# Quick Start Guide

Version 3.0

# Introduction and Requirements

This document is a Quick Start Guide for the .NET Telephony Tool, Voice Elements. For complete documentation on the Telephony API, please refer to http://help.voiceelements.com. For general support information, please refer to http://support.inventivelabs.com.

## Product Introduction

Voice Elements is a software tool that brings telephony to the wider development community of Microsoft .NET. All of the VoIP technology is built into Voice Elements so there is no need for other 3rd party drivers. The tool enables .NET developers to develop IVRs, conference bridges, dialers, call centers, gateways or any telephony application. Supporting all .NET languages, such as VB.NET and C#, Voice Elements includes pre-made voice application modules, sample code tutorials, and reporting features such as call monitoring and logging.

The standard Voice Elements toolkit opens up the .NET architecture for telephony. A typical Visual Studio developer can easily learn the Voice Elements classes and create voice applications. Connection to all of the voice resources that a .NET application requires can be accomplished with the Voice Elements Platform or the Telephony Bank.

## Technical Brief

The Voice Elements Technology Brief provides a useful overview of the Voice Elements Platform and Telephony Bank architecture, including a Call Flow Sequence Diagram and Class Diagram. It can be found at http://support.inventivelabs.com/index.php?title=Class_Diagram.

## Operating System Requirements

The following are the requirements for the OS and for Visual Studio:

Client OS requirements:

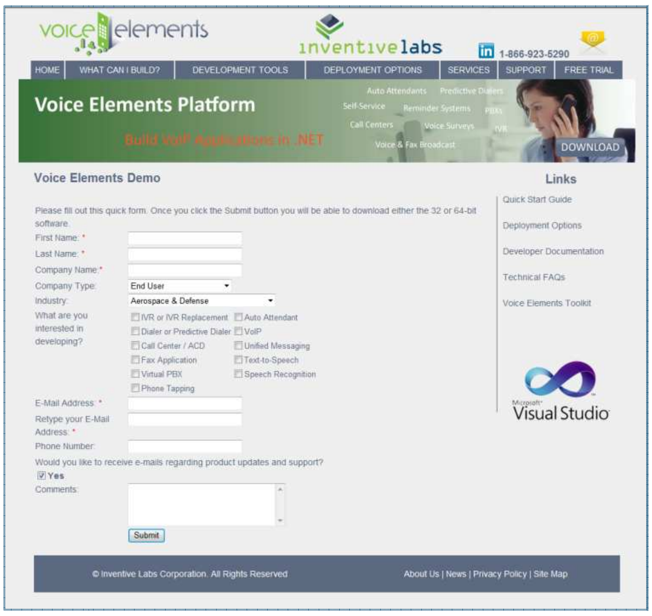
- Windows XP
- Windows Vista
- Windows 7
- Windows 2000 Server
- Windows 2003 Server
- Windows 2008 Server / Windows Server 2008 R2

Visual Studio:

- VS 2005
- VS 2008
- VS 2010
- VS 2012

## Getting Started

To download a 30-day trial of the Voice Elements Platform license, go to: http://voiceelements.com/Products/Contact.aspx. This is the exact same software you will use once you decide to purchase. To begin, you will need to fill out a quick and easy form in order for the license information to be sent to you.



Voice Elements Demo

Please fill out this quick form. Once you click the Submit button you will be able to download either the 32 or 64-bit software.

| | |
|---|---|
| First Name: * | |
| Last Name: * | |
| Company Name:* | |
| Company Type: | End User |
| Industry: | Aerospace & Defense |
| What are you interested in developing? | ☐ IVR or IVR Replacement ☐ Auto Attendant ☐ Dialer or Predictive Dialer ☐ VoIP ☐ Call Center / ACD ☐ Unified Messaging ☐ Fax Application ☐ Text-to-Speech ☐ Virtual PBX ☐ Speech Recognition ☐ Phone Tapping |
| E-Mail Address: * | |
| Retype your E-Mail Address: * | |
| Phone Number: | |

Would you like to receive e-mails regarding product updates and support?
☑ Yes

Comments:

Submit

Links
Quick Start Guide
Deployment Options
Developer Documentation
Technical FAQs
Voice Elements Toolkit

© Inventive Labs Corporation. All Rights Reserved
About Us | News | Privacy Policy | Site Map

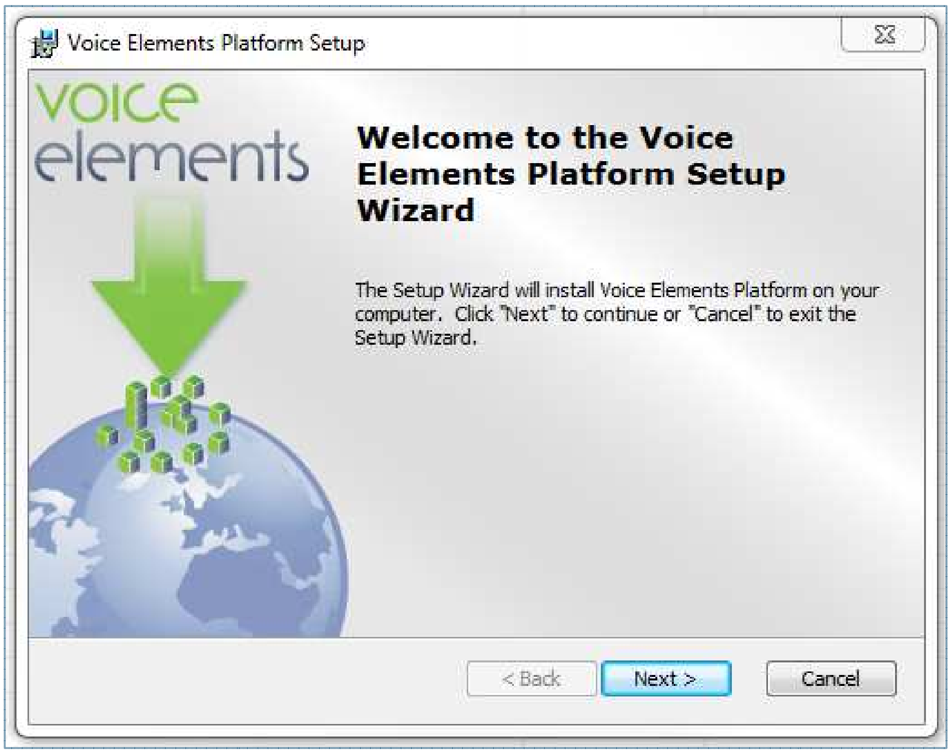
Once you have completed the form, you should receive an email within one minute with detailed instructions on how to get started. The email contains:

- Your License Key that you will need during installation.
- Instructions for installing and configuring Voice Elements Platform
- It contains everything you need to begin writing your own telephony application and testing it on your own system.

## Steps for Installation and Configuration

### Installation

The executable that you download will have all of the software that you need to create and run your Voice Elements Application. Click on the VoiceElementsPlatform.exe file to install Voice Elements. The following welcome screen will appear:

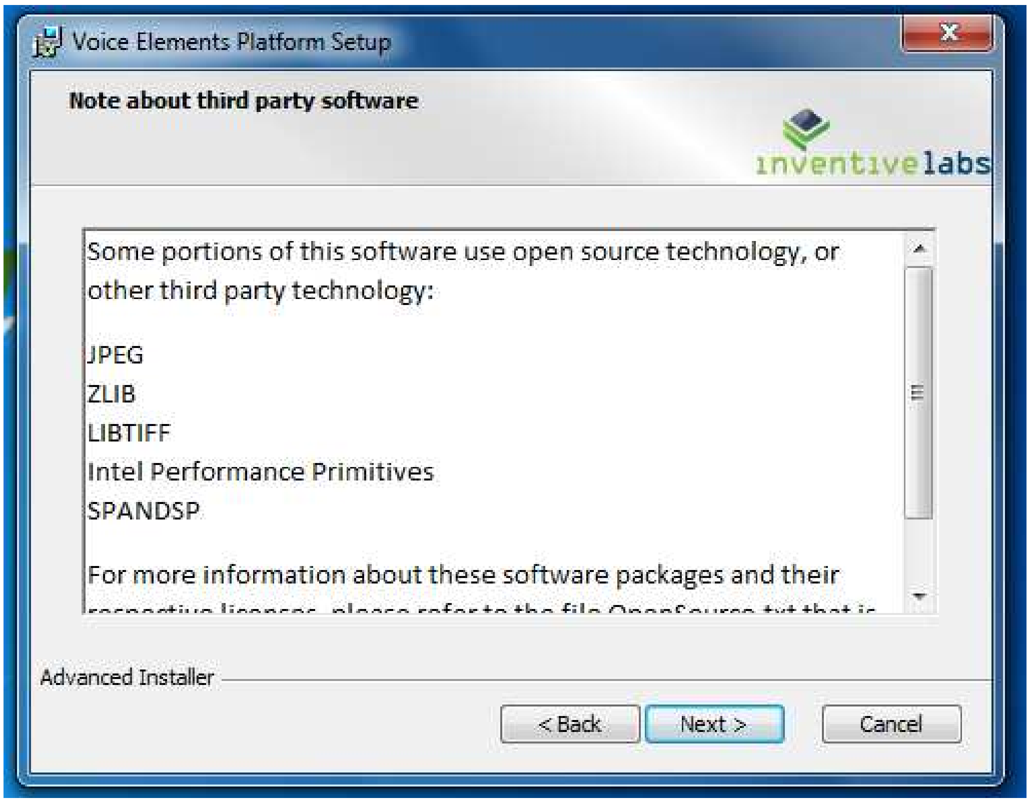


**Welcome to the Voice Elements Platform Setup Wizard**

The Setup Wizard will install Voice Elements Platform on your computer. Click "Next" to continue or "Cancel" to exit the Setup Wizard.

Once you click next, please read and accept the End User License Agreement to move to the next step. There is an option to print EULA for your records.

You will then see a screen which describes the different third party applications that are referenced from Voice Elements Platform.



