



# INTRODUCTION TO VISUAL BASIC 2015

---

FOR COMPLETE BEGINNERS



MARK MASLACH

# **Introduction to Visual Basic 2015**

by Mark Maslach

Geek University Press  
Malesnica 52, Zagreb  
Croatia

# **INTRODUCTION TO VISUAL BASIC 2015**

by Marko Maslac

First edition

Copyright© 2017 Geek University Press

## **Published by:**

Geek University Press

Malesnica 52, Zagreb, Croatia

ISBN-13: 978-1548117078

ISBN-10: 1548117072

## **Disclaimer**

This book is designed to provide information about Visual Basic 2015. Every effort has been made to make this book as complete and as accurate as possible, but no warranty is implied. The information is provided on an as is basis. Neither the authors, Geek University Press, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book. The opinions expressed in this book belong to the author and are not necessarily those of Geek University Press.

Note that this is not an official book. Microsoft Corporation is in no way affiliated with this book or its content.

## **Trademarks**

Geek University is a trademark of Signum Soft, LLC, and may not be used without written permission.

## **Feedback Information**

At Geek University Press, our goal is to create in-depth technical books of the highest quality and value. Readers' feedback is a natural continuation of this process. If you have any comments about how we could improve our books and learning resources for you, you can contact us through email at [books@geek-university.com](mailto:books@geek-university.com). Please include the book title in your message. For more information about our books, visit our website at <http://geek-university.com>.

## **About this book**

This book teaches you how to program in Visual Basic 2015 - an object-oriented programming language designed by Microsoft. This book is designed for people without much experience in the world of programming. Although the book presumes some knowledge about computer systems in general, it is customized for beginners.

## **What will you learn**

You will learn the basics of programming in Visual Basic 2015 - what a variable is, how to perform arithmetic operations, the if statement, for loops, how to capture user input, how to develop GUI applications in Visual Basic, connect application to a database, and much more.

## **Source Code**

There is no need to type in all the code included in this book manually - we've made the source code available at <http://geek-university.com/uploads/vb.zip>.

## **About the author**

Mark Maslach is a software developer from Zagreb, Croatia. Mark has been programming in various programming languages since his teens. Mark is also the co-founder of Geek University, a popular online portal for online learning, available at <http://geek-university.com>. Mark can be reached at [mark@geek-university.com](mailto:mark@geek-university.com).

## **Contents at a Glance**

|  |                            |
|--|----------------------------|
| <a href="#"><u>Chapter 1 - Introduction to Visual Basic 2015</u></a> | <a href="#"><u>10</u></a>  |
| <a href="#"><u>Chapter 2 - Visual Basic 2015 Concepts</u></a>        | <a href="#"><u>24</u></a>  |
| <a href="#"><u>Chapter 3 - Basic Controls</u></a>                    | <a href="#"><u>38</u></a>  |
| <a href="#"><u>Chapter 4 - Operators</u></a>                         | <a href="#"><u>54</u></a>  |
| <a href="#"><u>Chapter 5 - Flow control</u></a>                      | <a href="#"><u>64</u></a>  |
| <a href="#"><u>Chapter 6 - Working with data structures</u></a>      | <a href="#"><u>76</u></a>  |
| <a href="#"><u>Chapter 7 - String Manipulation</u></a>               | <a href="#"><u>90</u></a>  |
| <a href="#"><u>Chapter 8 - Working with files</u></a>                | <a href="#"><u>98</u></a>  |
| <a href="#"><u>Chapter 9 - Advanced forms</u></a>                    | <a href="#"><u>108</u></a> |
| <a href="#"><u>Chapter 10 - Error Management</u></a>                 | <a href="#"><u>126</u></a> |
| <a href="#"><u>Chapter 11 - Function and Subs</u></a>                | <a href="#"><u>136</u></a> |
| <a href="#"><u>Chapter 12 - Classes and Objects</u></a>              | <a href="#"><u>150</u></a> |
| <a href="#"><u>Chapter 13 - Visual Basic and Databases</u></a>       | <a href="#"><u>160</u></a> |
| <a href="#"><u>Chapter 14 - Network programming</u></a>              | <a href="#"><u>176</u></a> |
| <a href="#"><u>Appendices</u></a>                                    | <a href="#"><u>196</u></a> |
| <a href="#"><u>Conclusion</u></a>                                    | <a href="#"><u>210</u></a> |

## **Table of Contents**

|  |                           |
|--|---------------------------|
| <a href="#"><u>Chapter 1 - Introduction to Visual Basic 2015</u></a>   | <a href="#"><u>10</u></a> |
| <a href="#"><u>What is Visual Basic 2015?</u></a>                      | <a href="#"><u>11</u></a> |
| <a href="#"><u>What is .NET Framework?</u></a>                         | <a href="#"><u>12</u></a> |
| <a href="#"><u>Install Visual Studio 2015</u></a>                      | <a href="#"><u>13</u></a> |
| <a href="#"><u>Getting to know the IDE</u></a>                         | <a href="#"><u>15</u></a> |
| <a href="#"><u>Writing your first .NET code</u></a>                    | <a href="#"><u>16</u></a> |
| <a href="#"><u>Chapter 2 - Visual Basic 2015 Concepts</u></a>          | <a href="#"><u>24</u></a> |
| <a href="#"><u>Visual Basic 2015 Variables</u></a>                     | <a href="#"><u>25</u></a> |
| <a href="#"><u>Rules for Naming Variables in Visual Basic 2015</u></a> | <a href="#"><u>25</u></a> |
| <a href="#"><u>Visual Basic 2015 Data types</u></a>                    | <a href="#"><u>26</u></a> |
| <a href="#"><u>String Variables</u></a>                                | <a href="#"><u>28</u></a> |
| <a href="#"><u>Working with dates</u></a>                              | <a href="#"><u>30</u></a> |
| <a href="#"><u>Comments</u></a>  | <a href="#"><u>33</u></a> |
| <a href="#"><u>Events</u></a>  | <a href="#"><u>34</u></a> |
| <a href="#"><u>Chapter 3 - Basic Controls</u></a>                      | <a href="#"><u>38</u></a> |
| <a href="#"><u>Labels</u></a>  | <a href="#"><u>39</u></a> |
| <a href="#"><u>LinkLabel</u></a>                                       | <a href="#"><u>40</u></a> |
| <a href="#"><u>TextBox</u></a>   | <a href="#"><u>40</u></a> |
| <a href="#"><u>Combo Box</u></a>                                       | <a href="#"><u>42</u></a> |
| <a href="#"><u>ListBox</u></a>   | <a href="#"><u>43</u></a> |
| <a href="#"><u>GroupBox</u></a>  | <a href="#"><u>45</u></a> |
| <a href="#"><u>RadioButton</u></a>                                     | <a href="#"><u>46</u></a> |

|  |           |
|--|-----------|
| <u>CheckBox</u>  | <u>48</u> |
| <u>Get user input</u>                                    | <u>49</u> |
| <u>Chapter 4 - Operators</u>                             | <u>54</u> |
| <u>Arithmetic operators</u>                              | <u>55</u> |
| <u>Comparison operators</u>                              | <u>56</u> |
| <u>Logical operators</u>                                 | <u>58</u> |
| <u>Assignment operators</u>                              | <u>60</u> |
| <u>Chapter 5 - Flow control</u>                          | <u>64</u> |
| <u>If...else statements</u>                              | <u>65</u> |
| <u>Select Case Statement</u>                             | <u>66</u> |
| <u>Specify a range of values in a <i>Case</i> clause</u> | <u>68</u> |
| <u>For Loop</u>  | <u>68</u> |
| <u>Do Loops</u>  | <u>70</u> |
| <u>Chapter 6 - Working with data structures</u>          | <u>76</u> |
| <u>Arrays</u>  | <u>77</u> |
| <u>Visual Basic 2015 Constants</u>                       | <u>79</u> |
| <u>Enumerations</u>                                      | <u>82</u> |
| <u>Structures</u>  | <u>84</u> |
| <u>Chapter 7 - String Manipulation</u>                   | <u>90</u> |
| <u>ToUpper method</u>                                    | <u>91</u> |
| <u>ToLower method</u>                                    | <u>92</u> |
| <u>The Trim Method</u>                                   | <u>92</u> |
| <u>The InStr function</u>                                | <u>93</u> |
| <u>The Val function</u>                                  | <u>94</u> |
| <u>The Contains Method</u>                               | <u>95</u> |
| <u>The Insert Method</u>                                 | <u>95</u> |

|  |                     |
|--|---------------------|
| <a href="#">Chapter 8 - Working with files</a>               | <a href="#">98</a>  |
| <a href="#">How to Open a File in Visual Basic 2015</a>      | <a href="#">100</a> |
| <a href="#">Writing to a Text File</a>                       | <a href="#">102</a> |
| <a href="#">Copy a File in VB .NET</a>                       | <a href="#">103</a> |
| <a href="#">Move a File with VB .NET</a>                     | <a href="#">104</a> |
| <a href="#">Delete a file in Visual Basic 2015</a>           | <a href="#">104</a> |
| <a href="#">Chapter 9 - Advanced forms</a>                   | <a href="#">108</a> |
| <a href="#">Adding Menus and Sub Menus in an Application</a> | <a href="#">109</a> |
| <a href="#">Underline Shortcut</a>                           | <a href="#">112</a> |
| <a href="#">Anchoring and Docking</a>                        | <a href="#">113</a> |
| <a href="#">Docking</a>                                      | <a href="#">116</a> |
| <a href="#">Creating multiple forms</a>                      | <a href="#">117</a> |
| <a href="#">Modal forms</a>                                  | <a href="#">119</a> |
| <a href="#">Chapter 10 - Error Management</a>                | <a href="#">126</a> |
| <a href="#">Design Time errors</a>                           | <a href="#">127</a> |
| <a href="#">Run Time errors</a>                              | <a href="#">128</a> |
| <a href="#">Logical errors</a>                               | <a href="#">128</a> |
| <a href="#">Breakpoints</a>                                  | <a href="#">128</a> |
| <a href="#">Try...Catch</a>                                  | <a href="#">130</a> |
| <a href="#">Chapter 11 - Function and Subs</a>               | <a href="#">136</a> |
| <a href="#">Sub procedures</a>                               | <a href="#">137</a> |
| <a href="#">ByVal and ByRef</a>                              | <a href="#">139</a> |
| <a href="#">Functions</a>                                    | <a href="#">141</a> |
| <a href="#">Modules in Visual Basic 2015</a>                 | <a href="#">142</a> |
| <a href="#">Chapter 12 - Classes and Objects</a>             | <a href="#">150</a> |
| <a href="#">Create a Class</a>                               | <a href="#">151</a> |



|   |                            |
|---|----------------------------|
| <a href="#"><u>Create Properties in VB .NET Classes</u></a>                     | <a href="#"><u>153</u></a> |
| <a href="#"><u>Create Methods in your VB .NET Classes</u></a>                   | <a href="#"><u>155</u></a> |
| <a href="#"><u>Chapter 13 - Visual Basic and Databases</u></a>                  | <a href="#"><u>160</u></a> |
| <a href="#"><u>Database Terminology</u></a>                                     | <a href="#"><u>161</u></a> |
| <a href="#"><u>SQL Server 2014 Installation</u></a>                             | <a href="#"><u>161</u></a> |
| <a href="#"><u>Connecting an application to an SQL Server 2014 Database</u></a> | <a href="#"><u>167</u></a> |
| <a href="#"><u>Previewing the Contents of a Dataset</u></a>                     | <a href="#"><u>171</u></a> |
| <a href="#"><u>Chapter 14 - Network programming</u></a>                         | <a href="#"><u>176</u></a> |
| <a href="#"><u>Socket programming</u></a>                                       | <a href="#"><u>177</u></a> |
| <a href="#"><u>Writing a simple TCP/IP server</u></a>                           | <a href="#"><u>177</u></a> |
| <a href="#"><u>Writing a simple TCP/IP client</u></a>                           | <a href="#"><u>180</u></a> |
| <a href="#"><u>HTTP protocol</u></a>  | <a href="#"><u>185</u></a> |
| <a href="#"><u>FTP protocol</u></a>   | <a href="#"><u>187</u></a> |
| <a href="#"><u>SMTP protocol</u></a>  | <a href="#"><u>191</u></a> |
| <a href="#"><u>Appendices</u></a>   | <a href="#"><u>196</u></a> |
| <a href="#"><u>Appendix I: Deploying a Windows-based Application</u></a>        | <a href="#"><u>196</u></a> |
| <a href="#"><u>Appendix II: Cheat Sheet</u></a>                                 | <a href="#"><u>202</u></a> |
| <a href="#"><u>Conclusion</u></a>   | <a href="#"><u>210</u></a> |

# **Chapter 1 - Introduction to Visual Basic 2015**

## **IN THIS CHAPTER**

**Learning about Visual Basic 2015**

**Installing Visual Studio 2015**

**Getting to know the IDE**

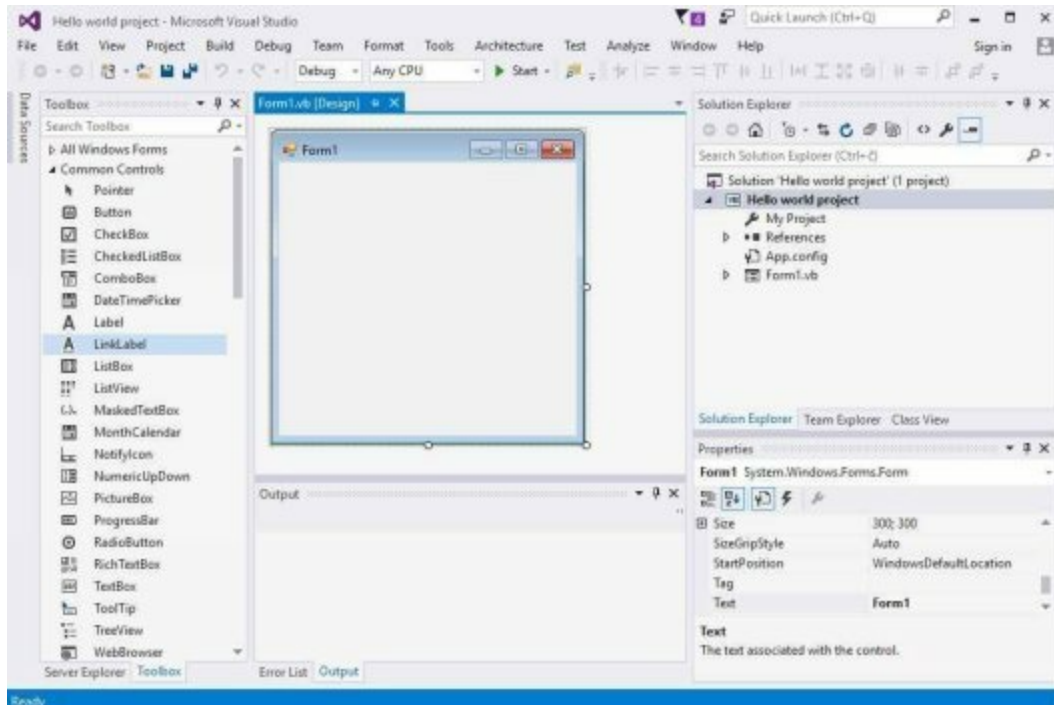
# What is Visual Basic 2015?

