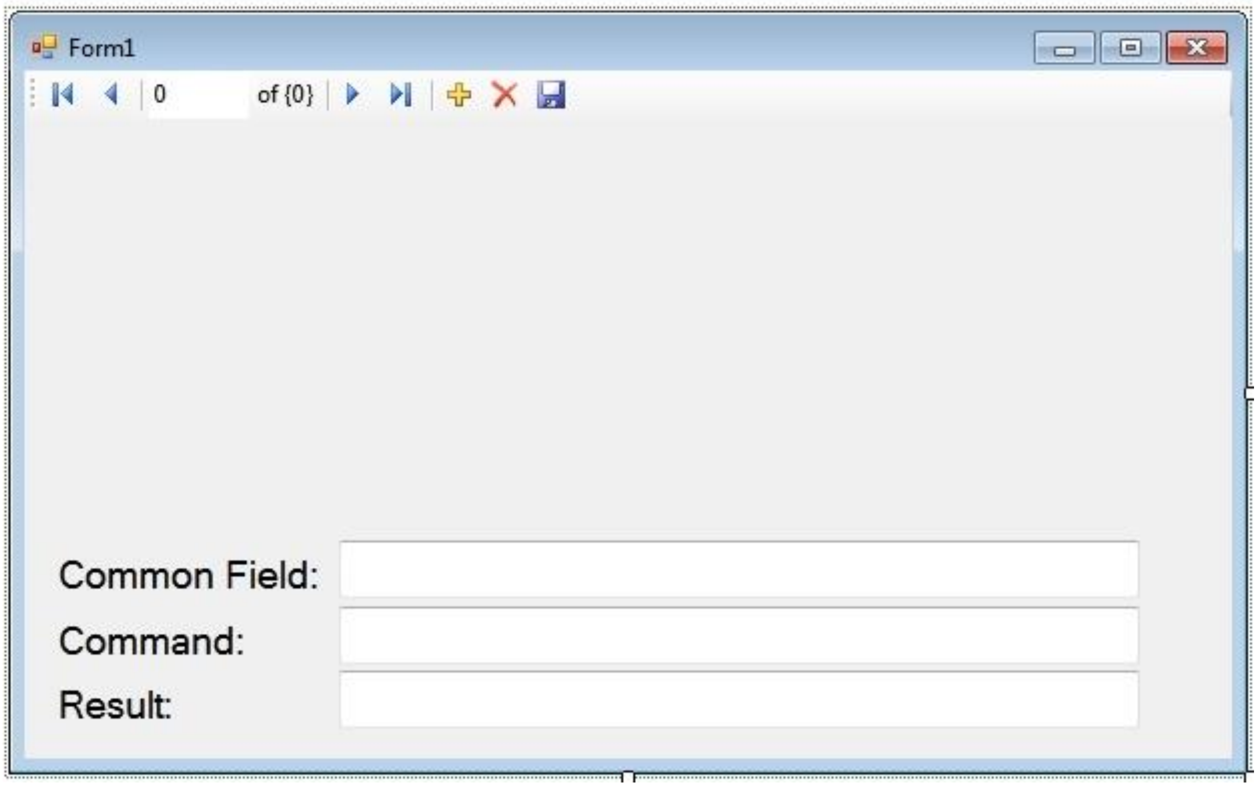
Where will the application get data from?

Lets you connect to a database and choose the database objects for your application.

After you have connected your program to the MS Access database you have created we will add that database to our program forms. You will see that you now have your DataSet under your Data Sources.



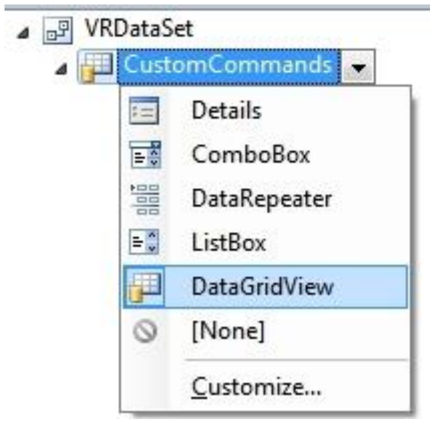
This is where you can click and hold the given field such as "Command" or "Result" and drag them onto your forms. Just make sure the field is set to TextBox and you should have a form that looks something like this:



The "Common Field" field is what the computer will speak to you, the "Command" field is what you would speak to the computer and "Result" field is the program that would be executed. To help me mentally keep these straight I will rename my labels to that effect. We just walked through connecting a database to only one of our forms. You can follow the appropriate steps listed in this section to connect this same database to your second form.