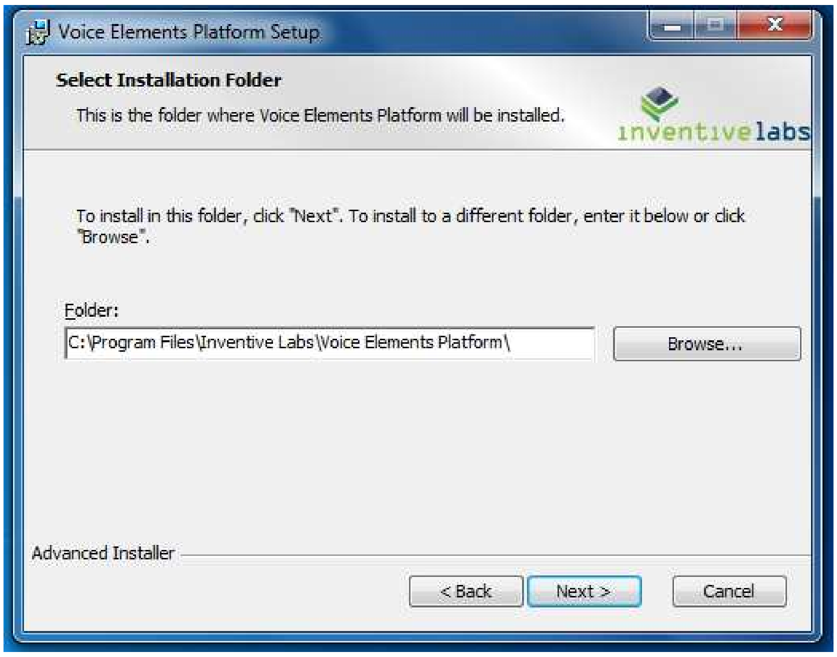
Note about third party software

Some portions of this software use open source technology, or other third party technology:

JPEG
ZLIB
LIBTIFF
Intel Performance Primitives
SPANDSP

For more information about these software packages and their

You will then be prompted to select an installation folder. When you are finished click "Next".


Voice Elements Platform Setup

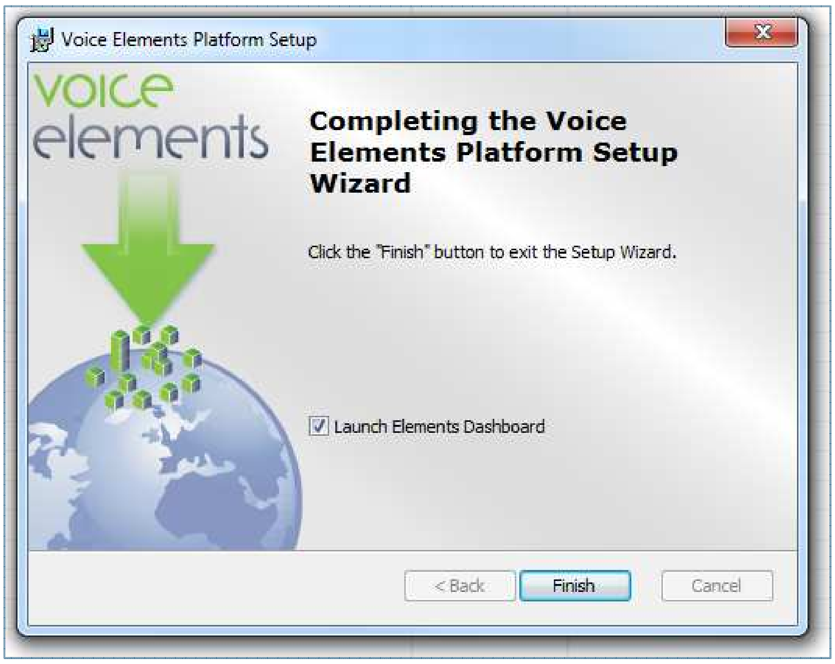
**Select Installation Folder**

This is the folder where Voice Elements Platform will be installed.

To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder: C:\Program Files\Inventive Labs\Voice Elements Platform\

Installation will then start, and when it is finished, you will be prompted to launch the Elements Dashboard.

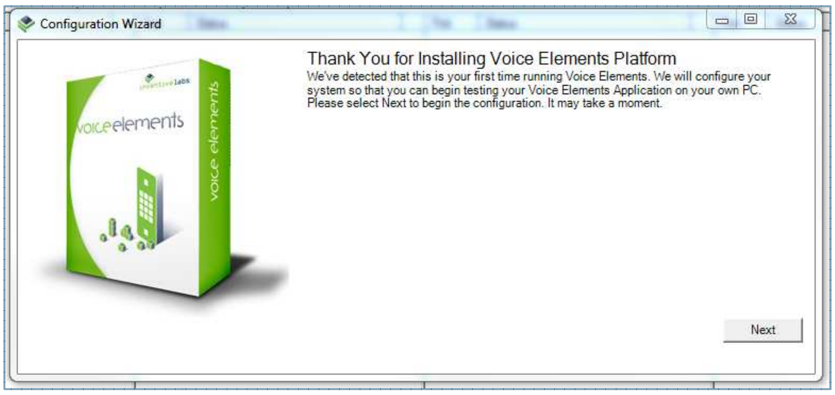

Voice Elements Platform Setup

**Completing the Voice Elements Platform Setup Wizard**

Click the "Finish" button to exit the Setup Wizard.

☑ Launch Elements Dashboard

Click "Finish" when you are ready and Elements Dashboard will guide you through the installation wizard.

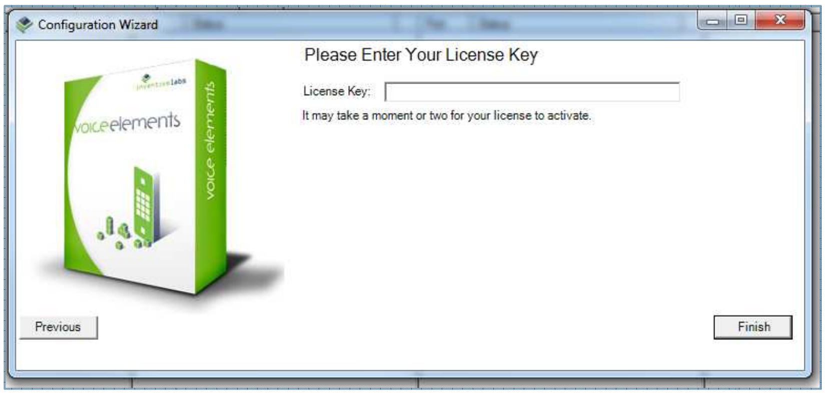


Thank You for Installing Voice Elements Platform

We've detected that this is your first time running Voice Elements. We will configure your system so that you can begin testing your Voice Elements Application on your own PC. Please select Next to begin the configuration. It may take a moment.

When you click "Next", the wizard will detect various settings, and configure the Voice Elements Platform so you can begin running the software locally.

You will then be prompted for your license key. Enter it into the text box, and click "Finish".

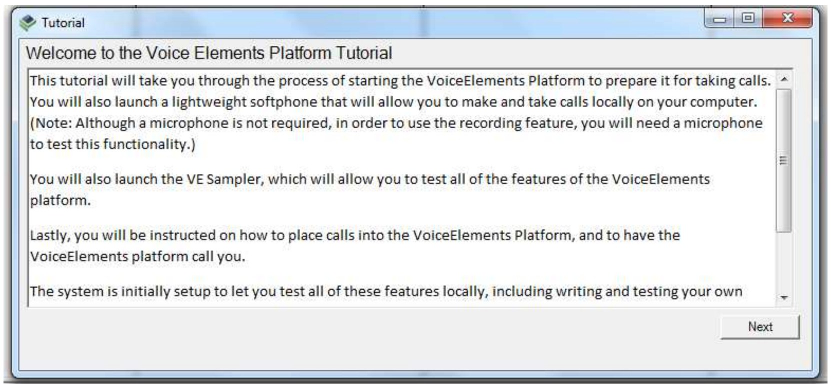


Please Enter Your License Key

License Key:

It may take a moment or two for your license to activate.

A tutorial wizard will then begin. This will explain several concepts, such as how you can start Voice Elements Platform, and how to test using the Soft Phone and the Sampler.



Welcome to the Voice Elements Platform Tutorial

This tutorial will take you through the process of starting the VoiceElements Platform to prepare it for taking calls. You will also launch a lightweight softphone that will allow you to make and take calls locally on your computer. (Note: Although a microphone is not required, in order to use the recording feature, you will need a microphone to test this functionality.)

You will also launch the VE Sampler, which will allow you to test all of the features of the VoiceElements platform.

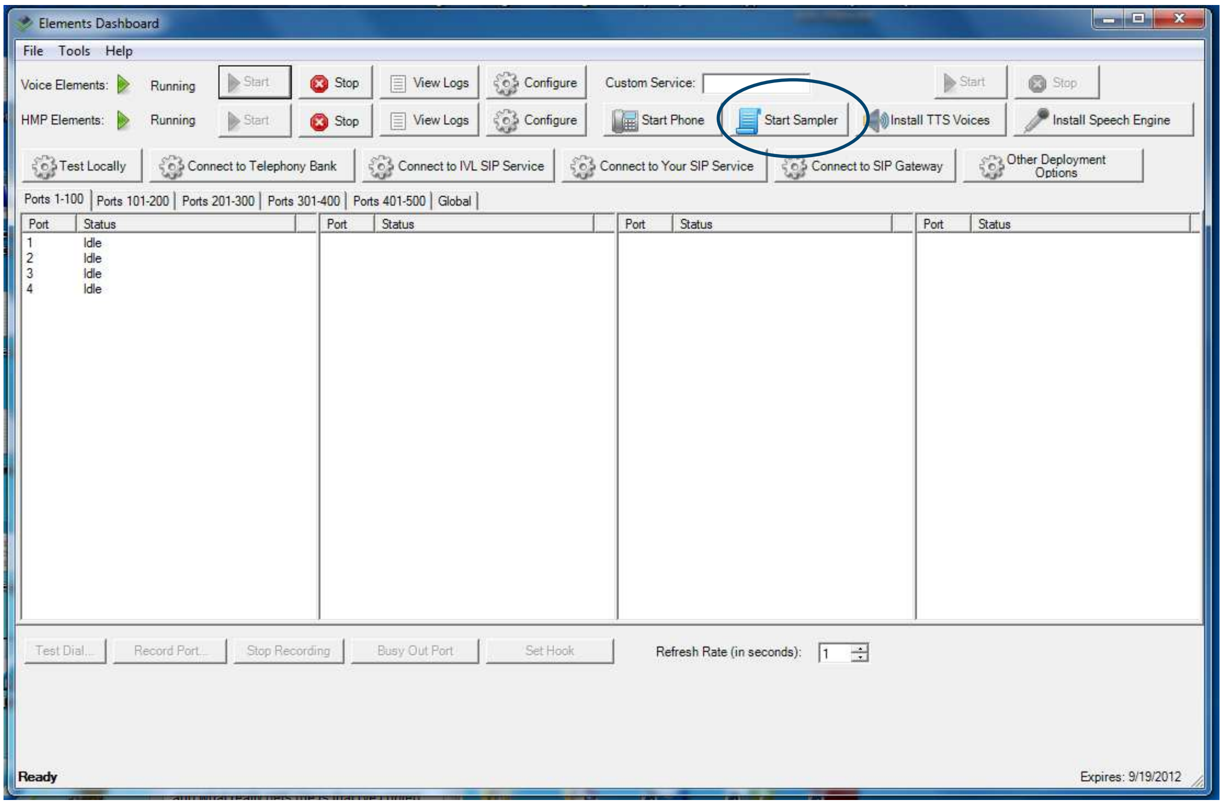
Lastly, you will be instructed on how to place calls into the VoiceElements Platform, and to have the VoiceElements platform call you.

