|  |
| --- |
|  |
| C# Introduction |
| Getting Started with C# |
|  |
|  |
|  |

|  |
| --- |
|  |

# 

# C# Introduction

## Objectives

Once you complete this lab, you should understand how to:

* Create, build, and debug a C# project in Visual Studio
* Use a speech synthesizer to impress your friends and family.

For this lab you’ll be working in the **before** directory underneath the location of this document. The **after** directory contains a completed version of this lab.

## Part 1 – Building a Project

1. Open Visual Studio. Select the menu option **File -> New Project**.
2. In the New Project dialog box, expand the **Installed Templates** -> **Visual C#** node and click on **Windows**.
3. Select **ConsoleApplication** from the list of templates. Give this project the name **Introduction** and set the folder to the **before** directory of the lab.

You should see the Program.cs file appear in the editor window.

1. Between the curly braces of the Main method, use **Console.WriteLine** to print a greeting (like Hello, World!).

class Program  
{  
    static void Main(string[] args)  
    {  
**Console.WriteLine("Hello, World!");**    }  
}

1. Go to the **Build** menu, and select **Build Solution**.

If everything is working, you should see “Build succeeded” in the lower left corner of Visual Studio.

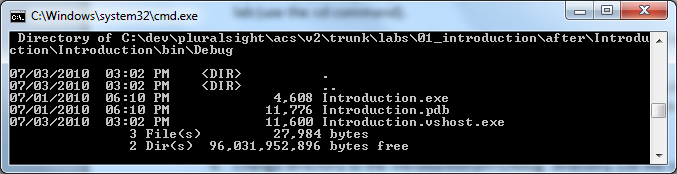
1. Go to the **Debug** menu and select **Start Without Debugging**.

You should see a console window appear with your message. The window will automatically pause and wait for you to press a key before closing. Close the program and return to Visual Studio.

1. Open a command prompt window (cmd.exe) and move to the **before** directory for this lab (use the cd command).
2. Change directory to the **Introduction** directory. List the contents of the directory (dir).

Notice there is an Introduction.sln file in this directory. A .sln (solution file) keeps track of all the projects you have in a Visual Studio solution. A single solution could contain multiple applications and projects with shared code.

1. Change directory to the **Introduction\bin\Debug** directory. List the contents of the directory.



The .exe file is your compiled application. The .pdb file is full of information the runtime will use for debugging (pdb stands for “program database”). For example, the runtime can use information in the pdb to correlate the location of a runtime error with the file and line number of the error in your source code. Visual Studio also uses the [.vshost.exe](http://msdn.microsoft.com/en-us/library/ms185331(VS.80).aspx) file during debugging sessions (you never want to deploy this file or run it directly).

Currently we are in the Debug directory because we are building a Debug version of the program. You can switch between Debug and Release using the Build -> Configuration Manager menu option.

1. **Run Introduction.exe** – it should print your message.

In the following steps we’ll take an argument from the command line and use it in our output.

1. Leave the command prompt open and return to Visual Studio.
2. Replace the code inside of Main with the following two lines.

string message = String.Format("Hello, {0}", args[0]);  
  
Console.WriteLine(message);

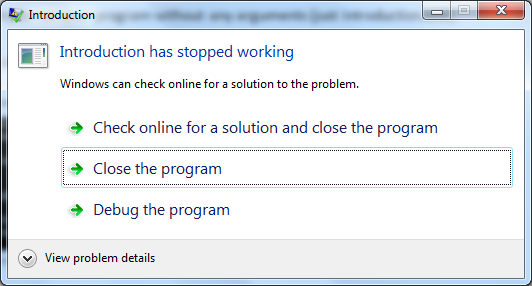
This code builds a string using String.Format. The “{0}” is a placeholder where the Format method will inject a parameter value. In this code we are injecting args[0], which would be the first argument passed to this program from the command line.

1. **Build** the solution (Shift+Ctrl+B).
2. Return to the command line**. Run introduction.exe** and pass your name as a parameter. For example: “Introduction.exe Susan”.

The program should execute and print (in this example) Hello, Susan.

1. Now try to run your program without any arguments (just introduction.exe).

The program should crash!