**Visual Basic 2015** is an object-oriented programming language designed by Microsoft and implemented on the .NET Framework. It is the latest version of Visual Basic, introduced by Microsoft in 2015. This version includes many new features, particularly for building mobile and web applications.

Visual Basic is a high-level programming language that allows you to write Windows, mobile, and web applications. Simply put, a programming language is a special kind of language used to develop software that will be executed on a computer. Visual Basic, as any other programming language, has its own syntax, keywords and rules for instructing computers to perform specific tasks. Each program on your computer has been written using a programming language (in many cases, that language is Visual Basic!).

Visual Basic 2015 is implemented by Microsoft's .NET Framework and has full access to all the libraries in the .NET Framework. You've probably already encountered .NET Framework because an application asked you to install it. I will describe .NET framework in more details in the following section.

Visual Basic code can be written in any text editor (such as Notepad). However, most of the time programmers use **Visual Studio 2015 Integrated Development Framework**, which is a software from Microsoft that enables you to write, test, run, and debug your code in a nice GUI environment. Below is the picture of the Visual Studio 2015 IDE we will be using throughout this book:



## NOTE

It's also possible to run Visual Basic 2015 programs on Mono, the open-source alternative to .NET, not only under Windows, but even on Linux or Mac OS X.

## What is .NET Framework?

**.NET Framework** is a software framework developed by Microsoft that runs primarily on Microsoft Windows. You can think of it as is a platform that provides tools and libraries you need to build and run networked application and web services.

.NET Framework consists of two parts:

- **The Common Language Runtime (CLR)** - the runtime environment of the .NET Framework. It provides a virtual machine-like sandbox in which applications run and supplies services such as such as memory and exception management, debugging and profiling, and security.
- **The .NET Framework Class Library (FCL)** - a huge collection of language-independent classes that are arranged into a logical grouping,

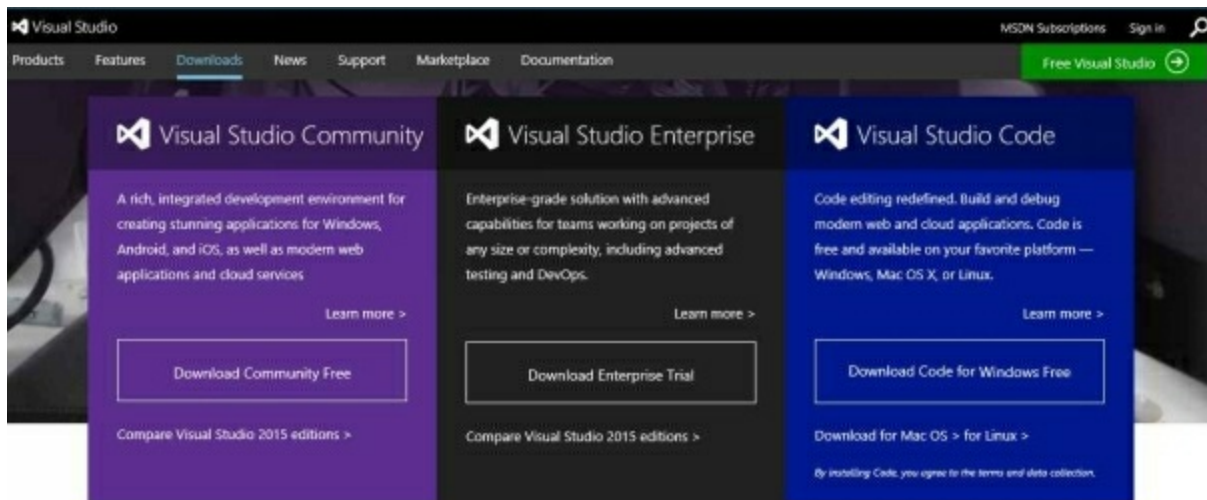
according to their functionality and usability. Some of the services provided by FCL include accessing system services, interaction with databases, establishing network connections, and much more.

## Install Visual Studio 2015

Before you start programming, you will need to download and install Visual Studio 2015, which is a development environment where you will write your code. To do this, browse to the official download site and download the latest version for Windows:

<http://www.visualstudio.com/en-us/downloads/visual-studio-2015-downloads-vs.aspx>

The **Download** page should appear:



I recommend that you download the 90-day trial of **Visual Studio 2015 Enterprise** version. Click on the **Download** button and the installer will be downloaded to your hard drive. Double-click the file to start the installation process.

### NOTE

As of time of writing this book, the current version of Visual Studio was 2015. All of the examples in the book use this version of Visual Studio. However, if

you have an older version of Visual Studio or a newer version appears, some of the example might look a little bit different in your environment.

The installation is pretty much straightforward and requires you to specify only a few things. First, you need to choose where to install the program and the type of the installation. I recommend that you leave all the default options and click the **Install** button:



The installation process should start:



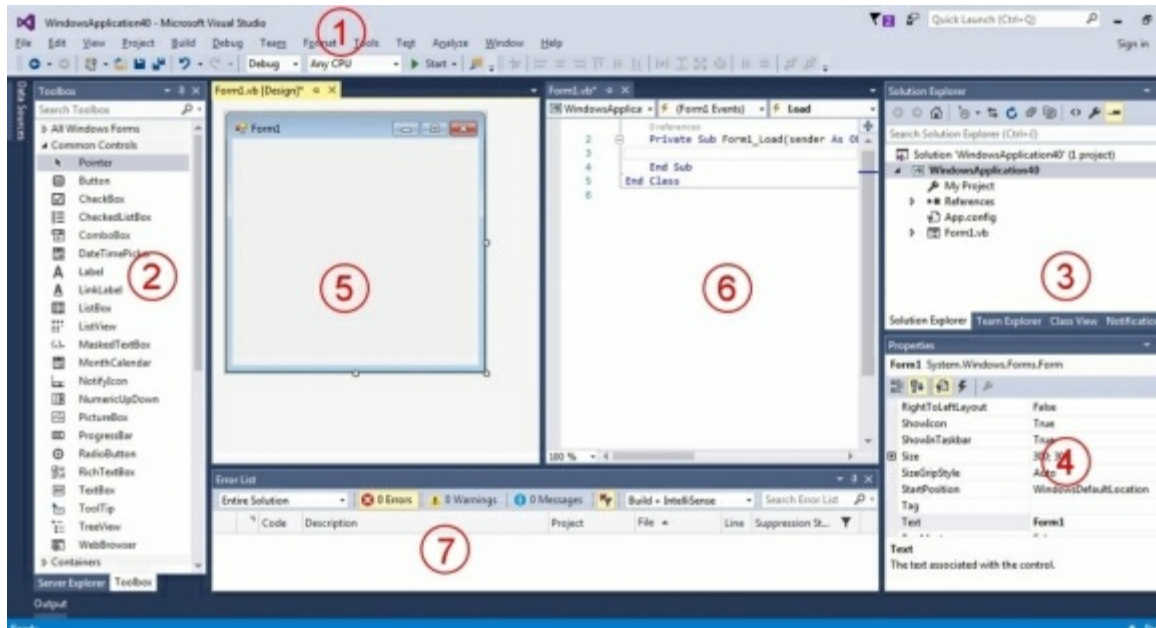
The installation could take quite some time, depending on the speed of your Internet connection. After completing the installation process, you might see an option to restart your computer to complete the installation process. If that is the case, press the *Restart Now* button to restart your computer.

## Getting to know the IDE

In this section you will learn a little bit about the **Visual Studio**, what it does, how it looks like, and what it is used for.

**Visual Studio 2015** comes with lots of new enhancements and features. Some of these features widely enhance the productivity of application development. Let's get our hands dirty and dive into the Visual Studio IDE features and components to help you get started with using Visual Basic.

The following image shows the Visual Studio 2015 IDE with an open project:



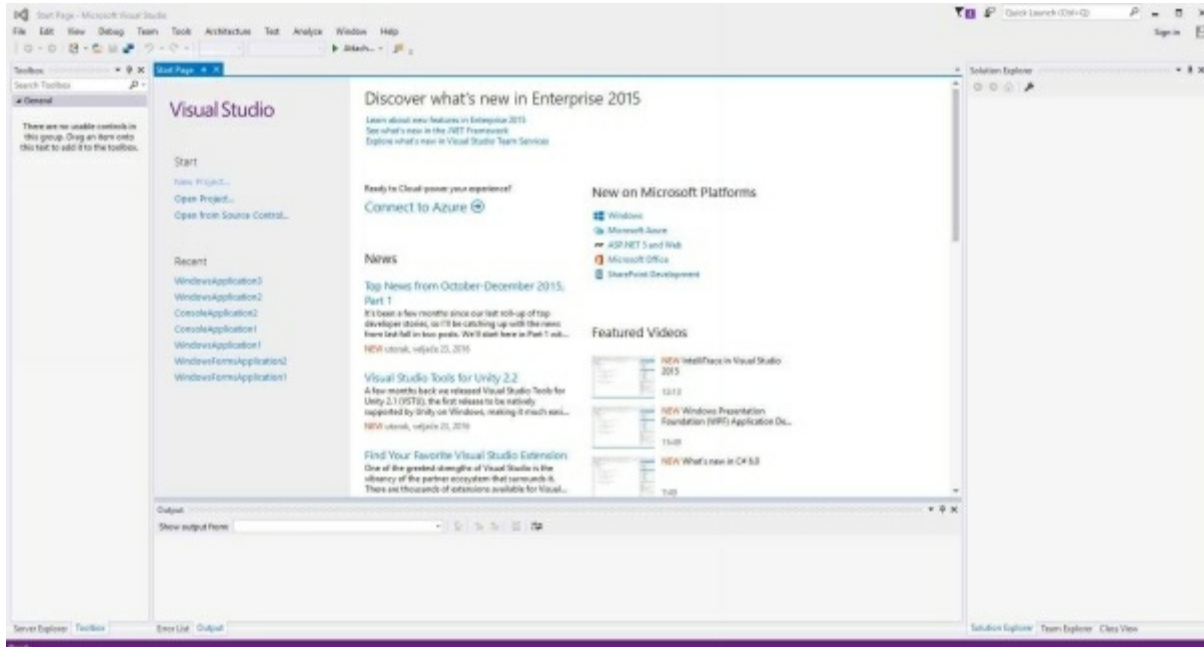
Visual Studio IDE has 7 main parts:

1. **Menu and toolbar** - the Menu contains group of related commands that, when selected, cause the Visual Studio to perform specific actions. (e.g. open a window, save a project, exit a program). For example, new projects are created by selecting **File > New > Project**.
2. **Toolbox window** - the Toolbox window contains all objects you can use to build your application's GUI. The control objects in the Toolbox are organized in a various tabs. The Common Controls tab contains the icons for the most common Windows controls, while the All Windows Controls tab contains all the controls you can place on your form.
3. **Solution Explorer** - once you open up a project, the Solution Explorer shows the main files of your application. You can use it to quickly move from one part of the application to the other.
4. **Properties window** - this window allows you to change the control's properties at a design time environment.
5. **Working area** - the central part of the IDE is your working area, where you design the user interface.
6. **Code window** - the place where you write your code.
7. **Debug window** - Visual Studio also includes a debugger, so you can run your applications in development time, without having to make executables or distribution packages.

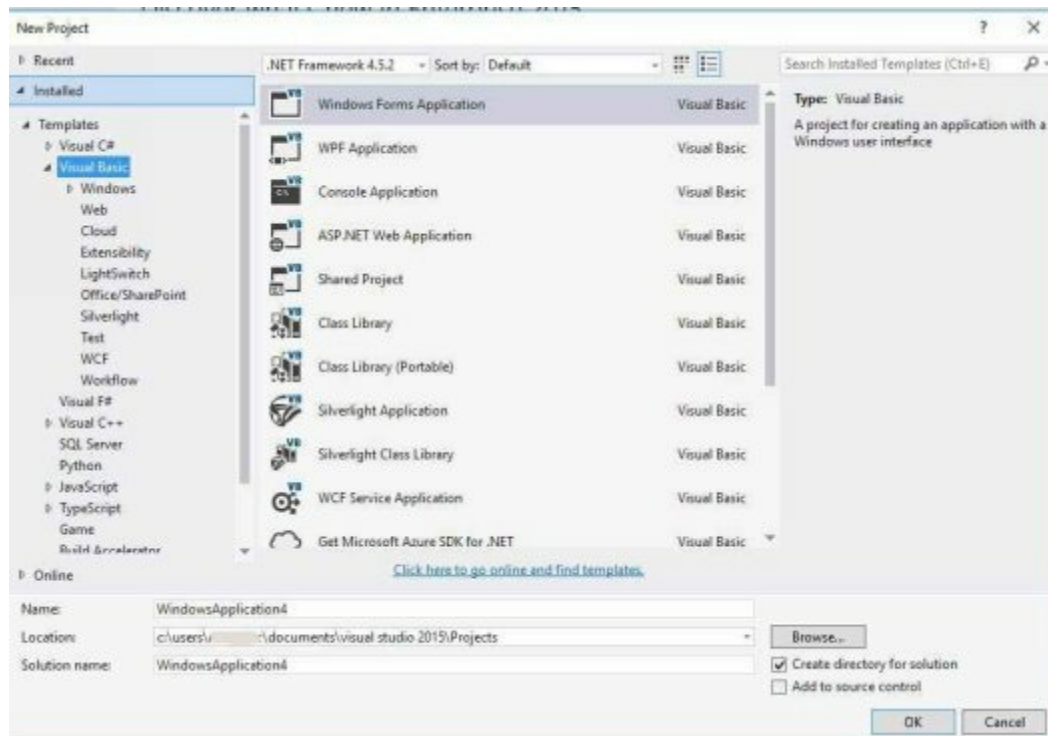


# Writing your first .NET code

Now that we've learned a little bit about the IDE, let's use it to write our first program. Start your Visual Studio 2015 and you should get a screen like this:



From the *File* menu select *New Project*. The following dialog box should appear:

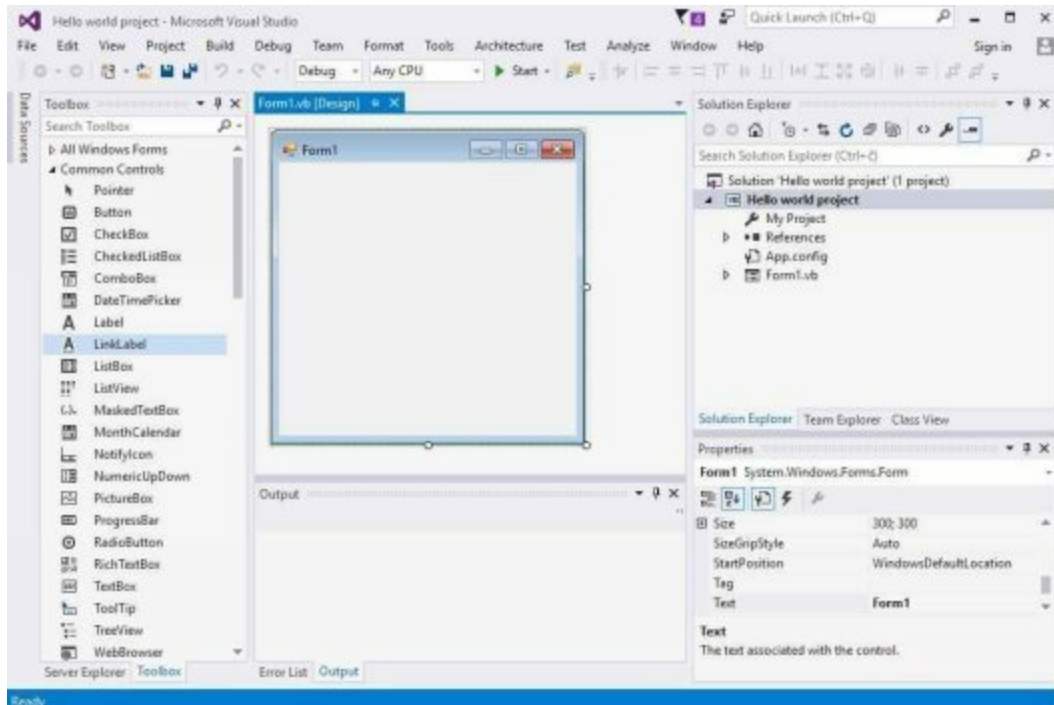


If not already selected, select *Visual Basic* under *Templates*. Choose *Windows Forms Application* and change the default project name to *Hello world project*. We've selected the *Windows Forms Application* type because this type provides us with a form in which we can easily add user interface elements, such as buttons, labels, menus, and other GUI elements.

## NOTE

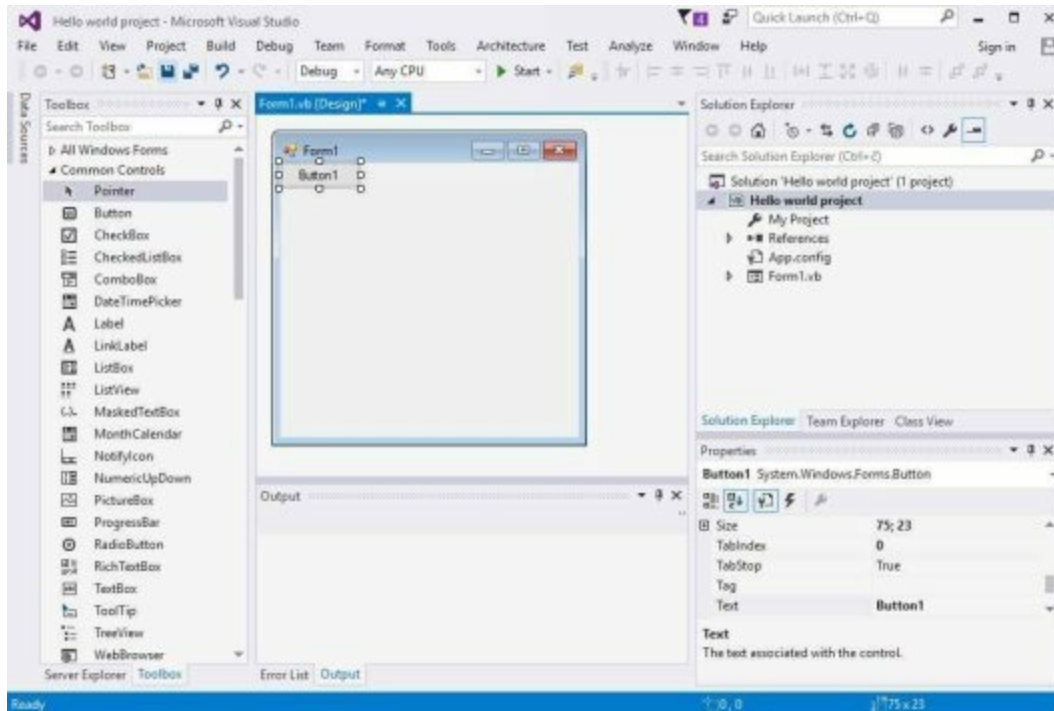
A little bit of trivia - we've named our first application *Hello world project* because of a custom in the programming world. The first program that a novice programmer writes usually simply outputs the *Hello World* message.

Click the *OK* button and the Visual Basic design environment will open. It should look something like this:



As you can see from the picture above, we've got a screen similar to the one we've described in the previous section. In the middle of the screen is the **Design** area with our blank form. Here we can add UI elements from the toolbox on the left. So let's add some elements to our form.

We will start our example by adding a button to your new form. Locate the *Button* tool from the toolbox and double-click on it. The button should be added to the top left position of your form:



In the bottom right corner you should see the *Properties* window that contains various button settings. Find and change the following properties of *Button1* to:

**Name:** btnHelloWorld

**Text:** Display message

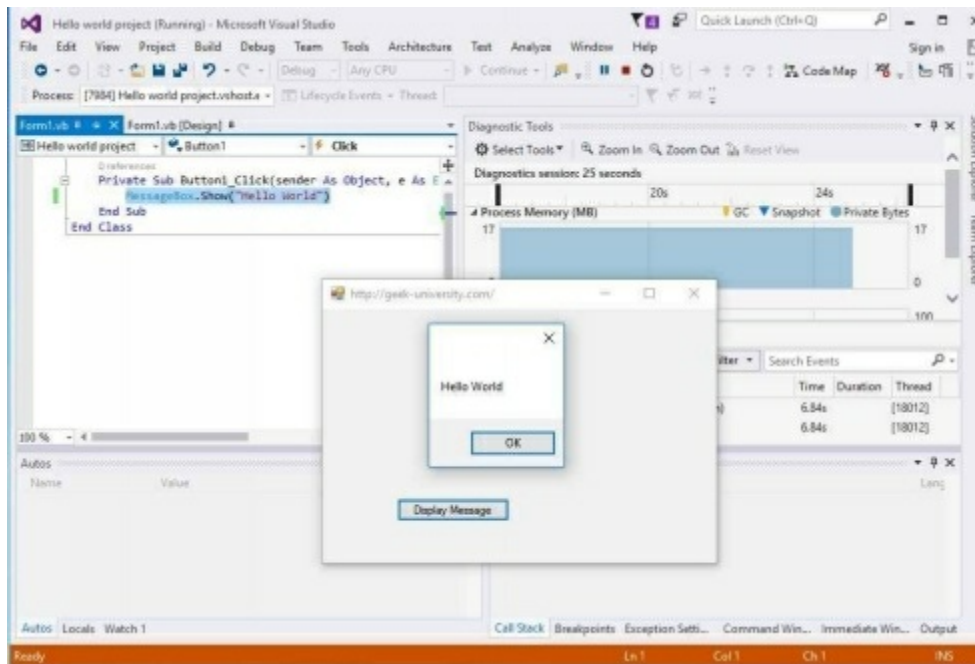
Now, double click your new Display message button, and add the following code:

```
MessageBox.Show("Hello World")
```

To run the program, do one of the following:

- From the menu bar, click **Debug**
- From the drop down menu, click **Start Debugging**
- Press the **F5** key on your keyboard

You have now created your first program! When executed, it should look something like this:



Congratulations! You have just written your first Visual Basic 2015 application.

# **Chapter 2 - Visual Basic 2015 Concepts**

## **IN THIS CHAPTER**

**Learning about variables and their naming conventions**

**Strings and dates**

**Using comments in your code**

# Visual Basic 2015 Variables

In programming, variables are objects - storage areas used to store information that can later be referenced and manipulated. You can consider a variable as a sort of a container where you store an information that can later be accessed using the variable name.

The variable is determined by its:

1. **Scope** - the scope level determines from where the variable can be accessed (e.g. private, public ...).
2. **Data type** - can be a string, boolean, integer...
3. **Accessibility** - determines which code in other modules can access the variable.
4. **Lifetime** - determines how long the variable value will be valid.

## Rules for Naming Variables in Visual Basic 2015

There are certain rules that need to be followed when naming your variables

1. The name must start with a letter or an underscore.
2. The name can contain only letters, numbers, and the underscore character. No punctuation characters, special characters, or spaces are allowed in the name.
3. Although the name can contain thousands of characters, 32 characters is the recommended maximum number of characters to use.
4. The name cannot be a reserved word, such as Sub or Double.

Now, let's examine the following code example:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
Dim number1 As Integer
number1 = 5
End Sub
```

Let's explain the code above:

**Dim** - Short for Dimension. The Dim statement is used for variable declaration. The Dim statement is used at class, module, structure, procedure or block level.

**number1** - this is the variable, our storage area. After the **Dim** statement, the name of variable is specified.

**As Integer** - we are indicating Visual Basic 2015 that variable is going to be a number (integer).

**number1 = 5** - the = sign means **assign value**. We are telling Visual Basic 2015 to assign a value of 5 to the variable **number1**.

## Visual Basic 2015 Data types

Variables in Visual Basic 2015 can be of a different data type. The data type of a variable is important because it specifies the kind of information that can be stored inside a variable. For example, to store numeric information, you need a variable of the type numeric. You can't store a string in a variable designed to store numeric information. You will need to create a variable of the string type to store a string.

Here is a table of available data types in Visual Basic 2015:

| Data Type | Storage Allocation               | Value Range              |
|-----------|----------------------------------|--------------------------|
| Boolean   | Depends on implementing platform | True or False            |
|           |                                  | 0 through 255 (unsigned) |



|          |  |   |
|----------|--|---|
| Byte     | 1 byte   |   |
| Char     | 2 bytes  | 0 through 65535 (unsigned)  |
| Date     | 8 bytes  | 0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999  |
| Decimal  | 16 bytes   | 0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal |
| Double   | 8 bytes  | -1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values<br>4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values                |
| Integer  | 4 bytes  | -2,147,483,648 through 2,147,483,647 (signed)   |
| Long     | 8 bytes  | -9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)  |
| Object   | 4 bytes on 32-bit platform<br>8 bytes on 64-bit platform | Any type can be stored in a variable of type Object   |
| SByte    | 1 byte   | -128 through 127 (signed)   |
| Short    | 2 bytes  | -32,768 through 32,767 (signed)   |
| Single   | 4 bytes  | -3.4028235E+38 through -1.401298E-45 for negative values;<br>1.401298E-45 through 3.4028235E+38 for positive values   |
| String   | Depends on implementing platform                         | 0 to approximately 2 billion Unicode characters   |
| UInteger | 4 bytes  | 0 through 4,294,967,295 (unsigned)  |
| ULong    | 8 bytes  | 0 through 18,446,744,073,709,551,615 (unsigned)   |
|          | Depends on   | Each member of the structure has a range  |

| User-Defined | implementing platform | determined by its data type and independent of the ranges of the other members |
|--------------|-----------------------|--|
| UShort       | 2 bytes               | 0 through 65,535 (unsigned)  |

The following example demonstrates the usage of some of the variable data types:

```
Dim val1 As Byte
```

```
Dim val2 As Integer
```

```
Dim val3 As Single
```

```
Dim val4 As Double
```

```
Dim val5 As Date
```

```
Dim val6 As Char
```

```
Dim content As String
```

```
Dim val7 As Boolean
```

```
val1 = 1
```

```
val2 = 1234567
```

```
val3 = 0.11223344556677881
```

```
val4 = 0.11223344556677881
```

```
val5 = Today
```

```
val6 = "a"
```

```
content = "Me"
```

```
val7 = True
```

```
Console.WriteLine(val1)
```

```
Console.WriteLine(val2)
```

```
Console.WriteLine(val3)
```

```
Console.WriteLine(val4)
```

```
Console.WriteLine(val5)
```

```
Console.WriteLine(val6)
```

```
Console.WriteLine(content)
Console.WriteLine(val7)
```

Here is the output of the code above:

```
1
1234567
0.11223344556677881
0.11223344556677881
Today
a
Me
True
```

## String Variables

**String** variables are nothing more than text. Strings in Visual Basic can be used to store a set of characters.

To define a string, you simply type the characters within the double quotes. For example, if the information we want to store in our variable is a name, first we need to set up the variable like this:

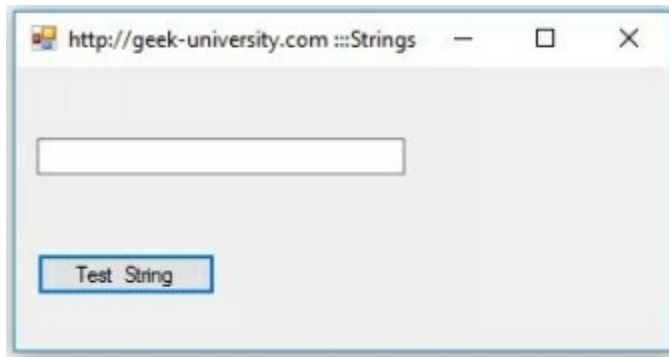
```
Dim FirstName As String
```

We have started our code with *Dim* keyword. With this word we declare or tell Visual Basic that you are setting up a variable. With *FirstName As String*, we are telling Visual Basic that the variable is going to be a string. Once we've declared our variable, we can store a name of person into it using the equal sign:

```
FirstName = "Jack"
```

Let's start our new example by adding one button and one textbox on our form.

You should get the screen like this:



Change the properties set in the bottom right corner for these controls as follows:

**(Button)**

Name: btnName

Text: Test String

**(TextBox)**

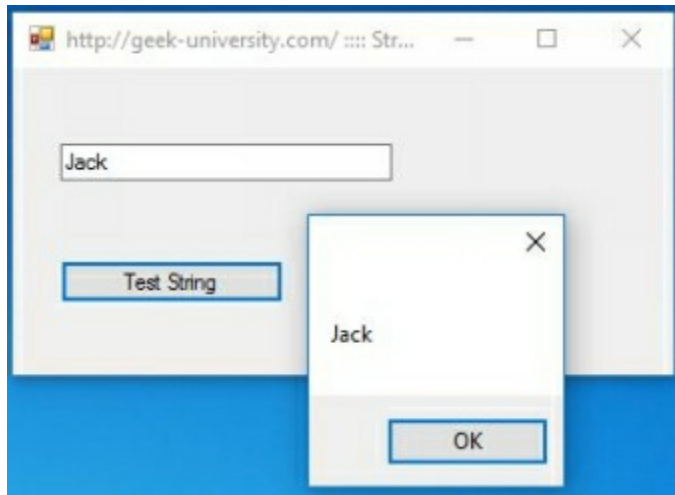
Name: txtName

Text: leave blank

Now, double-click the button. The Code editor should open. This is where you write your code. Add the following code:

```
Private Sub btnName_Click(sender As Object, e As EventArgs) Handles btnName.Click
    Dim firstName As String
    firstName = txtName.Text
    MessageBox.Show(firstName)
End Sub
```

Run your program to test it out. Click inside a TextBox and add a name. Next, click on the button and you should get something like this:



As you can see from the output above, the **MessageBox** control displayed the name you've entered in the **TextBox**. We will be using this control a lot in the book. For now, just remember that this control displays the string you have passed in the message box.

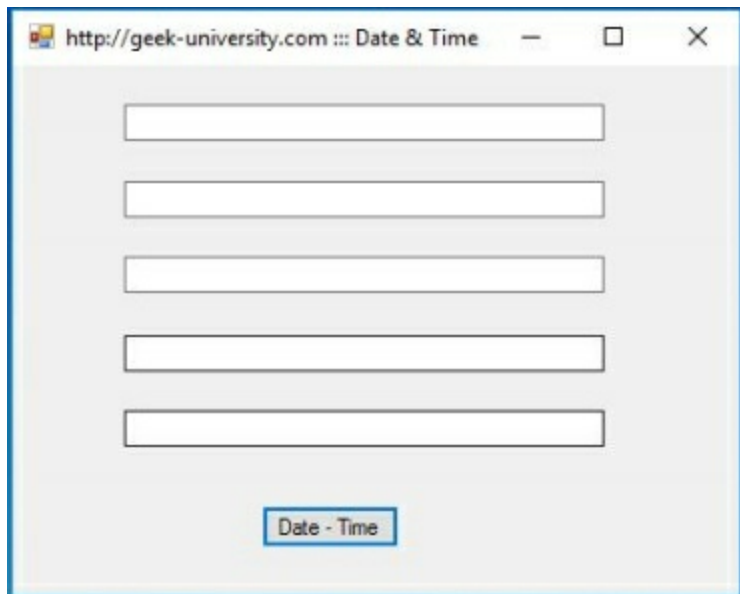
## Working with dates

Sometimes, you need the ability to manipulate the date and time. In Visual Basic 2015, date and time can be formatted using predefined or user-defined formats.

Here are the predefined formats of date and time:

| Format                      | Display                                  |
|-----------------------------|--|
| Format(Now, "General Date") | Formats the current date and time        |
| Format(Now, "Long Date")    | Display the current date in long format  |
| Format(Now, "short Date")   | Display the current date in short format |
| Format(Now, "Long Time")    | Display the current time in long format  |
| Format(Now, "Short Time")   | Display the current time in short format |

Here is an example. Add five new TextBoxes and one button to your Form. The form should look like this:



Open up the editor for your *Date - Time* button and add the following code to it:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    TextBox1.Text = Format(Now, "General Date")
    TextBox2.Text = Format(Now, "Long Date")
    TextBox3.Text = Format(Now, "short Date")
    TextBox4.Text = Format(Now, "Long Time")
    TextBox5.Text = Format(Now, "Short Time")
End Sub
```

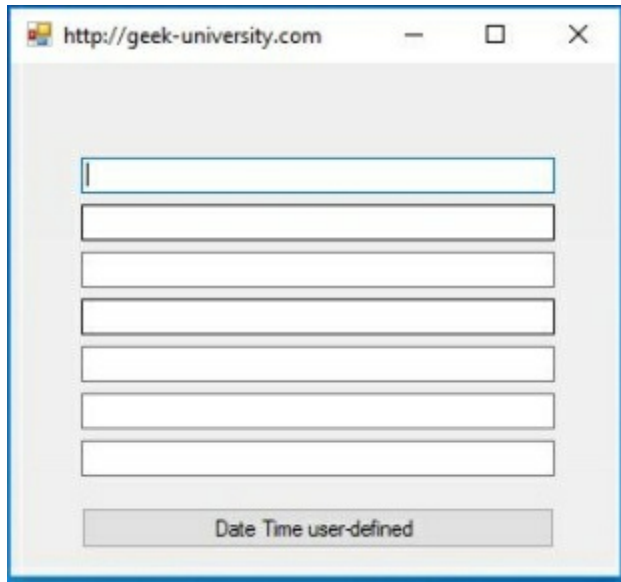
Run your program and click the button. Your form should look like this:

You can also use the following user-defined formatting functions to define the format you want. This can be useful if you want to display international formats for numbers, dates, and times. Here is a list of the formatting functions:

| Format                             | Explanation   |
|------------------------------------|---|
| Format(Now,"M")                    | Displays current month and date                                   |
| Format(Now,"MM")                   | Displays current month in double digitals                         |
| Format(Now,"MMM")                  | Displays abbreviated name of the current month                    |
| Format(Now,"MMMM")                 | Displays full name of the current month                           |
| Format(Now,"dd/MM/yyyy")           | Displays current date in the day/month/year format                |
| Format(Now,"MMM,d,yyyy")           | Displays current date in the Month, Day, Year Format              |
| Format (Now, "h:mm:ss tt")         | Displays current time in hour:minute:second format and show am/pm |
| Format (Now, "MM/dd/yyyy h:mm:ss") | Displays current date and time in hour:minute:second format       |

Let's start by creating a new project in our Visual Studio 2015. Before we start coding, we will add seven textboxes and one button on our form.

When you are finished, your form should look like this one below:



Now, double click the button and add the following code:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
    TextBox1.Text = Format(Now, "M")
```

```
    TextBox2.Text = Format(Now, "MM")
```

```
    TextBox3.Text = Format(Now, "MMM")
```

```
    TextBox4.Text = Format(Now, "MMMM")
```

```
    TextBox5.Text = Format(Now, "dd/MM/yyyy")
```

```
    TextBox6.Text = Format(Now, "MMM,d,yyyy")
```

```
    TextBox7.Text = Format(Now, "MM/dd/yyyy h:mm:ss tt")
```

```
End Sub
```

Press F5 key on your keyboard to launch your program. When the form loads, click the **Date Time user-defined** button, and you should get the something like this:



A screenshot of a web browser window with the address bar showing `http://geek-university.com`. The browser window contains a form with seven text input fields stacked vertically. The first six fields contain the following text: "25. veljače", "02", "vlj", "veljača", "25.02.2016", and "vlj,25,2016". The seventh field contains the date and time "02.25.2016 5:34:55". Below the input fields is a button with the text "Date Time user-defined".

## Comments

You can use comments in your Visual Basic source code in order to document it. With comments, you can remind yourself what a specific portion of the code does and make your code more accessible to other developers.

Since comments are written inside a file that is going to be executed, Visual Basic needs some way to determine that the text you write is a comment, and not a command that needs to be executed. The comment in Visual Basic starts with a **single quotation mark** (') and everything until the end of the line is part of the comment and ignored by Visual Basic.

It is good programming practice to use comments to make your code clear and understandable:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    MessageBox.Show("Hello World") 'This is comment

End Sub
```

## NOTE

You can't use line continuation characters to make a multi-line comment nor you can have block comment as they are not supported in Visual Basic 2015.

## Events

Visual Basic 2015 is an event driven language. This means the flow of execution is determined by the external occurrences known as **events**.

An event is a signal that informs an application that something has occurred. For example, when a user clicks a button on a form, the form can raise a **Click event** and invoke a procedure that handles the event. Events also allow separate tasks to communicate. Say, for example, that your application performs a sort task separately from the main application. If a user cancels the sort, your application can send a cancel event instructing the sort process to stop.

After creating your application's interface, you can begin writing the Visual Basic code that instructs the controls to respond to the user's actions. Those actions - such as clicking or double-clicking - are called events.

You can tell a control how to respond to an event by writing an event procedure, which is a set of Visual Basic instructions that are processed only when the event occurs.

Here is an example. Add a new button to a form and double click it. You should see the following code in the code editor:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
End Sub
```

Here is a breakdown of the syntax:

- **Private Sub** - a private subroutine.
- **Button1\_Click** - the name of the sub.
- **Button1.Click** - an event.

In this example, you will write a **Click** event procedure for the Message button, which should pop up a message when it is clicked:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click  
    MsgBox("Some message")  
End Sub
```

Run your application by pressing F5 on your keyboard and click on the button. The click event should be invoked and the message "Some message" should be displayed.

# **Chapter 3 - Basic Controls**

## **IN THIS CHAPTER**

**Using GUI elements**

**Adding buttons, labels, and checkboxes**

**Getting input from the user**

The GUI part of Visual Basic 2015 programming involves the use of the various controls available in the toolbox. With controls come properties (a set of variables that describe the appearance, behavior, and other aspects of the control), methods (procedures built into a control that tell the control how to do things), and events (actions that occur in your program). In this section we will learn about the basic controls used to interact with the user.

## Labels

Label controls are used to display text. They are used to identify objects on a form - provide a description of what a certain control will do if clicked, for example - or, at run time, they can display information in response to an event or process in your application.

Labels are used in many instances and for many different purposes. Most commonly they are used to label controls that don't have their own *Caption* properties. For example, you can use the label control to add descriptive labels to text boxes, list boxes, and combo boxes. Labels can also be used to add descriptive text to a form to provide the user with help information.

### NOTE

You can write code that changes the text displayed by a label control in response to events at run time. For example, if your application takes a few minutes to process a change, you can display a processing-status message in a label.

Create the following form with one label and a button:



Now, type the code below inside the button click event:

```
Label1.ForeColor = Color.Red
```

Next, press F5 and click on the button to see what will happen:



By pressing the button, the color of the label has changed.

## LinkLabel

The **Windows Forms LinkLabel** control enables you to add Web-style links to Windows Forms applications. You can use the LinkLabel control for everything that you can use the Label control for; you also can set part of the text as a link to an object or Web page

For this example add a link label control to the form and set these properties:

- Name: `lnkUrl`
- Text: Click here to visit: <http://geek-university.com>

Your form should look like something like this:



Next, paste the following code inside the **linkLabel** click event:

```
System.Diagnostics.Process.Start("http://geek-university.com")
```

Press F5 to run your application. This above example displays a web page in your default browser when you click on the link.

## TextBox

Text boxes are used to get input from the user or to simply display text. The **TextBox** control is generally used for editable text and can display multiple lines, wrap text to the size of the control, and add basic formatting.

### NOTE

To display multiple types of formatted text, use the **RichTextBox** control.

The text displayed by the control is contained in the **Text** property. By default, you can enter up to 2048 characters in a text box. If you set the **MultiLine** property to true, you can enter up to 32 KB of text. The **Text** property can be set at design time with the Properties window, at run time in code, or by user input at run time. The current contents of a text box can be retrieved at run time by reading the **Text** property.

The code below sets text in the control at run time:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles  
MyBase.Load
```

```
    TextBox1.Text = "http://geek-university.com"
```

```
End Sub
```

Press F5 on your keyboard and you should get something like this:



You can also collect the input value from a TextBox control to a variable like this way:

```
Dim var As String  
var = TextBox1.Text  
MessageBox.Show(var)
```



## Combo Box

A combo box control lets the user choose one of several choices from a drop-down list. In addition to display and selection functionality, the ComboBox also provides features that enables you to efficiently add items to the ComboBox.

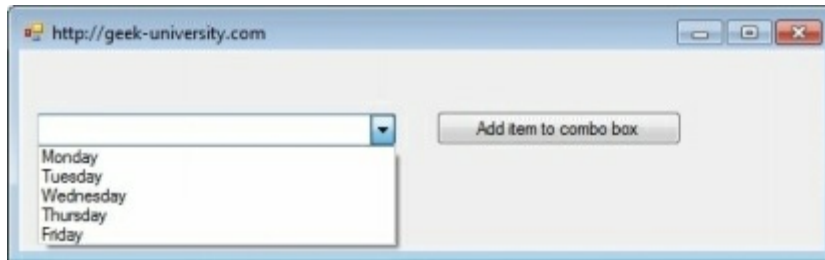
In this example we will fill a combo box with items, get the selected items in the combo box, and then show them in a textbox.



Add the following code in the form load event to fill the combobox with the items:

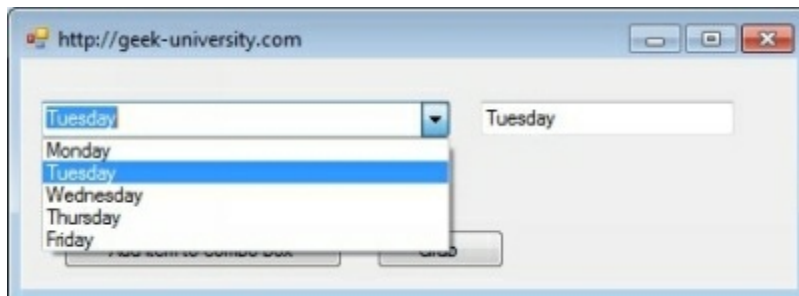
```
Dim irange As String() = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"}
```

```
ComboBox1.Items.AddRange(irange)
```



We are using the *AddRange* method to fill the listbox with contents of a string array from the FormLoad event. To grab the displayed item from the combo box to a TextBox control, type the following code inside the button click event:

```
TextBox1.Text = ComboBox1.Text
```



Now, when you press the *Grab* button, you should get the selected day of the week.

## NOTE

You can remove items from a combobox using the *clear* method:

```
ComboBox1.Items.Clear()
```

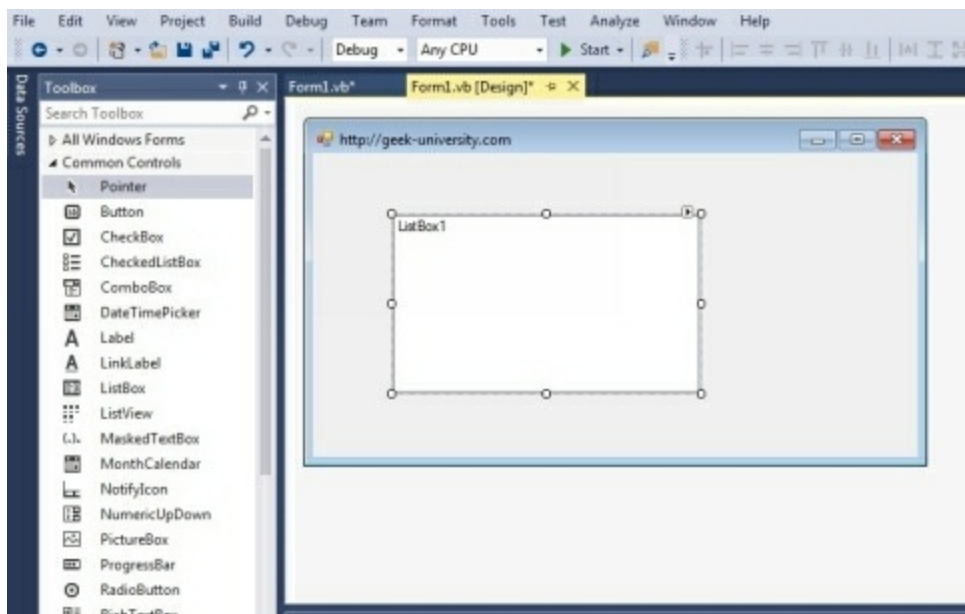
## ListBox

A **ListBox** control displays a list of choices which the user can select. It can interact with other controls, including the **Button** controls. This control automatically displays scrollbars, unless the **MultiColumn** property is set to true. If this is the case, the choices are displayed in multiple columns:

|                     |         |
|---------------------|---------|
| Modifiers           | Friend  |
| MultiColumn         | False   |
| RightToLeft         | No      |
| ScrollAlwaysVisible | False   |
| SelectionMode       | One     |
| Size                | 120; 95 |
| Sorted              | False   |
| TabIndex            | 0       |
| TabStop             | True    |
| Tag                 |         |
| UseTabStops         | True    |

Items

Let's create a list box by dragging a **ListBox** control from the Toolbox and dropping it on the form.



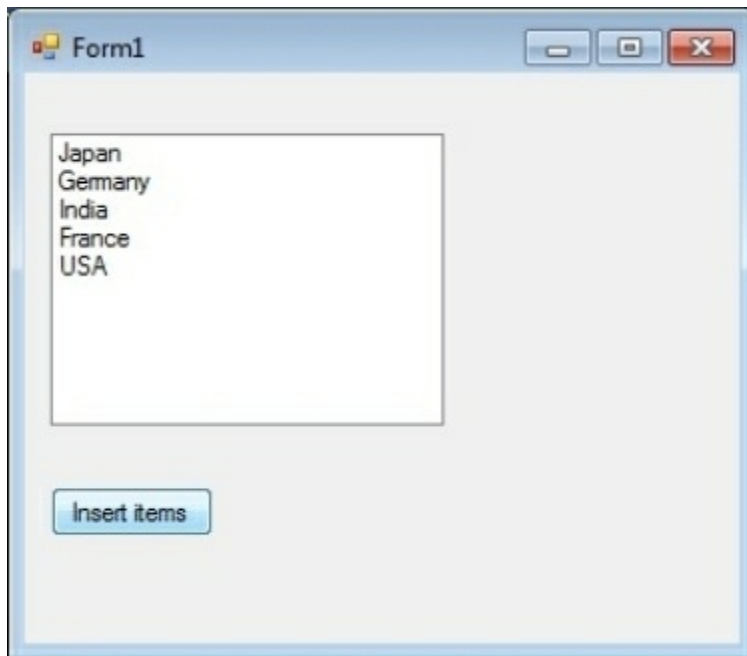
You can populate the list box with text either from the properties window or at runtime.

To insert items to a list box, use the *AddRange* method. Add a button to your form, double-click it and add the following code inside the click event:

```
Dim listboxrange As String() = {"Japan", "Germany", "India", "France",  
"USA"}
```

```
ListBox1.Items.AddRange(listboxrange)
```

Run your program and click the *Insert* button. You should get something like this:



If you want to add item at the end of an unsorted list box, you can use *Add* method:

```
ListBox1.Items.Add("UK")
```

Finally, you can remove items from a combobox using the *Clear* method:

```
ListBox1.Items.Clear()
```

# GroupBox

The GroupBox controls are used to provide an identifiable grouping for other controls. Typically, you use group boxes to subdivide a form by function. Grouping all options in a group box gives the user a logical visual cue, and at design time all the controls can be moved easily - when you move the single GroupBox control, all its contained controls move, too.

The GroupBox control is similar to the Panel control; however, only the GroupBox control displays a caption, and only the Panel control can have scroll bars. The Group box's caption is defined by the Text property.

In the following example we are going to use GroupBox to group the text boxes and Labels inside it. Here is what this GroupBox will look like when it's finished:

The image shows a screenshot of a Windows application window titled "Form1". Inside the window, there is a GroupBox control. The GroupBox has a caption "Customer Info" at the top left. Below the caption, there are four text boxes, each preceded by a label: "Customer Name:", "Address:", "Phone:", and "Email:". The text boxes are empty and have a standard Windows input style. The GroupBox itself has a light gray background and a thin border.

## NOTE

One of the best things about GroupBox control is that if you set its Enabled

property to False, then the all controls it contains are also disabled, and it is a perfect way to enable or disable a group of controls all at once.

## RadioButton

With a radio button users can make a choice among a set of mutually exclusive but otherwise related options. Users can choose only a single option.

The RadioButton control's most useful event is the click event, which occurs when the user clicks the control and GotFocus, which occurs when the control is checked or unchecked either because the user clicked a RadioButton in the group or because the code changed the button's state.

For example, say that you want to let the user select one of the following values: Red, Green, or Blue. You could add three RadioButtons to your form with those captions, and then when the user clicks on a button, Visual Basic 2015 selects it and deselects the others.



Consider the following code example:

```
If RadioButton1.Checked = True Then
```

```
    MessageBox.Show("You have selected Red!")
```

```
Else
```

```
    If RadioButton2.Checked = True Then
```

```
        MessageBox.Show("You have selected Green!")
    Else
        If RadioButton3.Checked = True Then
            MessageBox.Show("You have selected Blue!")
        End If
    End If
End If
```

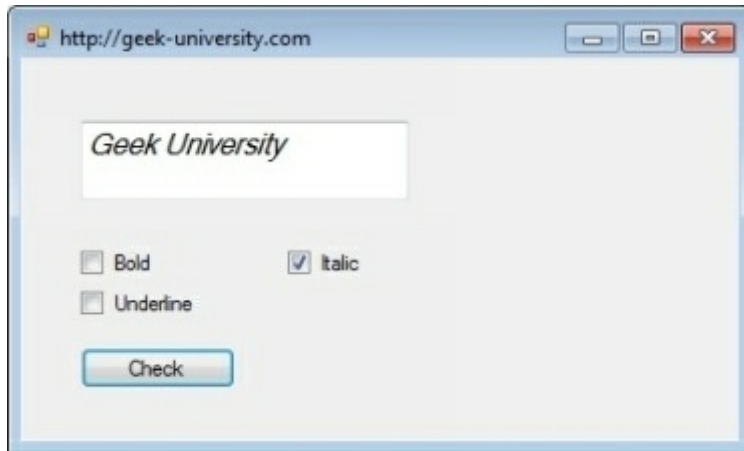
Press F5 on your keyboard to test your application and choose one of the RadioButtons. Click on the **Check** button and see what will happen.



## CheckBox

Like radio buttons, checkboxes can be either selected or deselected. You can determine whether a check box is selected by looking at the **CheckState** property during the runtime . The idea about having CheckBoxes is to offer your users multiple choices, and not just a single choice.

Let's write the code that will allow the user to type some text into a TextBox and format it using three checkboxes that represent bold, italic and underline. This is our form:



Double click the button and add the following code:

```
If CheckBox1.Checked = True Then
    TextBox1.Font = New Font(TextBox1.Font.FontFamily, 12, FontStyle.Bold)
ElseIf CheckBox2.Checked = True Then
    TextBox1.Font = New Font(TextBox1.Font.FontFamily, 12, FontStyle.Italic)
ElseIf CheckBox3.Checked = True Then
    TextBox1.Font = New Font(TextBox1.Font.FontFamily, 12,
    FontStyle.Underline)
End If
```

## NOTE

Don't worry if you don't understand the *if else* statements used in the code above - we will learn more about them in the following chapters.

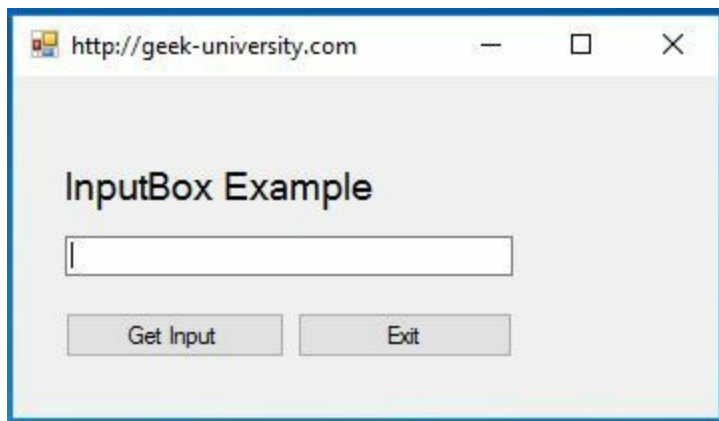
The above program uses the **checked** event to respond to the user selection by checking a particular checkbox. A True value indicates that the check box is selected, whereas a False value indicates that it is not selected.

The radio button and the check box are used for different functions. Use a radio button when you want the user to choose only one option. When you want the user to choose all appropriate options, use a check box.

# Get user input

In this section I will show you how to use the input boxes to get an input from the user. Most of the applications you will write will need some way of interacting with the users. The simplest way to obtain user input is by using the *InputBox* function. This function prompts the user for the input from the keyboard. The InputBox control can only be accessed through the code.

Create a new project for this example. On your new form, add a Button and a Label, laid out as shown below:



The properties set for these controls should be as follows:

## **(First Button):**

Name: btnGetInput

Text: Get Input

## **(TextBox):**

Name: txtInfo

AutoSize: False

## **(Second Button):**



Name: btnExit

Text: Exit

Now, double click the Get Input button, and add the following code:

```
Dim strUserName As String
```

```
strUserName = InputBox("Enter your name:", "Test", "Type your name here.")
```

```
    If strUserName = "" Then
```

```
        txtInfo.Text = ""
```

```
    Else
```

```
        txtInfo.Text = "User Name: " & strUserName
```

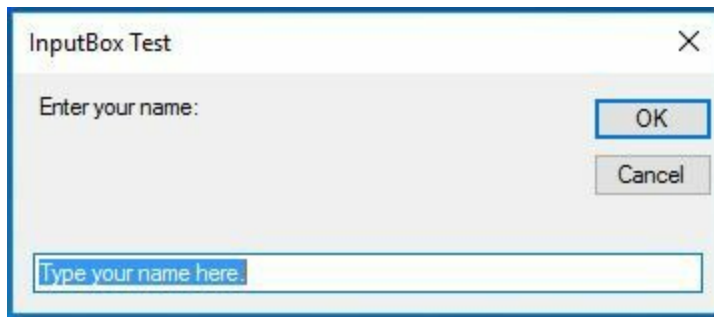
```
    End If
```

The first parameter of the input box is the text displayed in the prompt. The second parameter is the title of the prompt, and the third parameter is the default text displayed in the box.

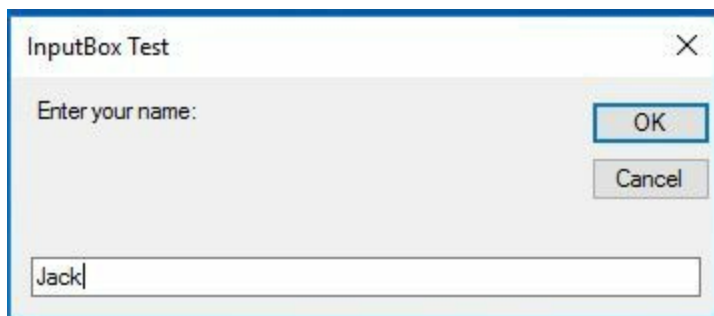
For the **Exit** button, add the following code to close the prompt:

```
Me.close()
```

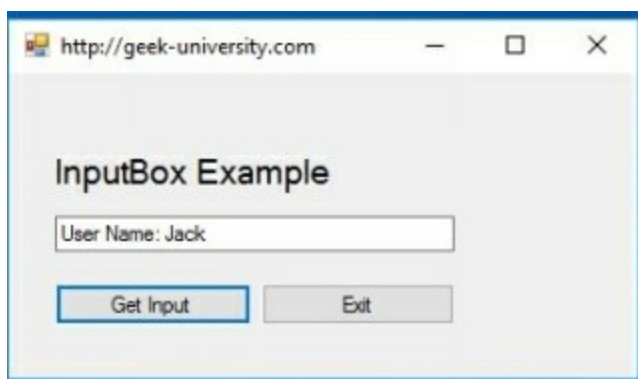
When the above code is executed, it should produce the following output:



Now, enter your name in the textbox (in my example, *Jack* is entered):



After you click the OK button, the *txtInfo* control will display the name you've entered:



# **Chapter 4 - Operators**

## **IN THIS CHAPTER**

**Types of operators**

**Perform math operations using arithmetic operators**

**Logical and assignment operators explained**

Now it's time to talk a little bit about the operators in Visual Basic 2015. An operator is a special symbol that indicates a certain process of manipulation. Operators in programming languages are taken from mathematics.

We have several types of operators:

- Arithmetic operators
- Comparison operators
- Logical operations
- Assignment operators

## Arithmetic operators

The arithmetic operators in Visual Basic 2015 are used to perform math operations, such as addition, subtraction, multiplication, and division. Visual Basic 2015 also offers a number of libraries that enable you to perform more complex math tasks.

Here is a list of the arithmetic operators in Visual Basic 2015:

- **Addition (+)** - adds two values together. Example:  $3 + 2 = 5$ .
- **Subtraction (-)** - subtracts the right hand operand from the left hand operand.  $3 - 2 = 1$ .
- **Multiplication (\*)** - multiplies the right operand by the left operand. Example:  $3 * 4 = 12$ .
- **Division (/)** - divides the left operand by the right operand. Example:  $9 / 3 = 3$ .
- **Modulus (Mod)** - divides the left operand by the right operand and returns the remainder. Example:  $10 \text{ Mod } 3 = 1$
- **Exponent (^)** - calculates the exponential value. Example:  $3 ^ 4 = 81$ .
- **Integer Division (\)** - performs integer division. Example:  $53 \setminus 25 = 2$ .

You are probably familiar with all operators mentioned above, except the **modulus operator (Mod)**. The concept is actually simple - the modulus operator returns the remainder after integer division.

Consider the following example:

### **20 Mod 3**

The integer division of the numbers above will give the result of **6**.

$$6 * 3 = 18.$$

This gives a remainder of **2** ( $18 + 2 = 20$ ).

Another example:

### **5 Mod 2**

Integer division = **2**.

$$2 * 2 = 4.$$

The remainder is **1**. So **5 Mod 2 = 1**.

One more example:

### **13 Mod 5**

Integer division = **2**

$$2 * 5 = 10$$

The remainder is **3**.

## **Comparison operators**

As their name suggests, the comparison operators (also referred to as relational operators) in Visual Basic 2015 are used to compare one value to another. The result of a comparison is a Boolean value, which can either be True or False. The following comparison operators exist in Visual Basic 2015:

- **Equal to (=)** - determines whether two values are equal.
- **Not equal to (<>)** - determines whether two values are not equal.

- **Greater than ( > )** - determines whether the value of the left operand is greater than the value of the right operand.
- **Less than ( < )** - determines whether the value of the left operand is less than the value of the right operand.
- **Greater than or equal to ( >= )** - determines whether the value of the left operand is greater than or equal to the value of the right operand.
- **Less than or equal to ( <= )** - determines whether the value of the left operand is less than or equal to the value of the right operand.

Here are a couple of examples.

### **Example 1**

If firstName = "John" Then

The condition evaluates to True when the *firstName* variable contains the string *John*; otherwise, it evaluates to False.

### **Example 2**

If yourAge > 18 Then

The condition evaluates to True when the value stored in the *yourAge* variable is greater than 18; otherwise, it evaluates to False.

### **Example 3**

If number1 >= 10 Then

The condition evaluates to True when the value stored in the *number1* variable is greater than or equal to 10; otherwise, it evaluates to False.

### **Example 4**

If num1 < num2 Then

The condition evaluates to True when the value stored in the *num1* variable is less than the value stored in the *num2* variable; otherwise, it evaluates to False.

### Example 5

If num1 <= 10 Then

The condition evaluates to True when the value stored in the *num1* variable is less than or equal to 10; otherwise, it evaluates to False.

### Example 6

If strName <> "John" Then

The condition evaluates to True when the *strName* variable does not contain the string *John*; otherwise, it evaluates to False.

## Logical operators

The logical operators in Visual Basic are used to combine the True or False values of variables or expressions, so you can figure out their resultant truth value.

Visual Basic provides these six logical operators:

- **And** - returns **True** only if both operands are true. In any other case, **False** will be returned.
- **AndAlso** - same as the *And* operator, except that it performs a short-circuit evaluation.
- **Or** - returns **True** when one or both of the operands are true.
- **OrElse** - same as the *Or* operator, except it performs a short-circuit evaluation.

- **Xor** - only one of the subconditions can be true for the compound condition to evaluate to True.
- **Not** - negates the truth value of a single operand. In other words, **True** becomes **False** and vice versa.

Here are a couple of examples:

### Example 1

If username = "John" And pass = "1234" Then

    MsgBox("Welcome")

Else

    MsgBox("your username or password was incorrect")

End If

The condition evaluates to True when the *username* variable contains the string *John* and when the *pass* variable contains the string *1234*; otherwise, it evaluates to False.

### Example 2

If num1 > 0 AndAlso num1 <0.20

The compound condition evaluates to True when the value of the *num1* variable is greater than 0 and, at the same time, less than 0.20; otherwise, it evaluates to False.

### Example 3

Dim user As String

    user = "admin"

    If user = "admin" Or user = "Admin" Then

        MsgBox("Welcome admin")



End If

The condition evaluates to True when the *usern* is *admin* or *Admin*; otherwise, it evaluates to False.

#### **Example 4**

If strCode = "2" OrElse decSales > 4999.99 Then

The compound condition evaluates to True when the *strCode* variable contains the string 2 or when the value in the *decSales* variable is greater than 4999.99; otherwise, it evaluates to False.

#### **Example 5**

If username1 = "John" Xor username2 = "John" Then

The compound condition evaluates to True when only one of the variables contains the string *John*; otherwise, it evaluates to False.

#### **Example 6**

If Not (2 = 3) Then

    MessageBox.Show("(2 = 3) is False. So Not False is True")

End If

The *Not* operator returns True when the condition is False. Otherwise, it returns False.

## **Assignment operators**

The assignment operators in Visual Basic 2015 are used to store data into a variable. We've already used the most common assignment operator (=), but

there are many more of them:

- `=` - assigns the value found in the right operands to the left operand.  
Example: `x = 5`.
- `+=` - adds the value found in the right operand to the value found in the left operand.  
Example: `x = 5`, `x += 3` results in `x = 8`.
- `-=` - subtracts the value of the right operand from the value found in the left operand.  
Example: `x = 5`, `x -= 3` results in `x = 2`.
- `*=` - multiplies the value of the right operand by the value of the left operand. Example: `x = 5`, `x *= 3` results in `x = 15`.
- `/=` - divides the value of the left operand by the value of the right operand. Example: `x = 5`, `x /= 3` results in `x = 1.667`.
- `&=` - Concatenates a String expression to a String variable or property and assigns the result to the variable or property  
Example:

```
Dim var1 As String = "Hello "  
Dim var2 As String = "World!"  
var1 &= var2  
' The value of var1 is now "Hello World!"
```

- `^=` - determines the exponential value found in the left operand when raised to the power of the value in the right operand.  
Example: `x = 5`, `x ^= 3` results in `x = 125`.

# **Chapter 5 - Flow control**

## **IN THIS CHAPTER**

**Explaining if...else statements**

**Select...case statement**

**Repeating actions using loops**

# If...else statements

The *if...else* statements in Visual Basic 2015 are used when you need to choose between one of two alternatives. The *else* clause in the code will be executed if the condition for the *if* statement isn't met, which allows you to perform an alternative task.

The syntax of the *if...else* statement is:

' Multiple-line syntax:

```
If condition [ Then ]
```

```
    [ statements ]
```

```
[ Else
```

```
    [ elstatements ] ]
```

```
End If
```

' Single-line syntax:

```
If condition Then [ statements ] [ Else [ elstatements ] ]
```

Consider the following example:

```
Dim age As Integer
```

```
age = InputBox("How old are you?:", "geek-university.com", "Type your age here.")
```

```
    If age >= 21 Then
```

```
        MessageBox.Show("you are old enough.")
```

```
    Else
```

```
        MessageBox.Show("you are NOT old enough.")
```

```
    End If
```

In the example above, the user is prompted to enter his name and age. The *if* statement will then evaluate whether the user is 21 or older and print the corresponding message. If the user enters a number less than 21, the *else* statement will be executed and the corresponding message will be printed. Note that the *else* statement will be executed only if the *if* statement evaluates to **false**. If the user enters a number greater or equal to 21, the code under the *else* statement will not be executed.

Here is another example:

```
Dim x As Integer
x = InputBox("Enter a number:", "geek-university.com", "Type your number here.")
If x Mod 2 = 0 Then
    MessageBox.Show("The number you have entered is an even number")
Else
    MessageBox.Show("The number you have entered is an odd number.")
End If
```

The output:

Enter a number: 5

The number you have entered is an odd number.

Enter a number: 67

The number you have entered is an odd number.

Enter a number: 100

The number you have entered is an even number.

## Select Case Statement

An alternative to the *if..else* statement is the *Select Case* statement, which compares the same expression to different values. It is very often simpler and clearer to code the selection structure using the *Select Case* statements rather than using the several *if...then...else* statements.

Here's the syntax of the Select Case statement:

```
Select Case[ Case ] testexpression
```

```
    [ Case expressionlist
```

```
        [ statements ] ]
```

```
    [ Case Else
```

```
        [ elstatements ] ]
```

```
End Select
```

A breakdown of the syntax:

- **Select Case** - the keyword that indicates that the **Select Case** statement begins.
- **testexpression** - the expression that must evaluate to one of the elementary data types (Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Object, SByte, Short, Single, String, UInteger, ULong, and UShort).
- **expressionlist** - list of expression clauses representing match values for testexpression. Multiple expression clauses are separated by commas.
- **statements** - optional. One or more statements following *Case* that run if testexpression matches any clause in *expressionlist*.
- **elstatements** - optional. One or more statements following *Case Else* that run if *testexpression* does not match any clause in the expressionlist of any of the *Case* statements..
- **End Select** - terminates the definition of the *Select...Case* construction.

So, let us start our new example by adding some objects in our form. Create a form like this:



The properties set for these controls are as follows:

**(First Button):**

Name: btnOk

Text: Ok

**(TextBox):**

Name:txtInfo

AutoSize: False

Now, in the click button event, add the following code:

```
Dim game As String
```

```
    game = txtInfo.Text
```

```
    Dim answer As String
```

```
    answer = game & " is your favourite game!"
```

```
    Select Case game
```

```
        Case "GTA 4"
```

```
            MessageBox.Show(answer)
```

```
        Case "Battlefield 3"
```

```
            MessageBox.Show(answer)
```

```
        Case "GRID"
```

```
        MessageBox.Show(answer)
    Case Else
        MessageBox.Show("Looks like your game is not on my list")
End Select
```

The code above first sets up a string variable. Next, we use a *Select Case* statement. The application will continue to check all the words after *Case* to see if one of them contains the text that is in the variable *game*. If it finds one of them it will execute the code below the *Case* keyword; if it doesn't find any matches, it will display the message *Looks like your game is not on my list*.

Press F5 on your keyboard to run application. Test whether the code works by entering different game names inside the textbox.

## Specify a range of values in a *Case* clause

In addition to specifying one or more values in a *Case* clause, you can also specify a range of values, such as the values 1 through 4 or values greater than 10. You can do this using either the keyword *To* or the keyword *Is*. You use the *To* keyword when you know both the upper and lower values in the range. The *Is* keyword is used when you know only one end of the range (either the upper or lower end).

Here is an example. Create a VB Form Application and name it *Select Case*, then write the following code:

```
Dim age As Integer = 22

Select Case age
    Case 15 To 21
        Console.WriteLine("You are still young")
    Case 21 To 45
```



```
        Console.WriteLine("You are not a kid anymore")
    Case Else
        Console.WriteLine("Didn't understand that")
    End Select
```

Now, click **Start Debugging** on the **Standard** toolbar. The program will run and the *Case 21 To 45* statement will be executed.

## For Loop

Looping statements in Visual Basic are used to repeat an action over and over again. One of the most common looping statements is the *for* loop.

The syntax of *for* loop is:

```
For counter [ As datatype ] = start To end [ Step step ]
    [ statements ]
Next [ counter ]
```

Here is a breakdown of the syntax:

- **For** - the keyword that indicates that the for statement begins.
- **Datatype** - Optional. The data type of the counter.
- **start** - Required. Numeric expression. The initial value of counter.
- **end** - Required. Numeric expression. The final value of counter.
- **Next** - Required. Terminates the definition of the **For** loop.

Take a look at the following example:

```
Dim number As Integer
For number = 1 To 10
    MessageBox.Show(number.ToString)
```

Next number

The code above will display the numbers 1 through 10 in message boxes. Here is the code breakdown:

**For** - starts the *for* statement.

**number = 1 To 10** - the name of the variable that will hold the current value. During the first iteration, the first item in the sequence will be assigned to the variable. During the second iteration, the second item in the sequence will be assigned to the variable, etc. So in our example, the first value of number will be 1, the second 2, and so on until the last value of 10.

**MessageBox.Show(number.ToString)** - the statement that will be executed in each iteration.

**Next number** - tells Visual Basic to grab the next number in the sequence.

## Do Loops

The *Do...Loop* statements are used when you want to repeat a set of instructions an indefinite number of times, until a condition is satisfied.

### NOTE

If you want to repeat the statements a set number of times, the *For...Next* statement is a better choice.

With a *Do Loop* you can use either *While* or *Until* keyword to specify condition, but not both at the same time. You can test condition only one time, at either the start or the end of the loop.

The basic syntax is as follows:

Do {While} condition

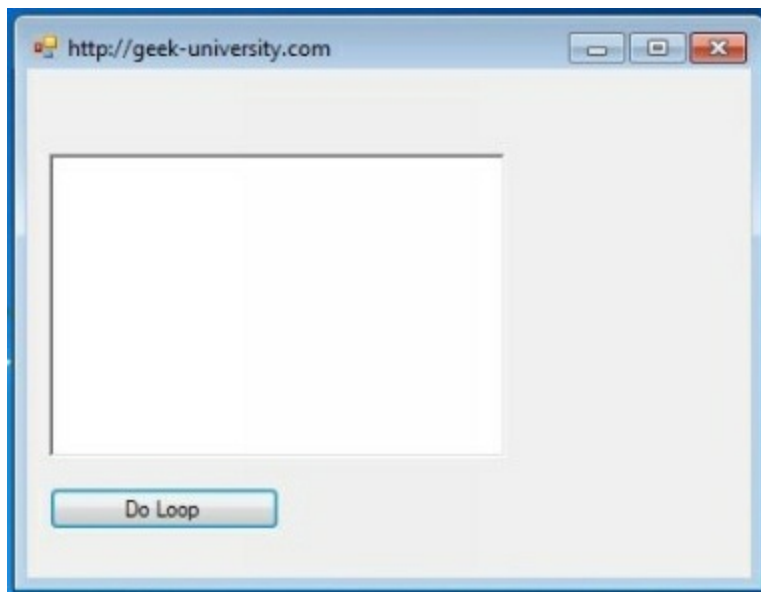
statements

Loop

Let's explain the syntax:

- **Do** - Required. Starts the definition of the *Do* loop.
- **While** - Required unless *Until* is used. Repeats the loop until condition is False.
- **Condition** - Optional Boolean expression. If the condition is Nothing, Visual Basic treats it as False.
- **Statements** - Optional. One or more statements that are repeated while, or until, condition is True.
- **Loop** - Required. Terminates the definition of the *Do* loop.

Here is an example of a *do* loop. Create a New project for this example. Add a Button and a Rich Text Box to the form, laid out as shown below:



The properties set for these controls are as follows:

**(Button)**

Name: btnDo

Text: Do Loop

**(Rich TextBox)**

Name: rchTextBox

Text: leave blank

Now, double click on the button and add the following code:

Dim line As String

FileOpen(1, "C:\test.txt", OpenMode.Input)

Do While Not EOF(1)

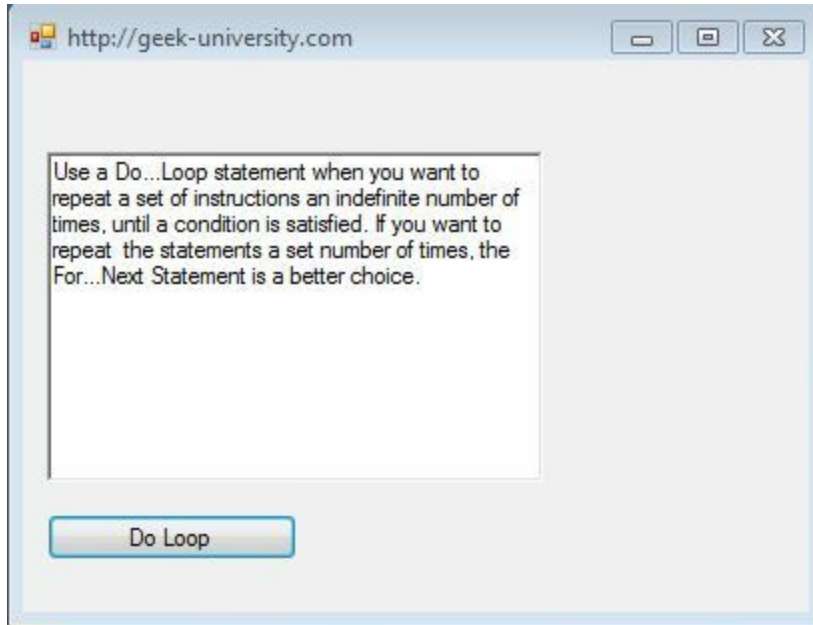
line = line & LineInput(1)

rchTextBox.Text = line

Loop

Enter some text in a text file and save it to **C:\test.txt**.

Press F5 key on your keyboard and launch your program. The code above will read the text file and show it on the screen until the EOF (End Of File) is reached. When the form loads click on the *Do Loop* button, and you should get the following (your text will be different):



# **Chapter 6 - Working with data structures**

## **IN THIS CHAPTER**

**Learning about arrays**

**Using constants in your code**

**Working with structures**

So far we've worked with simple data types such as integer or string variables. Although these data types are useful in their own right, more complex programs require data structures - that is, groups of data elements that are organized in a single unit. So let's learn about the various data structures available in Visual Basic 2015.

## Arrays

So far, we have learned that a variable is a storage that can hold only one piece of information. For example:

```
Dim Salary As Integer
```

```
Salary = 15000
```

Or this:

```
Dim name As String
```

```
name = "Jack"
```

But what if you wanted to store the salaries of 5 employees? You could either declare 5 variables - *Salary1*, *Salary2*, and so on up to *Salary5* - or declare an array with 5 elements.

An **array** is similar to a variable in that it has a name and can hold values. In fact, an array is actually a variable that can hold more than one piece of information at time.

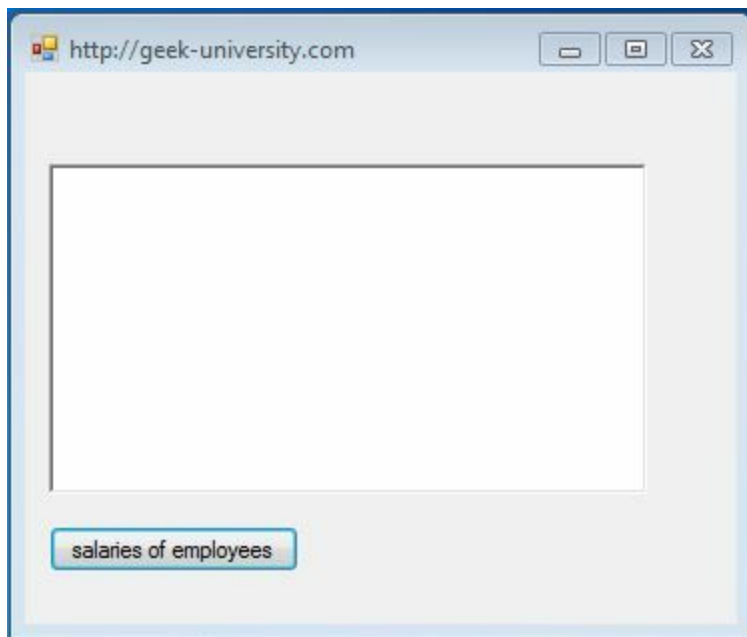
Arrays must be declared with the *Dim* (or *Public*) statement, followed by the name of the array and the index of the last element in the array in parentheses, as in the following example:

## Dim Salary(5) As Integer

In the example above, *Salary* is the name of an array that holds 5 values (the salaries of the 5 employees), with indices ranging from 0 to 4. Note that the numbering starts from 0 - *Salary(0)* is the first person's salary, *Salary(1)* the second person's salary, and so on, until *Salary(4)*, the last person's salary.

So, let us start our new example by adding one button and one textbox to our form.

The form will look like this when you are finished:



The properties set for these controls are as follows:

### **(Button)**

Name: btnSalaries

Text: salaries of employees

### **(Rich TextBox)**

Name: rchSalaries

Text: leave blank

Put the following code for your button:

```
Dim salesalaries(5) As Integer
```

```
    salesalaries(0) = 1000
```

```
    salesalaries(1) = 1500
```

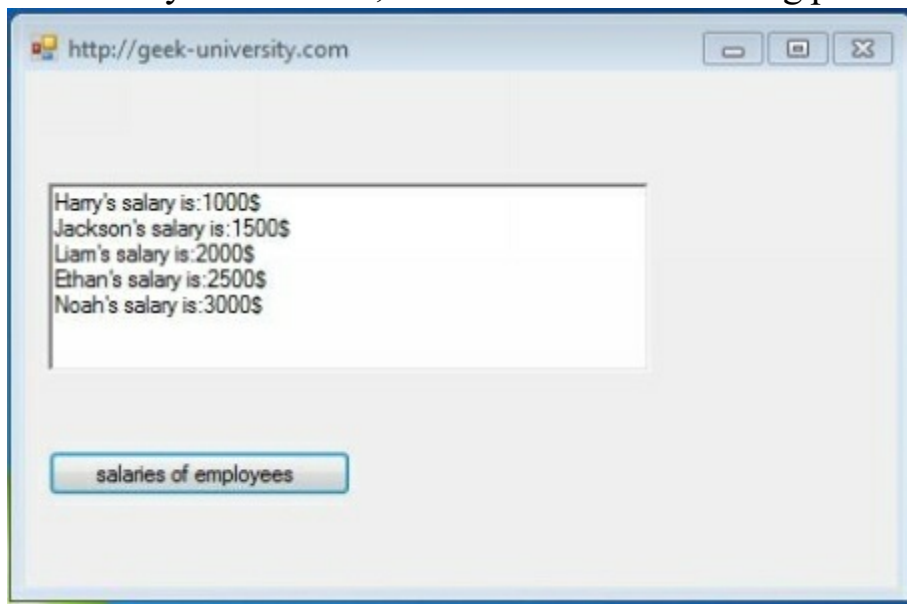
```
    salesalaries(2) = 2000
```

```
    salesalaries(3) = 2500
```

```
    salesalaries(4) = 3000
```

```
rchSalaries.Text = "Harry's salary is:" & salesalaries(0) & "$" & vbNewLine _  
    & "Jackson's salary is:" & salesalaries(1) & "$" & vbNewLine _  
    & "Liam's salary is:" & salesalaries(2) & "$" & vbNewLine _  
    & "Ethan's salary is:" & salesalaries(3) & "$" & vbNewLine _  
    & "Noah's salary is:" & salesalaries(4) & "$" & vbNewLine
```

Press F5 and run your program to test it out. The program fills the rich textbox with salary information, as shown in the following picture:



## Visual Basic 2015 Constants



A **constant** is variable whose value cannot be changed. These variables can appear many times in your code. They are declared using the *Const* statement.

Syntax:

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ]
```

Const constantlist

The *Const* statement specifies that the variable's value is constant and tells the compiler to prevent the programmer from modifying it. Here is the explanation of the syntax:

**attributelist** - optional. List of attributes that apply to all the constants declared in this statement.

**accessmodifier** - optional. Use this attribute to specify what code can access these constants. The possible values are Public, Protected, Friend, Protected Friend or Private.

**Shadows** - optional. Use this to redeclare and hide a programming element in a base class.

**constantlist** - required. List of constants being declared in this statement.

For example, if your program does math calculations, the value of **pi** (3.14159. . .) might appear many times. Instead of typing the value 3.14159 each time you need **pi**, you can define a constant, name it *pi*, and use the name of the constant in your code. Const values resolve faster than regular variables.

Start a new project for this example. Place two buttons, three labels and one textbox on the form.

Change the Properties of the first Textbox to the following:

**(First Button):**

Name: btnCalc

Text: Calculate Area

**(TextBox):**

Name: txtRadius

Text: leave the textbox blank

**(Second Button):**

Name: btnExit

Text: Exit

**(Label1):**

Name: lblCirclesRadius

Text: Circle's radius:

**(Label2):**

Name: lblCirclesArea

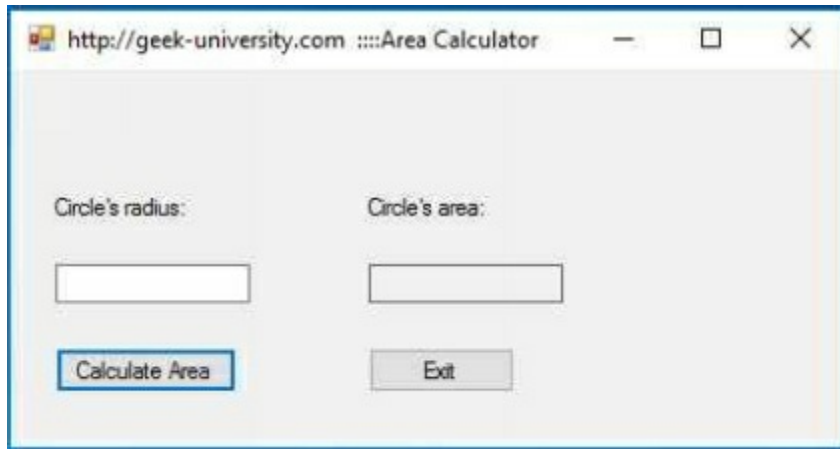
Text: Circle's area:

**(Label3):**

Name: lblArea

Text: Leave empty

Your form will look similar to this:



Now, double click your new **Calculate Area** button, and paste the following code:

```
Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
```

```
    Const dblPI As Double = 3.141593
```

```
    Dim dblRadius As Double
```

```
    Dim dblArea As Double
```

```
    ' calculate and display area
```

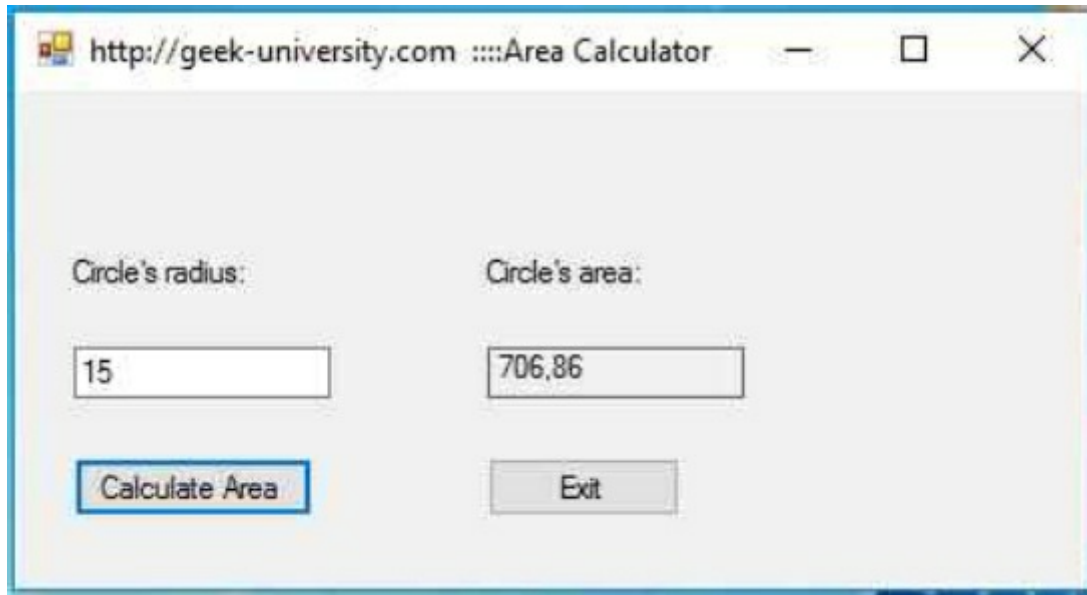
```
    Double.TryParse(txtRadius.Text, dblRadius)
```

```
    dblArea = dblPI * dblRadius * dblRadius
```

```
    lblArea.Text = Format(dblArea, "standard")
```

```
End Sub
```

Once you have typed your code, press F5 to start your program and test it out. For example, type 15 in the Circle's radius box and then click the **Calculate Area** button.

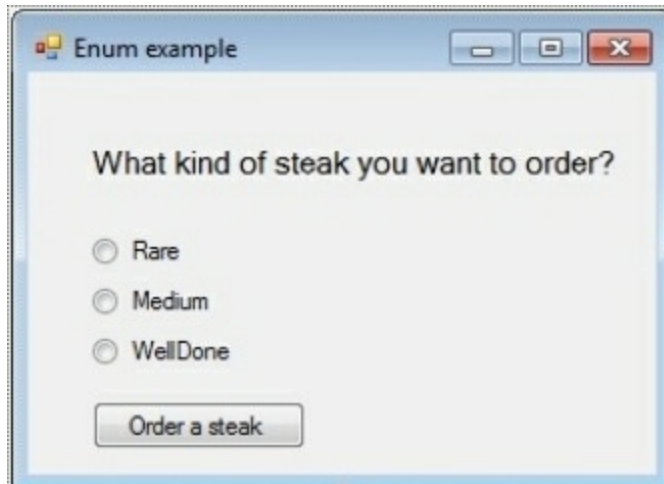


The number 706,86 appears in the Circle's area box, as shown in the image above.

## Enumerations

When you have a lot of constants in your program that are logically related to each other, you can define them together in an enumerator list. **Enumerations** are defined using the *Enum* keyword, followed by the enumeration name and type.

So let's start our new example by adding some objects to our form. Create a form like this:



The properties set for these controls should be as follows:

**(First Button):**

Name: btnOrder

Text: Order a steak

**(Label1):**

Text: What kind of steak you want to order?