The system is initially setup to let you test all of these features locally, including writing and testing your own

If you followed the basic configuration, Voice Elements Platform should be configured so that you can test everything locally. To begin making test calls, simply click "Start Sampler". This will start up the Voice Elements Sampler. By default, it should connect to your local instance of Voice Elements Server. Alternatively, you may test by connecting to the Inventive Labs Telephony Bank to make live phone calls. Contact support@inventivelabs.com for more information on connecting to the Telephony Bank.

## Testing Your Configuration

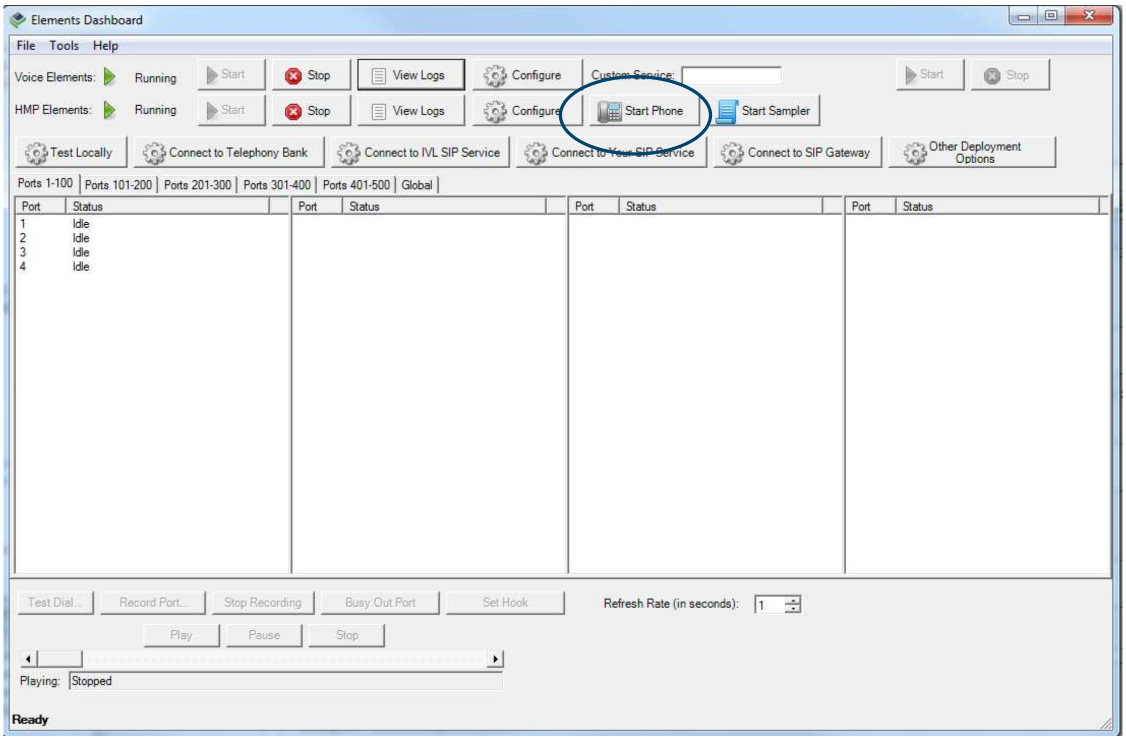
The Voice Elements Sampler contains several test scripts that show how to create basic telephony applications. It is also useful for testing your configuration. You can start it by clicking on the "Start Sampler" button.



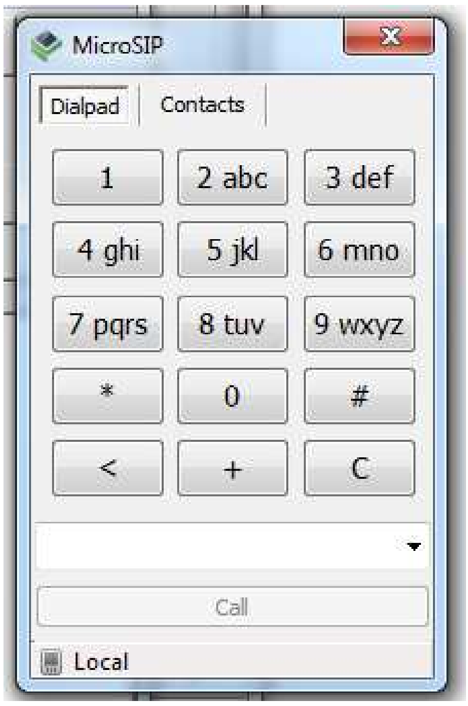
| Port | Status |
|---|---|
| 1 | Idle |
| 2 | Idle |
| 3 | Idle |
| 4 | Idle |

Once Voice Elements Sampler has started and connected, you should bring up MicroSIP. MicroSIP is a softphone that allows you to place VoIP calls, and is useful for testing your Voice Elements Application.

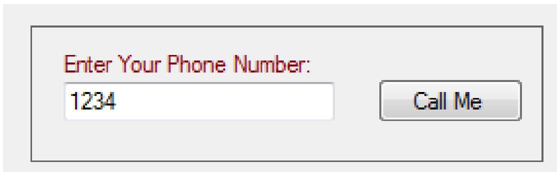
To start MicroSIP, click the "Start Phone" button.


Elements Dashboard
File Tools Help
Voice Elements: Running Start Stop View Logs Configure Custom Service: Start Stop
HMP Elements: Running Start Stop View Logs Configure Start Phone Start Sampler
Test Locally
Connect to Telephony Bank
Connect to IVL SIP Service
Connect to Your SIP Service
Connect to SIP Gateway
Other Deployment Options
Ports 1-100 | Ports 101-200 | Ports 201-300 | Ports 301-400 | Ports 401-500 | Global
Test Dial... Record Port... Stop Recording Busy Out Port Set Hook
Refresh Rate (in seconds): 1
Play Pause Stop
Playing: Stopped
Ready

| Port | Status |
|---|---|
| 1 | Idle |
| 2 | Idle |
| 3 | Idle |
| 4 | Idle |



Now that MicroSIP has started. You can make a test call from Voice Elements Sampler to MicroSIP. To do this, enter your phone number and click "Call Me" from the "Welcome" page on the sampler:



Once you click “Call Me” MicroSip should ring and a short message will play. We *highly encourage* you to try out this feature – it not only tests the Voice Elements configuration, it also demonstrates the power and ease of use of the Voice Elements program.

The code for the Inbound “Call Me” application is in the middle of the control panel, and is also displayed below. The code in the Voice Elements program contains full comments for almost all of the commands if you are interested in reviewing those at this point.

## Taking a Closer Look at the Source Code

### “Call Me” Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using VoiceElements.Client;
using VoiceElements.Common;

namespace VESampler
{
        public class Welcome
    {
        private static Log Log = Sampler.Log;
        private TelephonyServer m_TelephonyServer;
        private ChannelResource m_ChannelResource;
        private VoiceResource m_VoiceResource;
        private string m_NumberToCall;
        public Welcome(TelephonyServer telephonyServer, string
numberToCall)
            m_TelephonyServer = telephonyServer;
            m_NumberToCall = numberToCall;
            m_ChannelResource = m_TelephonyServer.GetChannel();
            m_VoiceResource = m_ChannelResource.VoiceResource;
            m_VoiceResource.Codec = Codec.MULAW_8Khz_8Bit;
            m_VoiceResource.VapFile = @"..\..\english.vap";
            m_ChannelResource.Disconnected += new
Disconnected(m_ChannelResource_Disconnected);
        }
        void m_ChannelResource_Disconnected(object sender,
DisconnectedEventArgs e)
```

Get an outbound channel, reference the supplied voice resource object, and set the standard Codec & Vap File. Also, subscribe to the disconnect event to know if the caller hangs up the phone.

```
            Log.Write("Disconnected Event Received");
```

Indicates the main script for the example and displays the command to place the call. Be sure to use logging often to help with debugging.

Instructs the voice resource to terminate the next voice function on any DTMF digit, to clear the digit buffer prior to the next voice function, and to play the prompt.

Telephony specific exceptions.

Disconnect the call, dispose of the channel resource, and set resource values to null so they will not be referenced.

```
public void RunScript()
        {
            try
            {
                Sampler.Log.Write("Welcome Script Starting");
                Log.Write("Dialing {0}", m_NumberToCall);
                DialResult dr =
m_ChannelResource.Dial(m_NumberToCall);
                Log.Write("The dial result for {0} was: {1}",
m_NumberToCall, dr);