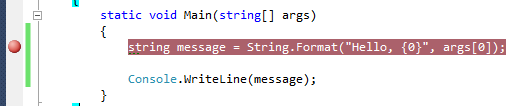


1. Click “Close the program” and return to Visual Studio.

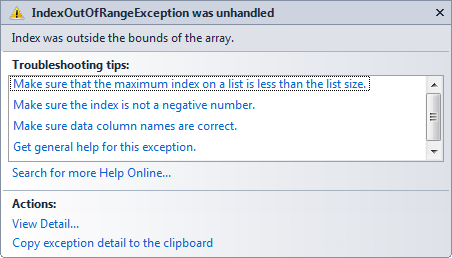
# Part II – Debugging

1. In Visual Studio, place the cursor on the line with String.Format and press **F9**

F9 creates a breakpoint. When you run the programmer with the debugger, the IDE will halt execution at this point.



1. Press **F5** to run the program with the debugger.
2. Hover the mouse over the **args** parameter to Main – it will show as empty.
3. Right-click args and click **Add Watch**.
4. Press **F10** to step to the next line of code. An error should occur – Visual Studio should display an exception box.



The problem is we are trying to access an element in an empty array (the args array). In the CLR this is an error condition that can stop the program from executing. We’ll see how to pass the parameter when using the debugger, but first let’s avoid the error by checking the length of the array.

1. Stop debugging (from the Debug menu, or Shift+F5).
2. Surround the string formatting and printing with an **if** statement that will prevent the exception when args is empty (hint: you can check the **Length** property of args). The code should look similar to the following:

if (args.Length > 0)  
{  
    string message = String.Format("Hello, {0}", args[0]);  
    Console.WriteLine(message);  
}

1. Press **F5** to run the application again.

The application should run without creating an error, but will also run and exit immedialty without printing a message.

1. Add a **Console.ReadLine** statement at the bottom of Main (outside the closing } for the if statement.

if (args.Length > 0)  
{  
    string message = String.Format("Hello, {0}", args[0]);  
    Console.WriteLine(message);  
}  
**Console.ReadLine();**

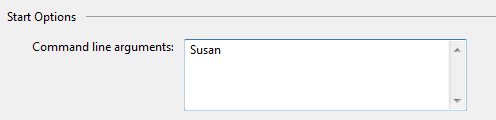
1. Press **F5** to run the application again.

The program should launch and wait for you to press Enter.

1. Press **Enter** to satisfy the ReadLine method and the program should finish executing.

Now let’s tell the Visual Studio Debugger to pass a parameter to the application when we launch the debugger.

1. Open the **Solution Explorer** window (View -> Solution Explorer).
2. Right-click the **Introduction** project and select **Properties**.
3. On the right-hand side, select the **Debug** tab.
4. Under command line arguments, enter your name.



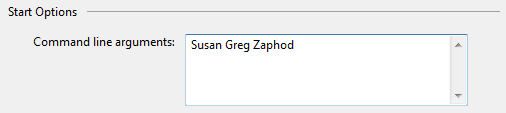
1. Press **F5** to run the application.

The application should hit the breakpoint.

1. Press **F5** again to resume execution.

The application should print a message to the console (like Hello, Susan).

1. Press **Enter** to close the application.
2. Return to the Debug properties for the application.
3. Enter multiple names for the command line arguments.