Creating Data Grids

On our main form, we will not need a data grid view, however, on our second form we will. Having the grid is not entirely necessary for the operation of the program but it does help you keep things organized as you are managing your commands. Under the Data Sources explorer select CustomCommands and from the dropdown menu select DataGridView. Then simply grab CustomCommands with your mouse and drag it onto your form. You can arrange your data grid view however you would like from there.



Creating Text Boxes and Buttons

Now, we can create all of the buttons and text boxes we will need on our forms. We'll just do this in steps to keep things simple.

For Form1 do the following:

Step 1 - Rename the following labels:

- "Command" field will now be "Your Spoken Command:"
- "Result" field will now be "Program to Launch:"
- "Common Field" field will now be "What the PC Speaks"

Step 2 - Add button1 and change the text to "Close this window"
Step 3 - Add button2 and change the text to "Open Search Box"
Step 4 - Add groupBox1 and remove text
Step 5 - Add radioButton1 inside of groupBox1 and change the text to "Website"
Step 6 - Add radioButton2 inside of groupBox1 and change the text to "Program"
Step 7 - Add button3 and change the name to btnSave and change the text to "Save"
Step 8 - Add button4 and change the name to btnOpen and change the text to "Open"
Step 9 - Add button5 and change the name to btnToListbox and change the text to "to listbox"

For Form4 do the following:

Step 1 - Add button1 and name it button4
Step 2 - Add text box and call it textBox1
Step 3 - Drag CustomCommands into this form so you have all of the textboxes from your database. Rename the labels so you don't forget which text box is linked to which thing *(this will help you keep things organized).*

Section 4 – Hammering out the Code

Section Objective

In this section we will cover:

- Code for Form1
- Code for Form4

Code for Form1

```
/*****************************************************************************
*      Author:      Matthew Mansfield                                        *
*      Date:         06/13/2013                                              *
*      Description: The purpose of this C# programming is to create a voice *
*                   recognition application that can provide text to speech *
*                   responses to speech to text commands.                   *
*****************************************************************************
*/
//---------------------------------------------------------------------------
//Using directives - This section lists the namespaces that the application
//will be using frequently, and saves the programmer from specifying a fully
//qualified name every time that a method that is contained within is used.
//---------------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Speech.Synthesis;
using System.Speech.Recognition;
//---------------------------------------------------------------------------
//Declare namespace
//---------------------------------------------------------------------------
namespace DATABASING
{
    //-----------------------------------------------------------------------
    //Begin the class
    //-----------------------------------------------------------------------
    public partial class Form1 : Form
    {
        //-------------------------------------------------------------------
        //Do this to initialize
        //-------------------------------------------------------------------
        public Form1()
        {
            InitializeComponent();
        }
        //-------------------------------------------------------------------
        //For our purposes, we are not using this but this is basic
        //navigator binding
        //-------------------------------------------------------------------
        private void customCommandsBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
        {
            this.Validate();
            this.customCommandsBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.vRDataSet);
        }
        //--------------------------------------------------------------------
        //Load the basic database information, set focus to the given textbox
        //as well as running the FillingLists_method method
        //--------------------------------------------------------------------
        private void Form1_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the
            //'vRDataSet.CustomCommands' table. You can move, or remove it,
            //as needed.
            this.customCommandsTableAdapter.Fill(this.vRDataSet.CustomCommands);
            this.customCommandsBindingSource.AddNew();
            this.commonFieldTextBox.Focus();
            FillingLists_method();
        }
        //--------------------------------------------------------------------
        //When a command is given we need to find out where it is "located"
        //in the database. This helps us to find the location of the database
        //entry
        //--------------------------------------------------------------------
        private void textBox3_TextChanged_1(object sender, EventArgs e)
        {
            int loc = customCommandsBindingSource.Find("Result", this.textBox3.Text);
            customCommandsBindingSource.Position = loc;
        }
        //--------------------------------------------------------------------
        //Close this Form when this button is pressed
        //--------------------------------------------------------------------
        private void button1_Click(object sender, EventArgs e)
        {
            this.Close();
        }
        //--------------------------------------------------------------------
        //When we leave this Form, do this
        //--------------------------------------------------------------------
        private void Form2_Leave(object sender, EventArgs e)
        {
            this.Close();
        }
        //--------------------------------------------------------------------
        //Establishing the get set
        //--------------------------------------------------------------------
        public bool visible { get; set; }
        //--------------------------------------------------------------------
        //Main method we run to take results from what the user searches for
        //and places it into the result textbox
        //--------------------------------------------------------------------
        public void runcode_method()
        {
            if (this.resultTextBox1.Text == "")
            {
            }
            else
            {
                System.Diagnostics.Process p = new System.Diagnostics.Process();
                p.StartInfo.FileName = resultTextBox1.Text;
                p.Start();
                //sleep for 1500ms
                System.Threading.Thread.Sleep(1500);
                //clear all textboxes
                this.commandTextBox.Text = "";
                this.resultTextBox.Text = "";
                this.txtProfileName.Text = "";
                this.resultTextBox1.Text = "";
                //set focus here
                this.commandTextBox.Focus();
            }
        }
        //-------------------------------------------------------------------
        //When a command is given we need to find out where it is "located"
        //in the database. This helps us to find the location of the database
        //entry and set the position of the binding source to it
        //(the next few methods are the same)
        //-------------------------------------------------------------------
        public void find_method()
        {
            int loc = customCommandsBindingSource.Find("Command",
this.toolStripTextBox1.Text);
            customCommandsBindingSource.Position = loc;
        }
        //-------------------------------------------------------------------
        //When a command is given we need to find out where it is "located"
        //in the database. This helps us to find the location of the database
        //entry and set the position of the binding source to it
        //-------------------------------------------------------------------
        public void checkingValue_method()
        {
            int loc = customCommandsBindingSource.Find("Command",
this.commandTextBox.Text);
            customCommandsBindingSource.Position = loc;
        }
        //-------------------------------------------------------------------
        //When a command is given we need to find out where it is "located"
        //in the database. This helps us to find the location of the database
        //entry and set the position of the binding source to it
        //-------------------------------------------------------------------
        private void toolStripButton1_Click(object sender, EventArgs e)
        {
            int loc = customCommandsBindingSource.Find("Command",
this.toolStripTextBox1.Text);
            customCommandsBindingSource.Position = loc;
        }
        //-------------------------------------------------------------------
        //Saves the database and then moves to the next available field
        //in the database
        //-------------------------------------------------------------------
        private void btnSave_Click(object sender, EventArgs e)
        {
            this.customCommandsBindingNavigatorSaveItem.PerformClick();
            this.customCommandsBindingSource.AddNew();
            this.customCommandsBindingSource.MoveLast();
            this.toolStripTextBox1.Text = "";
            this.btnOpen.Visible = false;
            this.radioButton1.Checked = false;
            this.radioButton2.Checked = false;
            this.commandTextBox.Focus();
        }
        //--------------------------------------------------------------------
        //If the user wants the system to open a webpage they can select this
        //then this text will be inserted to the correct textbox
        //--------------------------------------------------------------------
        private void radioButton1_CheckedChanged(object sender, EventArgs e)
        {
            if (this.radioButton1.Checked == true)
            {
                this.btnOpen.Visible = false;
                this.resultTextBox.Text = "http://www.";
                this.resultTextBox.Focus();
            }
        }