                switch (dr)
                {
                    case DialResult.Connected:
                    case DialResult.HumanDetected:
                    case DialResult.MachineDetected:
                    case DialResult.Successful:
                        break;
                    default:
                        return;
                }
                Log.Write("Playing 'Welcome.wav'");
                m_VoiceResource.TerminationDigits = "ANY";
                m_VoiceResource.ClearDigitBuffer = true;
                m_VoiceResource.Play(@"..\..\Welcome.wav");
            }
            catch (ElementsException ee)
            {
                if (ee is HangupException)
                    Log.Write("The Caller Hungup!");
                else
                    Log.WriteException(ee, "Script Elements
Exception");
            }
            catch (Exception ex)
            {
                Log.WriteException(ex, "Script General
Exception");
            }
            finally
            {
                string deviceName = m_ChannelResource.DeviceName;
                m_ChannelResource.Disconnect();
                m_ChannelResource.Dispose();
                m_ChannelResource = null;
                m_VoiceResource = null;
                m_TelephonyServer = null;
Log.WriteWithId(deviceName, "Welcome Script Finished");
                WelcomeUI.SignalUnlock(this);
            }
        }
    }
}
```

To load and edit this application click Start > Voice Elements Platform > Voice Elements Developer > Shortcut to VE Sampler. This will launch the Visual Studio Solution.

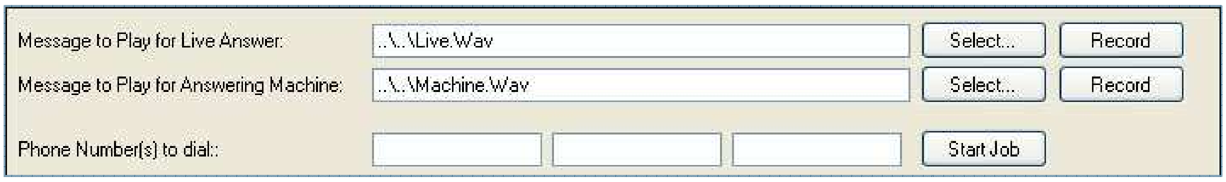
## Sample Applications

In addition to the "Call Me" inbound application there are several other sample applications in the Voice Elements Sampler: an Outbound IVR application, a more complex Inbound IVR application, a Calling Card application, an Inbound Voice Recognition application and a Text-to-Speech application.

### Outbound IVR

This sample application dials out to up to three phone numbers simultaneously. If the called party is detected as human, a custom human message will be played. If the called party is an answering machine, the sample application is programmed to play a different message.

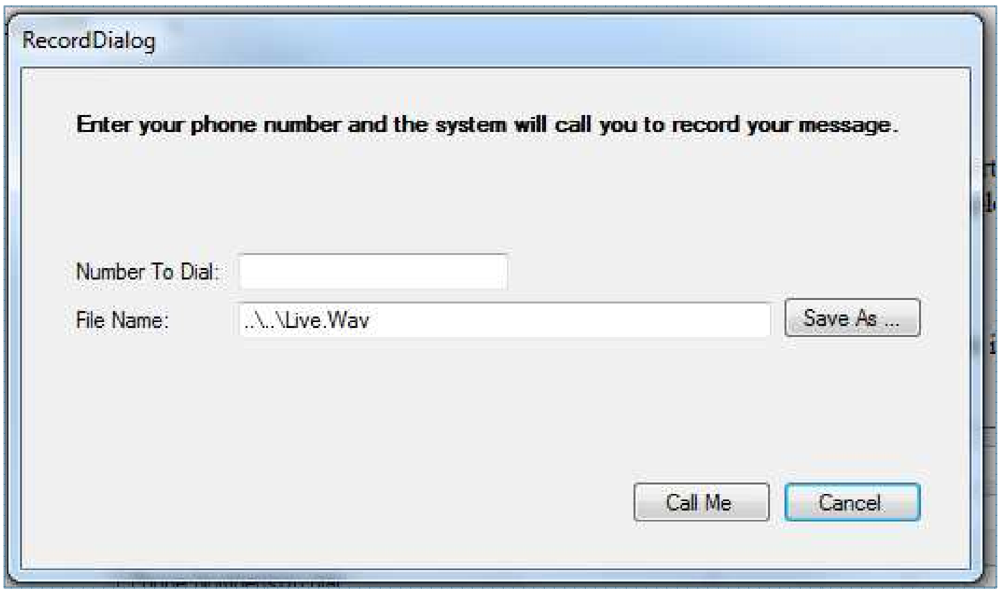
On your Voice Elements control panel you have the option to select or record a .Wav file to be played for a live human or machine answering.


Message to Play for Live Answer: ..\..\Live.Wav

Message to Play for Answering Machine: ..\..\Machine.Wav

Phone Number(s) to dial::

Choosing the "Select" button will let you browse to your own pre-recorded files, while clicking the "Record" button triggers the RecordDialog screen. The RecordDialog screen enables you to enter the phone number to dial where the system will call you so you may record and name your own broadcast message.



Enter your phone number and the system will call you to record your message.

Number To Dial:

File Name: ..\..\Live.Wav

Complete commented code for the Outbound IVR application can be found below the RecordDialog screen on your Voice Elements control panel.

## Outbound IVR Sample Code

Indicates the number to call and the files to play if an answering machine picks up the call.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using VoiceElements.Client;
using VoiceElements.Common;

namespace VESampler
{
    public class OutboundIVR
    {
        private static Log Log = Sampler.Log;
        private TelephonyServer m_TelephonyServer;
        private ChannelResource m_ChannelResource;
        private VoiceResource m_VoiceResource;
        private string m_NumberToCall;
        public string NumberToCall
        {
            get { return m_NumberToCall; }
            set { m_NumberToCall = value; }
        }
        private string m_MachineMessage;
        public string MachineMessage
        {
```

Instructs which files to play if a human answers.

Sets the Telephony Server and gets a channel for outbound usage from the Server.

Use WriteWithID to differentiate between separate instances of the class.

Instruct the Server to tell if a human or machine answers the phone.

```
            get { return m_MachineMessage; }
            set { m_MachineMessage = value; }
        }
        private string m_HumanMessage;
        public string HumanMessage
        {
            get { return m_HumanMessage; }
            set { m_HumanMessage = value; }
        }
        public OutboundIVR(TelephonyServer telephonyServer)
        {
            m_TelephonyServer = telephonyServer;
            m_ChannelResource = m_TelephonyServer.GetChannel();
            m_VoiceResource = m_ChannelResource.VoiceResource;
            Log.Write("Voice resource name: " +
m_VoiceResource.DeviceName);
            m_VoiceResource.Codec = Codec.MULAW_8Khz_8Bit;
            m_VoiceResource.VapFile = @"..\..\english.vap";
            m_ChannelResource.Disconnected += new
Disconnected(m_ChannelResource_Disconnected);
        }
            void m_ChannelResource_Disconnected(object sender,
DisconnectedEventArgs e)
        {
            Log.WriteWithId(m_ChannelResource.DeviceName,
"Disconnected Event Received");
        }
            public void RunScript()
        {
            try
            {
            Log.WriteWithId(m_ChannelResource.DeviceName,
"OutboundIVR Script Starting");
Log.WriteWithId(m_ChannelResource.DeviceName,"Dialing {0}",
m_NumberToCall);
                m_ChannelResource.CallProgress =
CallProgress.AnalyzeCall;
                m_ChannelResource.MaximumTime = 40;
                DialResult dr =
m_ChannelResource.Dial(m_NumberToCall);
                Log.WriteWithId(m_ChannelResource.DeviceName,"The
dial result for {0} was: {1}", m_NumberToCall, dr);
                m_VoiceResource.TerminationDigits = "";
                switch (dr)
                {
                    case DialResult.Connected:
                    case DialResult.HumanDetected:
                    case DialResult.Successful:
Log.WriteWithId(m_ChannelResource.DeviceName, "Playing File {0}",
m_HumanMessage);
                        m_VoiceResource.Play(m_HumanMessage);
                        break;
                    case DialResult.MachineDetected:
                        if (m_MachineMessage == null) return;
                        m_VoiceResource.MaximumSilence = 30;
                        m_VoiceResource.MaximumTime = 600;
Log.WriteWithId(m_ChannelResource.DeviceName, "Waiting for 3
seconds of silence.");
                        m_VoiceResource.GetSilence();
Log.WriteWithId(m_ChannelResource.DeviceName, "Playing File {0}",
m_MachineMessage);
                        m_VoiceResource.Play(m_MachineMessage);
                        break;
                    default:
Log.WriteWithId(m_ChannelResource.DeviceName, "Unexpect Dial
Result!  Cancelling Call");
                        return;
                }
            }
            catch (ElementsException ee)
            {
                if (ee is HangupException)
                    Log.Write("The Caller Hungup!");
                else
                    Log.WriteException(ee, "Script Elements
Exception");
            }
            catch (Exception ex)
            {
                Log.WriteException(ex, "Script General
Exception");
            }
            finally
            {
                string deviceName = m_ChannelResource.DeviceName;
                m_ChannelResource.Disconnect();
                m_ChannelResource.Dispose();
                m_ChannelResource = null;
                m_VoiceResource = null;
                m_TelephonyServer = null;
```

Hang up on the machine if no message is specified.

Set the maximum silence to wait for 3 seconds of silence.

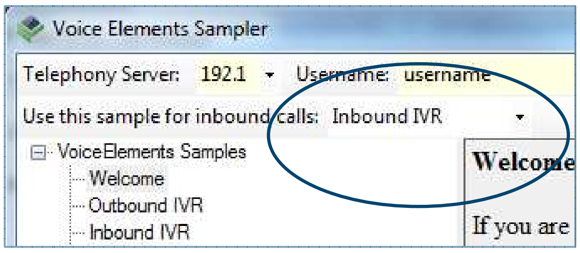
This is a general logic exception, such as a null reference violation or a .NET exception.

Always use the finally block to clean up the call.

```
                Log.WriteWithId(deviceName,"OutboundIVR Script
Finished");

                OutboundIVRUI.SignalUnlock(this);

            }
        }
    }
}
```

## Inbound IVR

This sample application will accept a call from the server. To use this sample, select "Inbound IVR" in the "Use this sample for inbound calls" drop down menu in the toolbar.



Press any digits in MicroSIP and press call.

This sample script demonstrates how simple it is to receive an inbound call and how to answer and terminate a call. It also shows how to use a voice resource. Commented code for the Inbound IVR can be found below the intro text on your Voice Elements control panel.

### Inbound IVR Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using VoiceElements.Client;
using VoiceElements.Common;

namespace VESampler
{
    public class InboundIVR
    {
        private static Log Log = Sampler.Log;
        private TelephonyServer m_TelephonyServer;
        private ChannelResource m_ChannelResource;
        private VoiceResource m_VoiceResource;
        public InboundIVR(TelephonyServer telephonyServer,
ChannelResource channelResource)
        {
            m_TelephonyServer = telephonyServer;
            m_ChannelResource = channelResource;
            m_VoiceResource = channelResource.VoiceResource;
            m_VoiceResource.Codec = Codec.PCM_11Khz_8Bit;
            m_VoiceResource.VapFile = @"..\..\english.vap";
            m_ChannelResource.Disconnected += new
Disconnected(m_ChannelResource_Disconnected);
        }
        void m_ChannelResource_Disconnected(object sender,
DisconnectedEventArgs e)
        {
            Log.Write("Disconnected Event Received");
        }
        public void RunScript()
        {
            try
            {
                m_ChannelResource.Answer();
                Log.Write("Playing 'PleaseEnter.wav'");
                m_VoiceResource.TerminationDigits = "ANY";
                m_VoiceResource.ClearDigitBuffer = true;
                m_VoiceResource.Play(@"..\..\PleaseEnter.wav");
                Log.Write("Getting Digits...");
                m_VoiceResource.ClearDigitBuffer = false;
                m_VoiceResource.MaximumDigits = 10;
                m_VoiceResource.TerminationDigits = "#";
                m_VoiceResource.GetDigits();
                Log.Write("Digits Returned: " +
m_VoiceResource.DigitBuffer);
                Log.Write("Playing 'YouHaveEntered.wav'");
                m_VoiceResource.ClearDigitBuffer = true;
                m_VoiceResource.TerminationDigits = "";
```

Constructor for this script.

A member variable to reference the supplied voice resource object more easily in the script.

Disconnected event processing code.

Instructs the voice resource to play the prompt, “Please enter your password followed by the pound sign.”

Play back the numbers entered by the caller.

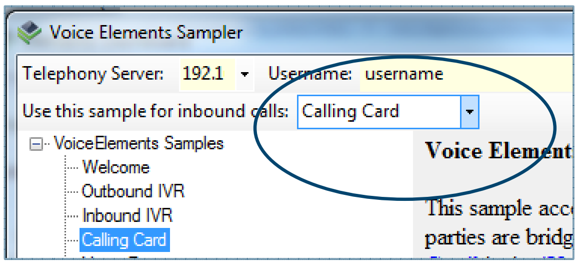
On the next voice function, stop on any DTMF digit input.

Play the Goodbye prompt.