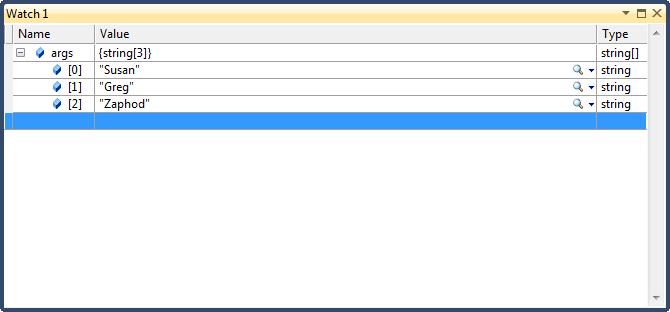


1. Press **F5** to debug the application.
2. When your breakpoint is hit, turn your attention to the **Watch** window.

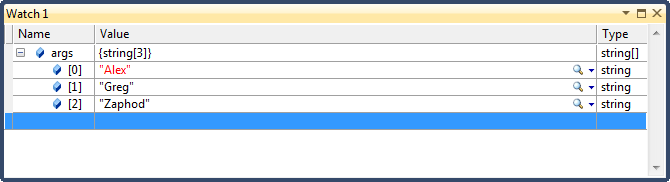
The Watch window should still contain a line for the args variable you added earlier. Any variables you place in the watch window will stick around between debugging sessions. At some point you might want to delete a watch, in which case you can just click on the variable and press the delete key (but don’t delete args just yet).

Also note we are passing 3 parameters to the program, but it only makes use of the first parameter.

1. Click on the + sign next to args and the debugger will show you the entire content of the array.



1. Right click on **args[0]** and select **Edit Value**.
2. Enter “Alex” (with the quotes) as the new value.



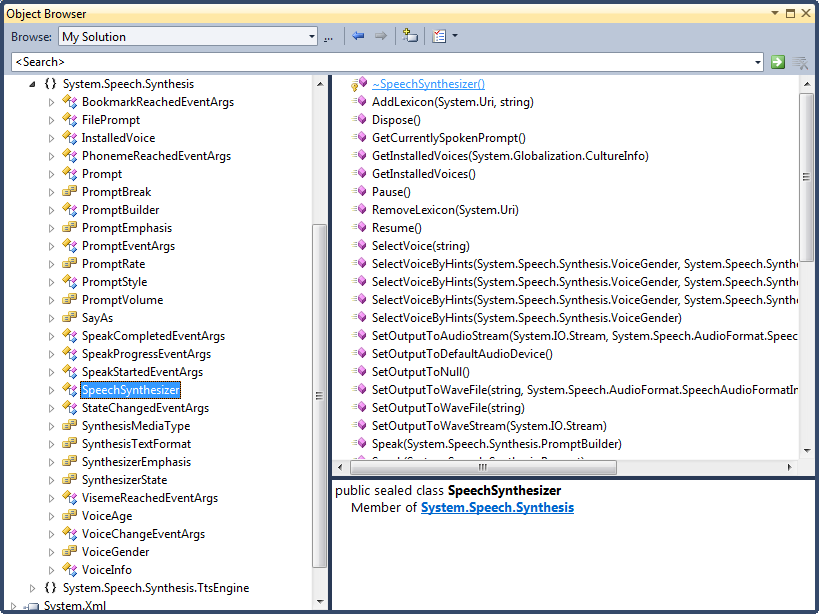
1. Press **F5** to resume execution.

The program should display “Hello, Alex”.

1. Press Enter to close the program.

# Part III – Speaking of References

1. Right-click the **Introduction** project and select **Add Reference**.
2. Locate **System.Speech** and add the assembly as a reference.
3. In the Solution Explorer window, expand the **References** node.
4. Right-click System.Speech and select **View in Object Browser**.
5. Drill into the System.Speech assembly until you find **System.Speech.Synthesis.SpeechSynthesizer**



Notice the left hand side of the Object Browser shows you the available types in an assembly, and these types are organized by namespaces. When you select a type, the members of the type appear on the right. These are the methods and properties you can use on the type. The SpeechSynthesizer has methods like Pause, Resume, and Speak.

1. Return to **Program.cs** in Visual Studio.
2. At the top of the file, after the other using statements, add a using for System.Speech.Synthesis.

using System.Speech.Synthesis;

This allows us to use the SpeechSynthesizer class in our Main method without specifying the entire namespace.

1. Inside the if statement and after the Console.WriteLine, create a new instance of the **Speech Synthesizer** class.
2. On the next line of code, tell the synthesizer to **speak** your message.

SpeechSynthesizer synth = new SpeechSynthesizer();  
        synth.Speak(message);

1. Press **F5** to run the application (make sure you have the volume on your computer turned up).

## Conclusion

Congratualations! You’ve built a simple C# project and learned a few tips about Visual Studio. If you’ve already worked with .NET, you might want to go one step further and make the speech synthesizer read from a text file on disk (you can use File.ReadAllLine to quickly read the text into a string).