        //--------------------------------------------------------------------
        //Run a quick check for the radio buttons
        //--------------------------------------------------------------------
        private void radioButton2_CheckedChanged(object sender, EventArgs e)
        {
            if (this.radioButton2.Checked == true)
            {
                this.btnOpen.Visible = true;
                this.resultTextBox.Text = "";
            }
        }
        //--------------------------------------------------------------------
        //Open up a dialog box if the user selects to open a program. This is
        //set to default to the C:\ drive
        //--------------------------------------------------------------------
        private void btnOpen_Click(object sender, EventArgs e)
        {
            if (openFileDialog1.ShowDialog() == System.Windows.Forms.DialogResult.OK)
                {
                    System.IO.StreamReader sr = new
                    System.IO.StreamReader(openFileDialog1.FileName);
                    this.openFileDialog1.InitialDirectory = @"C:\";
                    sr.Close();
                    //Place the results to this textbox
                    this.resultTextBox.Text = (openFileDialog1.FileName.ToString());
                }
        }
        //--------------------------------------------------------------------
        //If the user is attempting to place a duplicate entry for something
        //the system will create a message that pops up and lets them know.
        //This is to save you heart ache in the future. No person can serve
        //to masters and in the same no command can server two functions ;)
        //--------------------------------------------------------------------
        private void commandTextBox_Leave(object sender, EventArgs e)
        {
            int loc = customCommandsBindingSource.Find("Command",
this.commandTextBox.Text);
            if (loc != -1)
            {
                MessageBox.Show("This is a duplicate entry, please try another
word/phrase.", "Data Entry Error");
                this.commandTextBox.Text = "";
                this.commandTextBox.Focus();
            }
        }
        //---------------------------------------------------------------------
        //Reference to this method
        //---------------------------------------------------------------------
        private void btnToListbox_Click(object sender, EventArgs e)
        {
            FillingLists_method();
        }
        //---------------------------------------------------------------------
        //Pretty basic method here
        //---------------------------------------------------------------------
        private void FillingLists_method()
        {
            this.listBox1.DataSource = customCommandsBindingSource;
            this.listBox1.DisplayMember = "Command";
            this.listBox2.DataSource = customCommandsBindingSource;
            this.listBox2.DisplayMember = "Result";
            this.listBox3.DataSource = customCommandsBindingSource;
            this.listBox3.DisplayMember = "CommonField";
        }
        //---------------------------------------------------------------------
        //Create an instance of the SpeechSynthesizer named "synth"
        //---------------------------------------------------------------------
        SpeechSynthesizer synth = new SpeechSynthesizer();
        //---------------------------------------------------------------------
        //Create an instance of the SpeechRecognitionEngine called "sre"
        //---------------------------------------------------------------------
        SpeechRecognitionEngine sre = new SpeechRecognitionEngine();
        //---------------------------------------------------------------------
        //Create Speech Timer
        //---------------------------------------------------------------------
        Timer speechTimer = new Timer();
        //---------------------------------------------------------------------
        //Speak: Takes in a string. Then outputs the string as Speech
        //---------------------------------------------------------------------
        public void Speak(string text)
        {
            //Speak text
            synth.Speak(text);
        }
        //---------------------------------------------------------------------
        //Set the SpeechRate of the Speaker
        //---------------------------------------------------------------------
        public int SpeechRate
        {
            //Value range is -10 to 10
            //Default is 0
            get { return synth.Rate; }
            set { synth.Rate = value; }
        }
        //---------------------------------------------------------------------
        //Starts instance of speech recognition
        //--------------------------------------------------------------------
        public void StartDefaultRecognition()
        {
            string[] words = new string[this.listBox1.Items.Count];
            StartCustomRecognition(words);
        }
        //----------------------------------------------------------------
        //Create an instance of the SpeechSynthesizer named "synth"
        //----------------------------------------------------------------
        private void sendingtoarray_method()
        {
            string[] words = new string[this.listBox1.Items.Count];
        }
        //--------------------------------------------------------------------
        //Called to stop Speech Engine
        //--------------------------------------------------------------------
        public void StopRecognition()
        {
            //Stop Speech Recognition Engine
            sre.RecognizeAsyncStop();
        }
        //--------------------------------------------------------------------
        //Called to setup new grammer
        //--------------------------------------------------------------------
        public void StartCustomRecognition(String[] words)
        {
            //Stop any running instances
            StopRecognition();
            //Set input to default audio device
            sre.SetInputToDefaultAudioDevice();
            //Create a grammer builder instance called grammerBuilder
            GrammarBuilder grammarBuilder = new GrammarBuilder();
            grammarBuilder.Append(new Choices(words));
            //Create a grammer instance called customGrammer that uses the grammer
            //created from GrammerBuilder called "grammerBuilder"
            Grammar customGrammar = new Grammar(grammarBuilder);
            //Unload past grammer and set new grammer
            sre.UnloadAllGrammars();
            sre.LoadGrammar(customGrammar);
            //Create SpeechRecognized Event
            sre.SpeechRecognized += new
EventHandler<SpeechRecognizedEventArgs>(sre_Listen);
        }
        //--------------------------------------------------------------------
        //Called to listen for one command
        //--------------------------------------------------------------------
        public void StartListening()
        {
            System.Media.SoundPlayer player = new System.Media.SoundPlayer();
            player.SoundLocation = @"audio/b1.wav";
            player.Play();
            sre.RecognizeAsync(RecognizeMode.Multiple);
            speechTimer.Tick += new EventHandler(speechTimer_Tick);
            speechTimer.Interval = 5000;
            speechTimer.Enabled = true;
            speechTimer.Start();
        }
        //--------------------------------------------------------------------
        //Stop Listening after set time
        //-----------------------------------------------------------------
        void speechTimer_Tick(object sender, EventArgs e)
        {
            StopRecognition();
            speechTimer.Enabled = false;
            speechTimer.Stop();
            System.Media.SoundPlayer player = new System.Media.SoundPlayer();
            player.SoundLocation = @"audio/b1.wav";
            player.Play();
        }
        //-----------------------------------------------------------------
        //Called when word is detected. Passes action onto SpeechList Class
        //-----------------------------------------------------------------
        void sre_Listen(object sender, SpeechRecognizedEventArgs e)
        {
            //Stop Listening
            StopRecognition();
            //Stop Timer
            speechTimer.Enabled = false;
            speechTimer.Stop();
        }
    }
}
```