```
m_VoiceResource.Play(@"..\..\YouHaveEntered.wav");
                Log.Write("PlayNumber: " +
m_VoiceResource.DigitBuffer);
m_VoiceResource.PlayNumber(m_VoiceResource.DigitBuffer);
                Log.Write("Playing 'PleaseRecord.wav'");
                m_VoiceResource.Play(@"..\..\PleaseRecord.wav");
                Log.Write("Recording '" +
m_VoiceResource.DeviceName + ".Recording.wav'");

                m_VoiceResource.TerminationDigits = "ANY";

                m_VoiceResource.Record(@"..\..\" +
m_VoiceResource.DeviceName + ".Recording.wav");

                Log.Write("Playing '" +
m_VoiceResource.DeviceName + ".Recording.wav'");
                m_VoiceResource.Play(@"..\..\" +
m_VoiceResource.DeviceName + ".Recording.wav");
                Log.Write("Playing 'Goodbye.wav'");
                m_VoiceResource.Play(@"..\..\Goodbye.wav");
            }
            catch (ElementsException ee)
            {
                if (ee is HangupException)
                    Log.Write("The Caller Hungup!");
                else
                    Log.WriteException(ee, "Script Elements
Exception");
            }
            catch (Exception ex)
            {
                Log.WriteException(ex, "InboundIVR Exception");
            }
            finally
            {
                m_ChannelResource.Disconnect();
                m_ChannelResource.Dispose();
                m_ChannelResource = null;
                m_VoiceResource = null;
                m_TelephonyServer = null;
            }
        }
    }
}
```

## Calling Card Application

The Calling Card sample application accepts an incoming call, prompts for a phone number to be dialed, and then calls that number, bridging the two parties together. When either party hangs up, the call is disconnected. To use this sample, select "Calling Card" in the "Use this sample for inbound calls" drop down menu in the toolbar.



Press any digits in MicroSIP and press call.

Complete commented code for the Calling Card application can be found below the application introduction on your Voice Elements control panel.

### Calling Card Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using VoiceElements.Client;
using VoiceElements.Common;

namespace VESampler
{
    public class CallingCard
    {
        private static Log Log = Sampler.Log;
```

A reference to the Channel Resource the call arrived on.

Simply write to the log that the caller hung up the phone.

Instruct the voice resource to return a maximum of 15 digits.

```
        private TelephonyServer m_TelephonyServer;
        private ChannelResource m_ChannelResource;
        private VoiceResource m_VoiceResource;
        private ChannelResource m_OutboundChannelResource;
        private VoiceResource m_OutboundVoiceResource;
        public CallingCard(TelephonyServer telephonyServer, 
ChannelResource channelResource)
        {
            m_TelephonyServer = telephonyServer;
            m_ChannelResource = channelResource;
            m_VoiceResource = channelResource.VoiceResource;
            m_VoiceResource.Codec = Codec.MULAW_8Khz_8Bit;
            m_VoiceResource.VapFile = @"..\..\english.vap";
            m_ChannelResource.Disconnected += new 
Disconnected(m_ChannelResource_Disconnected);
        }
        private ManualResetEvent m_TerminateCall = new 
ManualResetEvent(false);
        void m_ChannelResource_Disconnected(object sender, 
DisconnectedEventArgs e)
        {
            Log.Write("Disconnected Event Received");
            m_TerminateCall.Set();
        }
        public void RunScript()
        {
            try
            {
                m_ChannelResource.Answer();
                Log.Write("Playing 'NumberToCall.wav'");
                m_VoiceResource.TerminationDigits = "ANY";
                m_VoiceResource.ClearDigitBuffer = true;
                m_VoiceResource.Play(@"..\..\NumberToCall.wav");
                Log.Write("Getting Digits...");
                m_VoiceResource.ClearDigitBuffer = false;
                m_VoiceResource.MaximumDigits = 15;
                m_VoiceResource.TerminationDigits = "#";
                m_VoiceResource.GetDigits();
                Log.Write("Digits Returned: " + 
m_VoiceResource.DigitBuffer);
                m_VoiceResource.Codec = Codec.PCM_11Khz_8Bit;
                Log.Write("Playing 'YouHaveEntered.wav'");
                m_VoiceResource.ClearDigitBuffer = true;
                m_VoiceResource.TerminationDigits = "";
m_VoiceResource.Play(@"..\..\YouHaveEntered.wav");
                Log.Write("PlayNumber: " + 
m_VoiceResource.DigitBuffer);
m_VoiceResource.PlayNumber(m_VoiceResource.DigitBuffer);
                m_VoiceResource.Codec = Codec.MULAW_8Khz_8Bit;
```

Play the prompt: "Please record your name, number and a detailed message..."

Get a new outbound channel.

Route the 1st and 2nd channels back to their voice resource.

```
                Log.Write("Playing 'Dialing.wav'");
                m_VoiceResource.Play(@"..\..\Dialing.wav");
                m_OutboundChannelResource =
m_TelephonyServer.GetChannel();

                try
                {
                    m_OutboundChannelResource.Disconnected += new
Disconnected(OutboundChannelResource_Disconnected);
                    m_OutboundVoiceResource =
m_OutboundChannelResource.VoiceResource;

m_ChannelResource.RouteFull(m_OutboundChannelResource);
                    Log.Write("Dialing {0}",
m_VoiceResource.DigitBuffer);
                    DialResult dr =
m_OutboundChannelResource.Dial(m_VoiceResource.DigitBuffer);

                    Log.Write("The dial result for {0} was: {1}",
m_VoiceResource.DigitBuffer, dr);
                    switch (dr)
                    {
                        case DialResult.Connected:
                        case DialResult.HumanDetected:
                        case DialResult.MachineDetected:
                        case DialResult.Successful:
                            break;
                        default:
                            m_TerminateCall.Set();
                            break;
                    }
                    m_TerminateCall.WaitOne();
                }
                finally
                {
                    m_ChannelResource.RouteFull(m_VoiceResource);
m_OutboundChannelResource.RouteFull(m_OutboundVoiceResource);
                    m_OutboundChannelResource.Disconnect();
                    m_OutboundChannelResource.Dispose();
                    m_OutboundChannelResource = null;
                    m_OutboundVoiceResource = null;
                }
            }
            catch (ElementsException ee)
            {
                if (ee is HangupException)
                    Log.Write("The Caller Hungup!");
                else
```

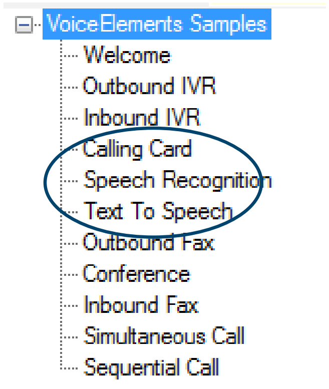
```
                    Log.WriteException(ee, "Script Elements
Exception");
            }
            catch (Exception ex)
            {
                Log.WriteException(ex, "Script Exception");
            }
            finally
            {
                m_ChannelResource.Disconnect();
                m_ChannelResource.Dispose();
                m_ChannelResource = null;
                m_VoiceResource = null;
                m_TelephonyServer = null;
            }
        }

        void OutboundChannelResource_Disconnected(object sender,
DisconnectedEventArgs e)
        {
            Log.Write("Disconnected Event Received");
            m_TerminateCall.Set();
        }
    }
```

Disconnect the call.

Set these values to null so they can't be referenced anymore. Once the call is disposed, the resources can't be utilized.

## Speech Recognition Application

The next sample that you are able to evaluate and try is a voice recognition (speech recognition, ASR) application.   The sample demonstrates how to receive an inbound call and how to answer and test the voice recognition functions.  To use this sample, select "Voice Recognition" in the "Use this sample for inbound calls" drop down menu in the toolbar.



Commented code for the Voice Recognition Application can be found below the intro text on your Voice Elements control panel.

## Speech Recognition Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using VoiceElements.Client;
using VoiceElements.Common;


namespace VESampler
{
    //
    // This sample script demonstrates how simple it is to
receive an inbound call.
    // It demonstrates how to answer and test the voice
recognition functions.
    //

    public class SpeechReco
    {
        // References the main Log File created at startup
        private static Log Log = Sampler.Log;

        // A reference to your Telephony Server Connection
        private TelephonyServer m_TelephonyServer;

        // a reference to the Channel Resource the call arrived
on
        private ChannelResource m_ChannelResource;

        // a reference to the Voice Resource assigned to the call
        private VoiceResource m_VoiceResource;

        // Constuctor for this script
        public SpeechReco(TelephonyServer telephonyServer,
ChannelResource channelResource)
        {
            m_TelephonyServer = telephonyServer;
            m_ChannelResource = channelResource;

            // Use this member variable to reference the supplied
voice resource object easier in the script.
            m_VoiceResource = channelResource.VoiceResource;

            // Set the Codec For this sample.  This is required
for the speech recognition engine.
            m_VoiceResource.Codec = Codec.G711_MULAW_8Khz_8Bit;
```

References the Channel Resource of the arrived call and the Voice Resource assigned to the call.

```
            // Tells the server what Vap File to use for this
call
            m_VoiceResource.VapFile = @"..\..\english.vap";

            // Suscribes to the disconnect event to let us know
if the caller hangs up the phone.
            m_ChannelResource.Disconnected += new
Disconnected(m_ChannelResource_Disconnected);

            m_VoiceResource.Digit += new
Digit(m_VoiceResource_Digit);

            m_VoiceResource.EnableDigitEvents = true;
        }

        void m_VoiceResource_Digit(object sender, DigitEventArgs
e)
        {
            Log.Write("Digit Received: {0}", e.Digit);
        }

        // The Disconnected event processing code
        void m_ChannelResource_Disconnected(object sender,
DisconnectedEventArgs e)
        {
            // Here we will simply write to the log that the
caller hung up the phone.
            Log.Write("Disconnected Event Received");
        }

        // The main script for the InboundIVR sample.
public void RunScript()
{
    try
    {
        // Answer the phone.
        m_ChannelResource.Answer();

        // Use the log file often to help with debugging.
        Log.Write("Answered");

        // Instruct the voice resource to terminate the next
voice function to terminate on ANY DTMF digit
        m_VoiceResource.TerminationDigits = "ANY";

        // Instruct the voice resource to clear the digit buffer
at the beginning of the the next voice function
        m_VoiceResource.ClearDigitBuffer = false;

        while (true)
        {
            // Enable the voice recognition functionality
```

Answers the phone. Use the log file often to help with debugging.

Enable the voice recognition functionality.

Select the grammar file to use for this method.

Barge-In allows the user to interrupt the play.

Get the response from the user.

Log what happened. Notice you are returned the word that was recognized, along with a score.

Turn off Speech Recognition

```
            m_VoiceResource.SpeechRecognitionEnabled = true;

           // Set to multiple plays
            m_VoiceResource.SpeechRecognitionMode = 
VoiceElements.Interface.SpeechRecognitionMode.MultiplePlays;

            // Enable Barge-In.  This allows the talker to stop 
the play by speaking.
            m_VoiceResource.SpeechRecognitionPermitBargeIn = 
true;

            // Select the grammar file to use for this method
            m_VoiceResource.SpeechRecognitionGrammarFile = 
@"..\..\YesNo.xml";

            m_VoiceResource.MaximumTime = 10;

            // Instruct the voice resource to play the prompt: 
"Please Enter your password followed by the pound sign."
            TerminationCode tc = 
m_VoiceResource.PlayTTS(@"Speaking slowly, please say yes or 
no");

            if (tc == TerminationCode.BargeIn)
            {
                m_VoiceResource.MaximumTime = 5;
            }

            // GetDigits works for both getting digits, and for 
getting Speech Recognition responses from a user.
            m_VoiceResource.GetResponse();

            // Turn off voice recognition
            m_VoiceResource.SpeechRecognitionEnabled = false;

            // Log what happened
            Log.Write("Captured Speech: {0}  Score: {1}", 
m_VoiceResource.SpeechRecognitionReturnedWord, 
m_VoiceResource.SpeechRecognitionScore);

            string responseText = "There was an error.";
            if 
(!String.IsNullOrEmpty(m_VoiceResource.SpeechRecognitionReturnedW
ord))
            {
                responseText = 
m_VoiceResource.SpeechRecognitionReturnedWord;
            }

            // Disable Speech recognition for the next play
            m_VoiceResource.SpeechRecognitionEnabled = false;
            // Instruct the voice resource to play what was 
interpreted.
            m_VoiceResource.PlayTTS(responseText);

            // Turn on voice recognition
            m_VoiceResource.SpeechRecognitionEnabled = true;

            // Instruct the voice resource to play the prompt: 
"Please Enter your password followed by the pound sign."
            m_VoiceResource.PlayTTS(@"Was this correct?  You may 
say yes or no.");

            m_VoiceResource.GetResponse();

            // Turn off voice recognition
            m_VoiceResource.SpeechRecognitionEnabled = false;

            // Log what happened
            Log.Write("Captured Speech: {0}  Score: {1}", 
m_VoiceResource.SpeechRecognitionReturnedWord, 
m_VoiceResource.SpeechRecognitionScore);

            if 
(m_VoiceResource.SpeechRecognitionReturnedWord.ToUpper() == 
"TRUE") break;
        }
        // Log often
        Log.Write("Playing 'Goodbye.wav'");

        // Play the goodbye prompt.
        m_VoiceResource.PlayTTS("Goodbye");

    }
    catch (ElementsException ee)
    {
        // These are Telephony Specific exceptions, such an the 
caller hanging up the phone during a play or record.
        if (ee is HangupException)
            Log.Write("The Caller Hungup!");
        else
            Log.WriteException(ee, "Script Elements Exception");
    }
    catch (Exception ex)
    {
        // This would be a general logic exception, such as a 
null reference violation or other .NET exception.
        Log.WriteException(ex, "InboundIVR Exception");
    }
    finally
    {
        // Always use the finally block to clean up the call.
```

This grammar returns "TRUE" when someone says "Yes", or "Yeah", and False, when they say "No", or "Nah". Here we play back either True or False.

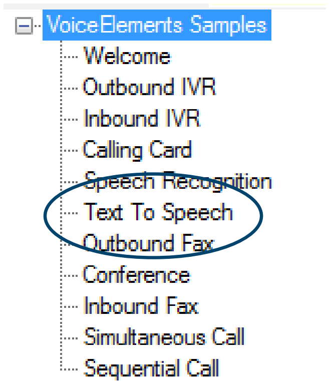
Play the goodbye prompt.

```
        // Note: In the finally block, you should protect each
call to the server with a try-catch block.

        // Disconnect the call (I.E. Hangup)
        try { m_ChannelResource.Disconnect(); }
        catch (Exception ex) { Log.WriteException(ex, "VE Command
Failure In Finally Block"); }

        // Dispose of the channel resource, (this will dispose of
its attached voice resource automatically)
        try { m_ChannelResource.Dispose(); }
        catch (Exception ex) { Log.WriteException(ex, "VE Command
Failure In Finally Block"); }

        // Set these values to null so they cant be referenced
anymore.  Once the call is disposed, the resources cannot be
utilized.
        m_ChannelResource = null;
        m_VoiceResource = null;
        m_TelephonyServer = null;
    }
}
    }
}
```

Dispose the Channel Resource

## Text-to-Speech Application

The last sample that you can test demonstrates the use of Text-to-Speech technology. The sample demonstrates how to receive an inbound call and how to instruct a voice resource to “speak” the entered text. To use this sample, select “Text To Speech” in the “Use this sample for inbound calls” drop down menu in the toolbar.



Commented code for the Text-to-Speech Application can be found below the intro text on your Voice Elements control panel.

### Text-to-Speech Sample Code

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Net.Sockets;
using VoiceElements.Client;
using VoiceElements.Common;

namespace VESampler
{
    public class TextToSpeech
    {
        private static Log Log = Sampler.Log;
        private TelephonyServer m_TelephonyServer;
        private ChannelResource m_ChannelResource;
        private VoiceResource m_VoiceResource;
        public TextToSpeech(TelephonyServer telephonyServer,
ChannelResource channelResource)
        {
            m_TelephonyServer = telephonyServer;
            m_ChannelResource = channelResource;
            m_VoiceResource = channelResource.VoiceResource;
            m_VoiceResource.Codec = Codec.PCM_11Khz_8Bit;
```

Use this member variable to reference the supplied voice resource object easier in the script.

```
            m_VoiceResource.VapFile = @"..\..\english.vap";
            m_ChannelResource.Disconnected += new
Disconnected(m_ChannelResource_Disconnected);
```

```
            m_VoiceResource.Digit += new
Digit(m_VoiceResource_Digit);
            m_VoiceResource.EnableDigitEvents = true;
        }
        void m_VoiceResource_Digit(object sender, DigitEventArgs
e)
        {
            Log.Write("Digit Received: {0}", e.Digit);
        }
        void m_ChannelResource_Disconnected(object sender,
DisconnectedEventArgs e)
        {
            Log.Write("Disconnected Event Received");
        }
        public void RunScript()
        {
            try
            {
                m_ChannelResource.Answer();
                Log.Write("Playing TTS Message");
                m_VoiceResource.TerminationDigits = "ANY";
                m_VoiceResource.ClearDigitBuffer = true;
                string finalText =
Sampler.TTS_Message.ToLower().Replace("record", "<pron sym=\"r iy
- k ao r d 1\">record</pron>");
                m_VoiceResource.PlayTTS(finalText);
                Log.Write("Playing 'Goodbye.wav'");
                m_VoiceResource.Codec = Codec.PCM_11Khz_8Bit;
                m_VoiceResource.Play(@"..\..\Goodbye.wav");
            }
            catch (ElementsException ee)
            {
                if (ee is HangupException)
                    Log.Write("The Caller Hungup!");
                else
                    Log.WriteException(ee, "Script Elements
Exception");
            }
            catch (Exception ex)
            {
                Log.WriteException(ex, "InboundIVR Exception");
            }
            finally
            {
                m_ChannelResource.Disconnect();
                m_ChannelResource.Dispose();
                m_ChannelResource = null;
```

The Disconnected event processing code.

Fix up the word "record" so it sounds like "ree" "kord" and not like an old vinyl "wreck" "ord"!

Instruct the voice resource to play the text.

Disconnect the call (i.e. Hangup)

```
                m_VoiceResource = null;
                m_TelephonyServer = null;
            }
        }
    }
```

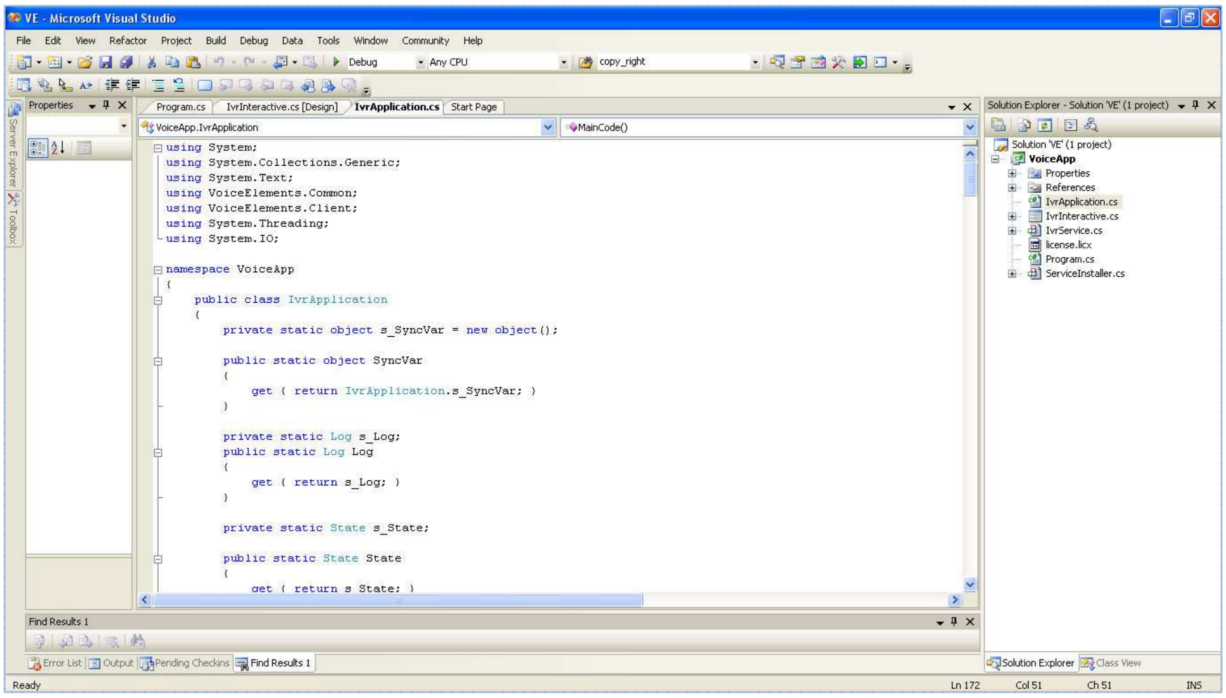
## Your First Project in Visual Studio

- This project can load as a Windows System Service
- Contains a SKELETON project that you can start all your production applications with
- Choose C# or VB.NET

Now that you have tried out a few of the sample applications, you are ready to start your first project.