Code for Form4

```
/*****************************************************************************
*      Author:      Matthew Mansfield                                        *
*      Date:         06/13/2013                                              *
*      Description: The purpose of this C# programming is to create a voice *
*                   recognition application that can provide text to speech *
*                   responses to speech to text commands.                    *
******************************************************************************
*/
//---------------------------------------------------------------------------
//Using directives - This section lists the namespaces that the application
//will be using frequently, and saves the programmer from specifying a fully
//qualified name every time that a method that is contained within is used.
//---------------------------------------------------------------------------
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Threading;
using SpeechLib;
using MouseKeyboardLibrary;
using System.Speech.Synthesis;
using System.Speech.Recognition;
using System.Collections;
//---------------------------------------------------------------------------
//Declare namespace
//---------------------------------------------------------------------------
```

```
namespace DATABASING
{
    //----------------------------------------------------------------------
    //Begin the class
    //----------------------------------------------------------------------
    public partial class Form4 : Form
    {
        //------------------------------------------------------------------
        //Create an instance of the SpeechSynthesizer named "synth"
        //------------------------------------------------------------------
        SpeechSynthesizer synth = new SpeechSynthesizer();
        //------------------------------------------------------------------
        //Create an instance of the SpeechRecognitionEngine called "sre"
        //------------------------------------------------------------------
        SpeechRecognitionEngine sre = new SpeechRecognitionEngine();
        //------------------------------------------------------------------
        //SAPI Object Classes
        // - Declaring SAPI Application Object Classes
        //------------------------------------------------------------------
        private SpeechLib.SpSharedRecoContext objRecoContext;
        private SpeechLib.ISpeechRecoGrammar grammar;
        //------------------------------------------------------------------
        //Assigning string as variable
        //------------------------------------------------------------------
        private string strData = "No recording yet";
        //------------------------------------------------------------------
        //Do this to initialize
        //------------------------------------------------------------------
        public Form4()
        {
            InitializeComponent();
        }
        //------------------------------------------------------------------
        //***** This is hidden because the Form is too small to fit it ******
        //Should this button be pressed run our Try-Catch
        //------------------------------------------------------------------
        private void button1_Click(object sender, EventArgs e)
        {
            try
            {
                if (objRecoContext == null)
                {
                    objRecoContext = new SpeechLib.SpSharedRecoContext();
                    objRecoContext.Recognition += new
_ISpeechRecoContextEvents_RecognitionEventHandler(RecoContext_Recognition);
                    grammar = objRecoContext.CreateGrammar(1);
                    grammar.DictationLoad("", SpeechLoadOption.SLOStatic);
                }
                grammar.DictationSetState(SpeechRuleState.SGDSActive);
            }
            catch (Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("Exception caught when initializing
SAPI." + " This application may not run correctly.\r\n\r\n" + ex.ToString(), "Error");
            }
        }
        //------------------------------------------------------------------
        //***** This is hidden because the Form is too small to fit it ******
        //------------------------------------------------------------------
        private void button2_Click(object sender, EventArgs e)
        {
            grammar.DictationSetState(SpeechRuleState.SGDSInactive);
        }
        public void RecoContext_Recognition(int StreamNumber, object StreamPosition,
SpeechRecognitionType RecognitionType, ISpeechRecoResult Result)
        {
            strData = Result.PhraseInfo.GetText(0, -1, true);
            Debug.WriteLine("Recognition: " + strData + ", " + StreamNumber + ", " +
StreamPosition);
            textBox1.Text = strData; //strData = strData;
        }
        //------------------------------------------------------------------
        //***** This is hidden because the Form is too small to fit it ******
        //This was used to save as a wav file but the function has not been
        //used since the early days of this program. This code used to work
        //but a lot of changes have been made and it may not work anymore.
        //You are welcome to play with it if you find you may use some of it.
        //------------------------------------------------------------------
        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                SpeechVoiceSpeakFlags SpFlags = SpeechVoiceSpeakFlags.SVSFlagsAsync;
                SpVoice Voice = new SpVoice();
                if (checkBox1.Checked)
                {
                    SaveFileDialog sfd = new SaveFileDialog();
                    sfd.Filter = "All files (*.*)|*.*|wav files (*.wav)|*.wav";
                    sfd.Title = "Save to a wave file";
                    sfd.FilterIndex = 2;
                    sfd.RestoreDirectory = true;
                    if (sfd.ShowDialog() == DialogResult.OK)
                    {
                        SpeechStreamFileMode SpFileMode =
SpeechStreamFileMode.SSFMCreateForWrite;