Download the skeleton project here:
C#: http://download.voiceelements.com/ve.zip
VB.NET: http://download.voiceelements.com/VEVB.zip

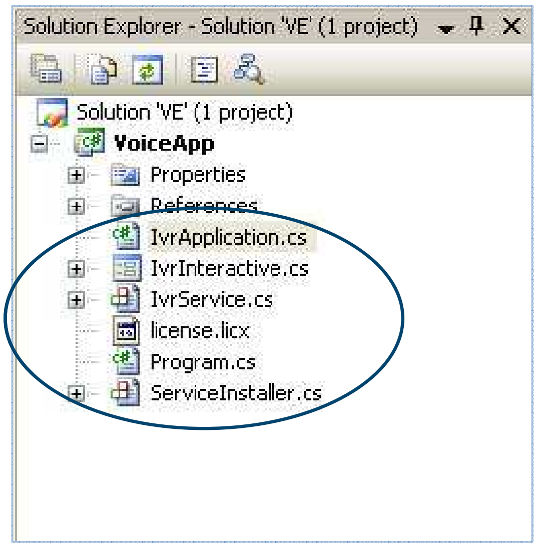
Unzip the project into a folder and open the VE solution.



```
using System;
using System.Collections.Generic;
using System.Text;
using VoiceElements.Common;
using VoiceElements.Client;
using System.Threading;
using System.IO;

namespace VoiceApp
{
    public class IvrApplication
    {
        private static object s_SyncVar = new object();

        public static object SyncVar
        {
            get { return IvrApplication.s_SyncVar; }
        }

        private static Log s_Log;
        public static Log Log
        {
            get { return s_Log; }
        }

        private static State s_State;

        public static State State
        {
            get { return s_State; }
```

## Modules

There are six modules in the 'VE' Solution.



Starting from the top of the list, they are:

1. **IvrApplication.cs**
   IvrApplication.cs is the control application to set up Voice Elements.

2. **IvrInteractive.cs**
   The IvrInteractive.cs module is the code to run an interactive form.

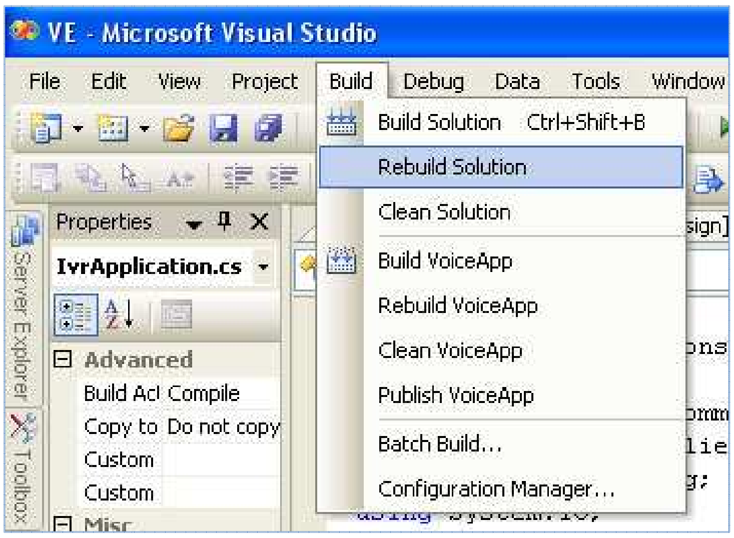
3. **IvrService.cs**
   IvrService.cs is the code that enables the solution to be installed as a Windows system service.

4. **license.licx**
   The license.licx module is no longer required.

5. **Program.cs**
   This module starts the program and decides whether you are interactive or running as a service.

6. **ServiceInstaller.cs**
   The ServiceInstaller.cs is the code for installing as service.

Next, try to “Rebuild Solution” to make sure that it builds.



## Settings

To go to the main code to view and edit the ‘VE’ Solution, open the **IvrApplication.cs** module. There are a few settings at this point that you should define.

First, set the host address (or IP address) of your Voice Elements server. To do this, scroll approx. half way down the code to where you see the following comments:

```
// UPDATE YOUR SERVER ADDRESS HERE
System.Net.IPAddress[] ips =
System.Net.Dns.GetHostAddresses("bank.voicelements.com");

if (ips == null || ips.Length == 0) throw new Exception("Error:
Could not resolve Telephony Server specified!");

string sIpaddress = @"gtcp://" + ips[0].ToString() + ":54331";

Log.Write("Connecting to: {0}", sIpaddress);
```

To do this, simply replace bank.voiceelements.com with your local IP Address. This would be the same IP address that you see in the Voice Elements Sampler under Telephony Server.

Next, set your Username and Password.

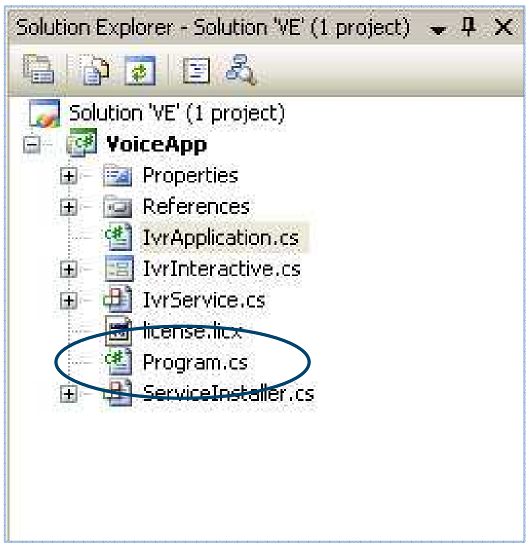
```
// CHANGE YOUR USERNAME AND PASSWORD HERE
s_TelephonyServer = new TelephonyServer(sIpaddress, "username",
"password");
```

## Connecting

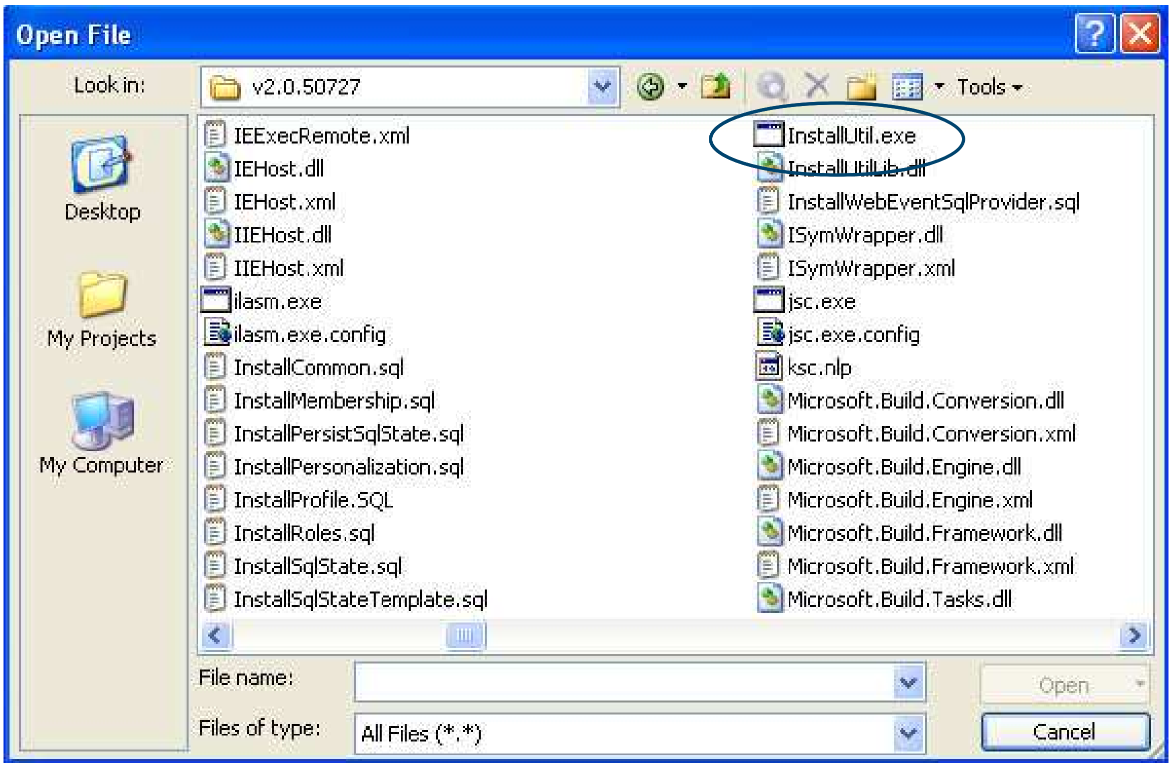
Lastly, try running or debugging the program to make sure it connects.  Test dialing or receiving calls using MicroSIP.

## Installing as a Service

To install the 'VE' Solution as a service, change the **program.cs** module so that you can set the service name and display name.

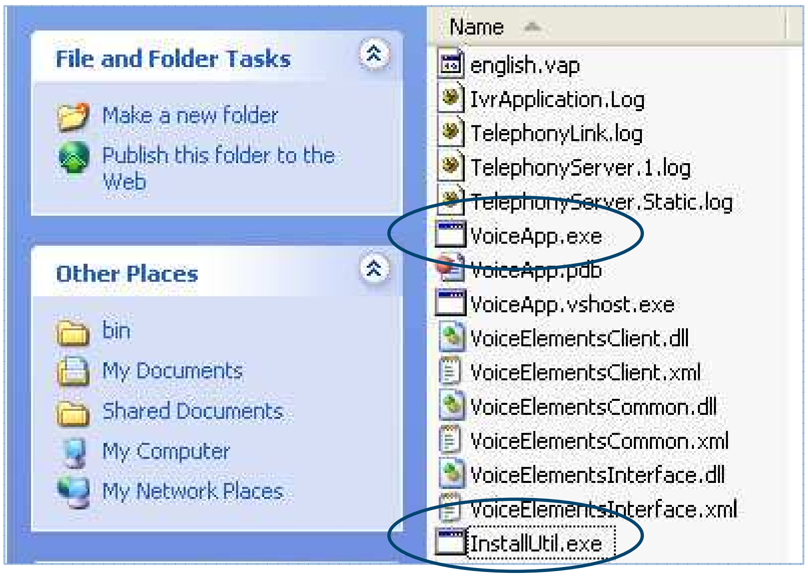


Next, find the InstallUtil in your C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727 folder.