                        SpFileStream SpFileStream = new SpFileStream();
                        SpFileStream.Open(sfd.FileName, SpFileMode, false);
Voice.AudioOutputStream = SpFileStream;
                        Voice.Speak("You recorded the following message " + strData,
SpFlags);
                        Voice.WaitUntilDone(Timeout.Infinite);
                        SpFileStream.Close();
                    }
                }
                else
                {
                    Voice.Speak("You recorded the following message" + strData, SpFlags);
                }
            }
            catch
            {
                MessageBox.Show("Speak error", "SimpleTTS", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
        //------------------------------------------------------------------
        //For our purposes, we are not using this but this is basic
        //navigator binding
        //-----------------------------------------------------------------
        private void customCommandsBindingNavigatorSaveItem_Click(object sender,
EventArgs e)
        {
            this.Validate();
            this.customCommandsBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.vRDataSet);
        }
        //-----------------------------------------------------------------
        //This loads the "connection" to the database and populates the table
        //-----------------------------------------------------------------
        private void Form4_Load(object sender, EventArgs e)
        {
            // TODO: This line of code loads data into the 'vRDataSet.CustomCommands'
table. You can move, or remove it, as needed.
            this.customCommandsTableAdapter.Fill(this.vRDataSet.CustomCommands);
            button1_Click(sender, e);
        }
        //-----------------------------------------------------------------
        //This is the "D.B." button
        //-----------------------------------------------------------------
        private void button4_Click(object sender, EventArgs e)
        {
            Form1 form = new Form1();
            form.Show();
        }
        //-----------------------------------------------------------------
        //When a change happens, keep the focus
        //-----------------------------------------------------------------
        private void textBox1_TextChanged(object sender, EventArgs e)
        {
            this.textBox1.Focus();
            find_method();
        }
        //-----------------------------------------------------------------
        //This searches for the matching command in the database
        //-----------------------------------------------------------------
        public void find_method()
        {
            int loc = customCommandsBindingSource.Find("Command", this.textBox1.Text);
            customCommandsBindingSource.Position = loc;
        }
        //-----------------------------------------------------------------
        //When the text is changed in the "resultant" text box run this
        //method. This is what actually makes the computer speak anything to
        //you.
        //-----------------------------------------------------------------
        private void resultTextBox_TextChanged(object sender, EventArgs e)
        {
            runcode_method();
        }
        //-----------------------------------------------------------------
        //This method speaks the matching text, pauses and then resets the
        //input textbox by setting it to ""
        //-----------------------------------------------------------------
        public void runcode_method()
        {
            if (this.resultTextBox.Text == "")
            {
            }
            else
            {
                System.Diagnostics.Process p = new System.Diagnostics.Process();
                // p.StartInfo.WorkingDirectory = Path.GetDirectoryName(path);
                p.StartInfo.FileName = resultTextBox.Text;
                p.Start();

                System.Threading.Thread.Sleep(1500);

                this.textBox1.Text = "";
                this.resultTextBox.Text = "";
                this.textBox1.Focus();
            }
        }
        //--------------------------------------------------------------------
        //A quick and dirty conversion of text to speech
        //--------------------------------------------------------------------
        void speakbox()
        {
            Convert.ToString(commonFieldTextBox.Text);
            Speak(commonFieldTextBox.Text);
        }
        //--------------------------------------------------------------------
        //Take string text and turn it into a speech using the "synth"
        //that was initialized earlier using SpeechSynthesizer
        //--------------------------------------------------------------------
        public void Speak(string text)
        {
            synth.Speak(text);
        }
        //--------------------------------------------------------------------
        //This uses the speakbox method to take the data in the
        //commonFieldTextbox and turn it to speech
        //--------------------------------------------------------------------
        private void commonFieldTextBox_TextChanged_1(object sender, EventArgs e)
        {
            speakbox();
        }
        //--------------------------------------------------------------------
        //When text is changed in this textbox then run this method
        //--------------------------------------------------------------------
        private void resultTextBox_TextChanged_1(object sender, EventArgs e)
        {
            runcode_method();
        }
        //--------------------------------------------------------------------
        //This button is not visible but still can be accessed if needed.
        //This simply runs a batch file.
        //--------------------------------------------------------------------
        private void button5_Click(object sender, EventArgs e)
        {
            System.Diagnostics.Process.Start("D:\\VR_MAIN\\VR1\\DATABASING current 
version\\DATABASING\\OpenDB.bat");
        }
```

```
    }
}
```