IEExecRemote.xml
IEHost.dll
IEHost.xml
IIEHost.dll
IIEHost.xml
ilasm.exe
ilasm.exe.config
InstallCommon.sql
InstallMembership.sql
InstallPersistSqlState.sql
InstallPersonalization.sql
InstallProfile.SQL
InstallRoles.sql
InstallSqlState.sql
InstallSqlStateTemplate.sql
InstallUtil.exe
InstallUtilLib.dll
InstallWebEventSqlProvider.sql
ISymWrapper.dll
ISymWrapper.xml
jsc.exe
jsc.exe.config
ksc.nlp
Microsoft.Build.Conversion.dll
Microsoft.Build.Conversion.xml
Microsoft.Build.Engine.dll
Microsoft.Build.Engine.xml
Microsoft.Build.Framework.dll
Microsoft.Build.Framework.xml
Microsoft.Build.Tasks.dll

Once you have found IntallUtil.exe, copy it into the same folder as the compiled VoiceApp.exe – which should be located in the Debug folder on your local drive …\VE\VoiceApp\bin\Debug folder.

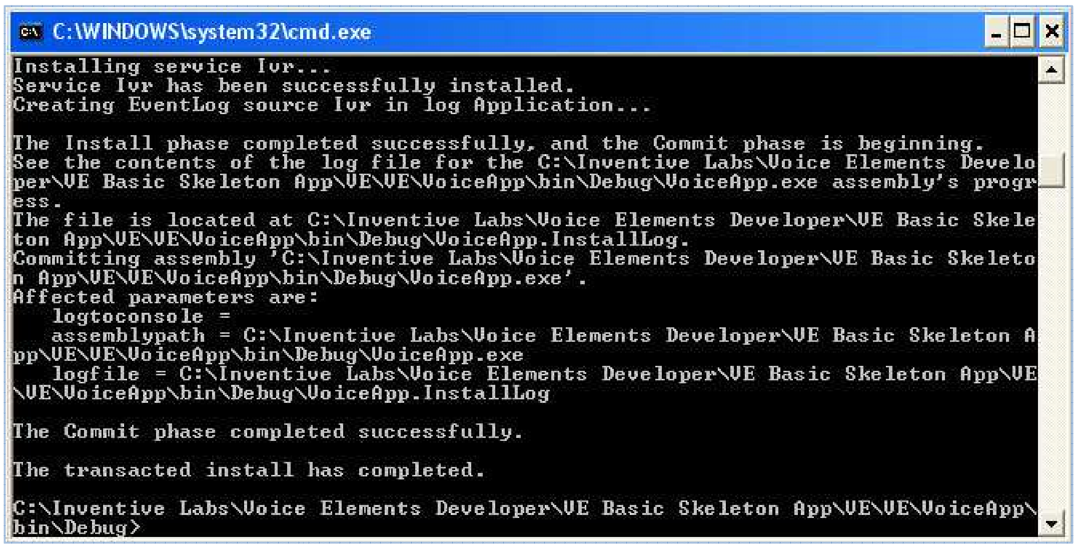


english.vap
IvrApplication.Log
TelephonyLink.log
TelephonyServer.1.log
TelephonyServer.Static.log
VoiceApp.exe
VoiceApp.pdb
VoiceApp.vshost.exe
VoiceElementsClient.dll
VoiceElementsClient.xml
VoiceElementsCommon.dll
VoiceElementsCommon.xml
VoiceElementsInterface.dll
VoiceElementsInterface.xml
InstallUtil.exe

Next, open a command prompt, navigate to that folder and type the installutil voiceapp.exe.

Then, run from a command prompt:

Installutil VoiceApp.exe (or the name of your application if you renamed it).

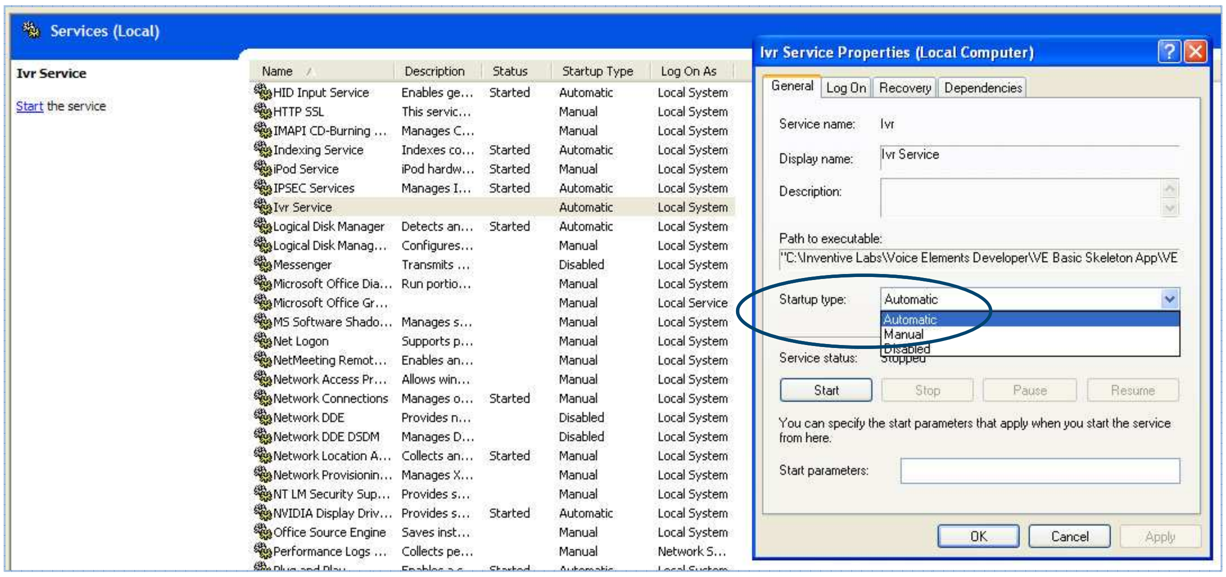


C:\WINDOWS\system32\cmd.exe

```
Installing service Ivr...
Service Ivr has been successfully installed.
Creating EventLog source Ivr in log Application...

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the C:\Inventive Labs\Voice Elements Develo
per\VE Basic Skeleton App\VE\VE\VoiceApp\bin\Debug\VoiceApp.exe assembly's progr
ess.
The file is located at C:\Inventive Labs\Voice Elements Developer\VE Basic Skele
ton App\VE\VE\VoiceApp\bin\Debug\VoiceApp.InstallLog.
Committing assembly 'C:\Inventive Labs\Voice Elements Developer\VE Basic Skeleto
n App\VE\VE\VoiceApp\bin\Debug\VoiceApp.exe'.
Affected parameters are:
   logtoconsole =
   assemblypath = C:\Inventive Labs\Voice Elements Developer\VE Basic Skeleton A
pp\VE\VE\VoiceApp\bin\Debug\VoiceApp.exe
   logfile = C:\Inventive Labs\Voice Elements Developer\VE Basic Skeleton App\VE
\VE\VoiceApp\bin\Debug\VoiceApp.InstallLog

The Commit phase completed successfully.

The transacted install has completed.

C:\Inventive Labs\Voice Elements Developer\VE Basic Skeleton App\VE\VE\VoiceApp\
bin\Debug>
```

If the install is successful, as shown above, the program will install as a system service which you can then manage from the Service manager.

At this point you can go into the Services manager located in the Administrative Tools folder of your Control Panel , find your service (in this case Ivr Service) and set to Automatic if you want it to start automatically when the system reboots.



Services (Local)

**Ivr Service**

Start the service

| Name | Description | Status | Startup Type | Log On As |
|---|---|---|---|---|
| HID Input Service | Enables ge... | Started | Automatic | Local System |
| HTTP SSL | This servic... | | Manual | Local System |
| IMAPI CD-Burning ... | Manages C... | | Manual | Local System |
| Indexing Service | Indexes co... | Started | Automatic | Local System |
| iPod Service | iPod hardw... | Started | Manual | Local System |
| IPSEC Services | Manages I... | Started | Automatic | Local System |
| Ivr Service | | | Automatic | Local System |
| Logical Disk Manager | Detects an... | Started | Automatic | Local System |
| Logical Disk Manag... | Configures... | | Manual | Local System |
| Messenger | Transmits ... | | Disabled | Local System |
| Microsoft Office Dia... | Run portio... | | Manual | Local System |
| Microsoft Office Gr... | | | Manual | Local Service |
| MS Software Shado... | Manages s... | | Manual | Local System |
| Net Logon | Supports p... | | Manual | Local System |
| NetMeeting Remot... | Enables an... | | Manual | Local System |
| Network Access Pr... | Allows win... | | Manual | Local System |
| Network Connections | Manages o... | Started | Manual | Local System |
| Network DDE | Provides n... | | Disabled | Local System |
| Network DDE DSDM | Manages D... | | Disabled | Local System |
| Network Location A... | Collects an... | Started | Manual | Local System |
| Network Provisionin... | Manages X... | | Manual | Local System |
| NT LM Security Sup... | Provides s... | | Manual | Local System |
| NVIDIA Display Driv... | Provides s... | Started | Automatic | Local System |
| Office Source Engine | Saves inst... | | Manual | Local System |
| Performance Logs ... | Collects pe... | | Manual | Network S... |

Ivr Service Properties (Local Computer)

General | Log On | Recovery | Dependencies

Service name: Ivr

Display name: Ivr Service

Description:

Path to executable:
"C:\Inventive Labs\Voice Elements Developer\VE Basic Skeleton App\VE

Startup type: Automatic

Automatic
Manual
Disabled

Service status: Stopped

Start | Stop | Pause | Resume

You can specify the start parameters that apply when you start the service from here.

Start parameters:

OK | Cancel | Apply

## Your Voice Elements Solution

Now you are free to customize your application. In the code you will find the following comments which alert you to where you can add your own outbound IVR logic.

```
 // At this point you are in control.  You can farm out calls
from a database,
 // or you could code the IvrInteractive Form and create a GUI
for handling your calls.
  // Follow the OutboundIvr example from the Sampler on how to
make an outbound class for new calls.
```

For Inbound IVR logic it is here:

```
// Handle the New Call Here
// You should actually create a class to handle this call.  See
the InboundIvr example from the Sampler application on how to do
this.

// You can subscribe to get the disconnected event.
```

If you have any questions regarding any of the sample applications or the 'VE' solution application, please contact Inventive Labs at support@inventivelabs.com.

**www.voiceelements.com**

**(866) 923-5290**
**sales@inventivelabs.com**

**Inventive Labs Corporation**
**4955 E. Preserve Court**
**Greenwood Village, CO 80121**