Section 5 – Extra Features and Functions

Section Objective

In this section we will cover:

- More Ideas
- Batch Files
- VBScripts
- Automating Appliances
- Conclusion

More Ideas

At this point your program should function properly. We've covered all aspects of the program, right down to the code. Of course, what you do with your new VR program from here is up to you but perhaps I could make a few suggestions that may help you as you explore the many fun things that could enhance the Jarvis experiences. In the sections to follow, I will not be explaining how to do each of these things rather I will be simply giving you some ideas from where you can build on.

Batch Files

If you are not familiar with batch files, they are MS-DOS based and were introduced by Windows in 1985 and still are very effective. Using your new VR program you can execute batch files which can do a lot to assist in automating tasks. For instance, if you would like your VR program to send you an email of something you can write a batch file to do this with the assistance of BLAT for Windows.

VBScripts

VBScript *(Visual Basic Scripting Edition)* is a scripting language that Microsoft developed and it has some very useful things for your VR application. For instance, you can create a VBScript easily by creating a text file and saving the following text in it:

```
Set objVoice = CreateObject("SAPI.SpVoice")
objVoice.Speak (FormatDateTime(Date(), 1))
```

If you change the file extension from .txt to .vbs you will now have a file that when clicked on will tell you the current date. Consequently, the following will tell you the time:

```
Set objVoice = CreateObject("SAPI.SpVoice")
objVoice.Speak (Time())
```

Using these scripts can bring some really cool elements to your application and can help to get your creative juices flowing. The beauty of VBScripts is they are incredibly easy to create, the coding is simple and yet they are extremely powerful when run. Also, with your new VR application, when you set the application to be launched as a .vbs file it will launch just like an executable.

Automating Applicances

Just because you have a VR program doesn't mean your lights will automatically be turning on/off at the sound of your voice. You need to connect controls to your devices and in turn connect those controls to your VR application. To control an application, such as a light, you must control the electricity going to it. As a suggestion, I would recommend doing research on Parallax Microcontroller's which will allow you to control the switch to your device (or light). The language that is used for Parallax's Microcontrollers is called Basic Stamp and with a bit of research you can have all of your lights connected to your VR program and functioning at the sound of your voice.

Conclusion

While I was programming mine for the first time I had so many questions and often times I would just wish I could see the code of a program that's actually working. I hope that this tutorial provided you with answers to many of your questions and I hope you have enjoyed this walk through of how to create your very own VR program in C#. Should you have any additional questions please feel free to contact me using my website matthewmansfield.me.