**(RadioButton1):**

Text: Rare

**(RadioButton2):**

Text: Medium

**(RadioButton3):**

Text: WellDone

Now, double click on your form and paste the following code:

```
Public Class Form1
    Enum Steak
        Rare
        Medium
        WellDone
    End Enum
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load

    End Sub

    Private Sub btnOrder_Click(sender As Object, e As EventArgs) Handles btnOrder.Click
        Dim value As Steak
        If RadioButton1.Checked = True Then
            value = Steak.Rare
            MessageBox.Show("You ordered a " & value.ToString & " steak")
        ElseIf RadioButton2.Checked = True Then
            value = Steak.Medium
            MessageBox.Show("You ordered a " & value.ToString & " steak")
        ElseIf RadioButton3.Checked = True Then
            value = Steak.WellDone
            MessageBox.Show("You ordered a " & value.ToString & " steak")
        Else
            MessageBox.Show("Please decide which steak you want")
        End If
    End Sub
End Class
```

End If

End Sub

End Class

The code above demonstrates the declaration and use of the *Enum* variable *Steak*.

Run your application. If you choose the third radio button, you should get the following:



## Structures

Structures are very useful when you want a single variable to hold several related pieces of information.

For example, you might want to keep information about the computer name, OS version and the amount of RAM together. You could use several variables for this information, or you could define a structure and use it for a single system information variable. The advantage of the structure becomes apparent when

you have many system information and therefore many instances of the variable.

Create a form similar to this one:

A screenshot of a web browser window with the address bar showing 'http://geek-university.com'. The page content is a form titled 'System Information'. It contains three input fields: 'Name:', 'RAM:', and 'OS:'. Below these fields is a button labeled 'Get System Information'.

Next, double click on the form and type the following code:

```
Public Class Form1
```

```
    Private Structure systemInformation
```

```
        Public name As String
```

```
        Public ram As Long
```

```
        Public os As String
```

```
    End Structure
```

```
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
End Sub
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
    Dim mypcinfo As systemInformation
```

```
    mypcinfo.name = Environment.MachineName
```

```
    TextBox1.Text = mypcinfo.name
```

```
    mypcinfo.ram = My.Computer.Info.TotalPhysicalMemory
```

```
    TextBox2.Text = mypcinfo.ram
```



```
mypcinfo.os = My.Computer.Info.OSFullName
```

```
TextBox3.Text = mypcinfo.os
```

```
End Sub
```

```
End Class
```

When you are done, run the application. Click on the button and you should get something like this:



# **Chapter 7 - String Manipulation**

## **IN THIS CHAPTER**

**Working with strings**

**Converting strings**

**Using various string methods**

The important thing about strings in Visual Basic is that they are case sensitive - the string *Word* is not the same as either the string *WORD* or the string *word*. Because of this, a problem might occur when comparing strings that are entered by the user who may insert the string using any combination of uppercase and lowercase letters.

The *String* class of the .NET Framework provides many built-in methods for the comparison and manipulation of strings. In this section I will show you several different ways to manipulate strings. Some of the methods are a part of the Visual Basic language, and others are inherent in the *String* class.

## ToUpper method

If you want to convert a string to uppercase, use the *ToUpper* method of the *String* class.

The following example illustrates how this method is used:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim word As String
    word = "hello world"
    MessageBox.Show(word.ToUpper)
End Sub
```

The code above returns the string stored in the *word* variable, converted to uppercase.

Create a form with a single button, add the code above, and press F5 key on your keyboard and launch your program. The form you get should look like this:



## ToLower method

If you want to convert a string to lowercase, use the *ToLower* method of the *String* class.

The following example illustrates how this method is used:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim word As String
    word = "HELLO WORLD"
    MessageBox.Show(word.ToLower)
End Sub
```

The code above returns the string stored in the *word* variable, converted to lowercase.

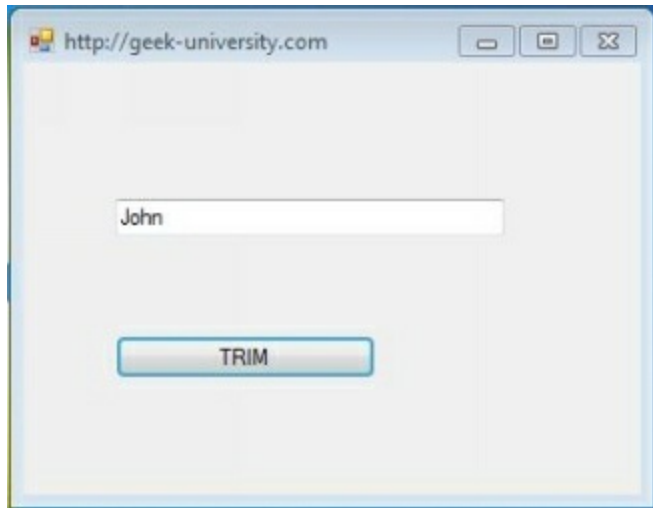
## The Trim Method

The *Trim* method in the *System.String* class is used to remove (trim) any number of spaces or other characters from a specified position in the string.

Consider the following example:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim FirstName As String
    FirstName = "  John  "
    TextBox1.Text = FirstName.Trim
End Sub
```

Run your program to test it out. You should get something like this:



As you can see in the example above, the Trim function trimmed the empty spaces on both sides of the word *John*.

## The InStr function

The *InStr* function looks for a word that is embedded within the original phrase and returns the starting position of the embedded phrase. For example, if you want to verify that strings are in valid email format, you could use the *InStr()* function to check for the presence of the character @.

So, let's start our new project by adding one button and one textbox to our form.

The properties set for these controls are as follows:

**(Button)**

Name: btnCheck

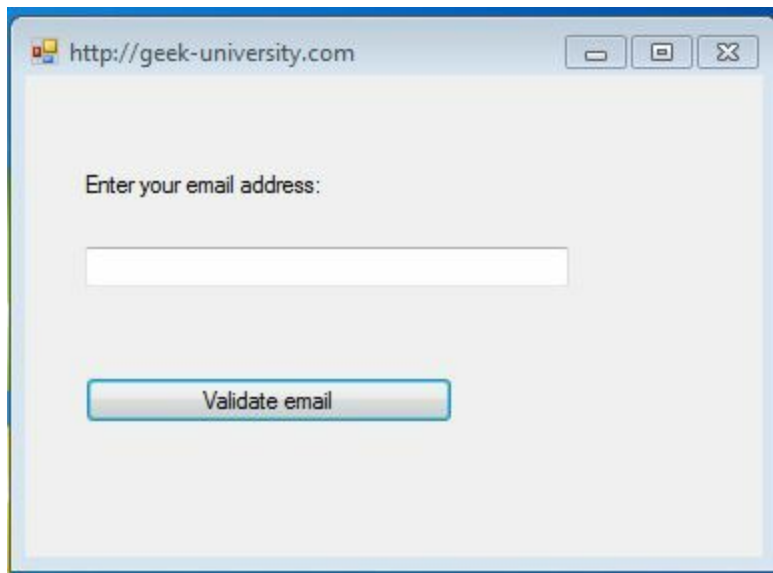
Text: Validate Email

**(TextBox)**

Name: txtMail

Text: leave blank

Your form should look like this:



Now, double click the button. In the button click event, paste this code:

```
If InStr(txtMail.Text, "@") Then
```

```
' Return true if strIn is in valid e-mail format.
```

```
Else
```

```
    MessageBox.Show("Please insert a valid email address!")
```

```
End If
```

The *txtMail* object is the string we want to check. In the code above, if the e-mail address is not well-formed inside the *txtMail* object, an error message will appear, instructing the user to enter a valid email address.

## The Val function

The *Val function* converts a string to a number and then returns the number. The number is stored in the computer's memory only while the function is processing.

The Val function must be able to interpret the string as a numeric value, so the string cannot include a letter, a comma, or a special character (such as the dollar sign or the percent sign); however, it can include a period or a space. When the Val function discovers an invalid character in its string argument, it stops converting the string to a number.

Here is a Val function call example :

```
Val("16")
```

```
Val("-5")
```

```
Val("1.66")
```

```
Val("80a5")
```

```
Val("8+5")
```

```
Val(" + 1 2 3 4 5 ")
```

```
Val(" hello world")
```

The output:

16

-5

1.66

80

8

12345

0

## The Contains Method

The **Contains** method determines whether a string contains a specific sequence of characters. It returns a Boolean that indicates whether the argument string was located in the instance string.

Consider the following example:

```
Dim firstName As String
    firstName = txtFirstName.Text
    If firstName.Contains("John") Then
        MessageBox.Show("Welcome")
    Else
        MessageBox.Show("Who are you?")
    End If
```

The code above will assign True to the firstName variable if you enter the string John in the textbox and the *Welcome* message will be displayed; otherwise, the message *Who are you?* will be shown to the user.



# The Insert Method

The *Insert* method allows you to insert characters anywhere in a string. The *Insert* method does not affect to the original string.

Consider the following example:

```
Dim firstName As String = "John"  
    Dim lastName As String = "Doe"  
    Dim fullName As String = firstName.Insert(4, lastName)  
    MsgBox(fullName)
```

The Insert function will insert text contained in the *lastname* variable. When the program is executed, you should get the message box containing the text *John Doe*.

# **Chapter 8 - Working with files**

## **IN THIS CHAPTER**

**Reading and writing files**

**Copying and moving files**

**Deleting files**

This section of the book teaches you the basics of file manipulation. Working with files in Visual Basic 2015 is quite simple: you associate a filehandle with an external file (for example a text file) and then use a variety of operators and functions within Visual Basic to read and update the data stored within the data stream associated with the filehandle.

The classes in the *System.IO* namespace are used to manipulate drives, files, and directories. This namespace contains the *File* and *Directory* classes, which provide the functionality that you need to manipulate files and directories. The classes are shared, which means that you can call their methods without having to create class instances.

The following table shows some commonly used classes in the *System.IO* namespace, so you can use it as a reference:

| <b>I/O Class</b> | <b>Description</b>   |
|------------------|--|
| BinaryReader     | Reads primitive data types as binary values in a specific encoding   |
| BinaryWriter     | Writes primitive types in binary to a stream and supports writing strings in a specific encoding.                                      |
| BufferedStream   | Adds a buffering layer to read and write operations on another stream. This class cannot be inherited.                                 |
| Directory        | Exposes static methods for creating, moving, and enumerating through directories and subdirectories. This class cannot be inherited.   |
| DirectoryInfo    | Exposes instance methods for creating, moving, and enumerating through directories and subdirectories. This class cannot be inherited. |
| DriveInfo        | Provides access to information on a drive.   |
| File             | Provides static methods for the creation,  |

|              |   |
|--------------|---|
|              | copying, deletion, moving, and opening of a single file, and aids in the creation of FileStream objects.  |
| FileInfo     | Provides properties and instance methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of FileStream objects. This class cannot be inherited. |
| FileStream   | Provides a Stream for a file, supporting both synchronous and asynchronous read and write operations.   |
| MemoryStream | Creates a stream whose backing store is memory.   |
| Path         | Performs operations on String instances that contain file or directory path information. These operations are performed in a cross-platform manner.   |
| StreamReader | Implements a TextReader that reads characters from a byte stream in a particular encoding.  |
| StreamWriter | Implements a TextWriter for writing characters to a stream in a particular encoding.  |

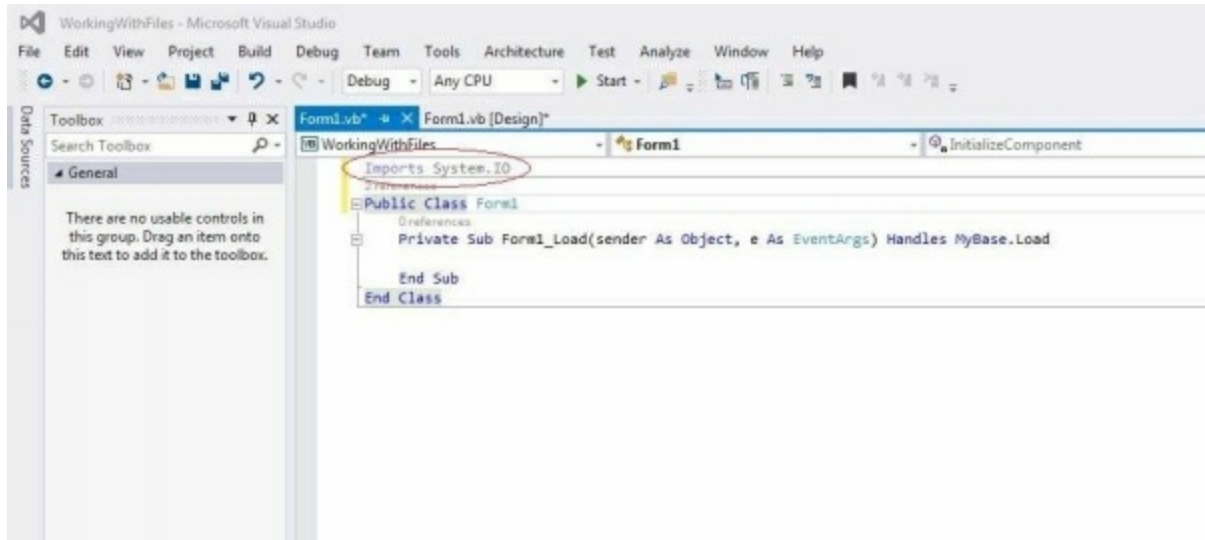
## How to Open a File in Visual Basic 2015

Whenever you need to open a file, you need to force the user to type the full path and the name of the file. You can use the *OpenFileDialog* class or a string variable with the full path instead of the file dialog.

In our example we are using a *StreamReader* class to read data from a file. First, we need to include the following statement in the program code:

## Imports System.IO

This statement has to be placed on top of the program, above the *Public Class Form1*:



The word *Imports* indicates that we are importing the namespace *System.IO* inside the program. Now we can declare a variable of the *StreamReader* data type using the following statement:

```
Dim objReader As New System.IO.StreamReader(file_name_to_open)
```

In the line above, *System* represents the main object. *IO* is an object within *System*. And *StreamReader* is an object within *IO*.

To read a file, *StreamReader* needs its name. This parameter goes inside a pair of brackets:

```
System.IO.StreamReader(file_name_to_open)
```

Finally, we need to use the *ReadToEnd* method to read the entire text of a text file. The syntax is:

```
TextBox1.Text = objReader.ReadToEnd()
```

Let us start our new example by adding one button and one textbox onto our form.

The properties set for these controls are as follows:

**(Button)**

Name: btnRead

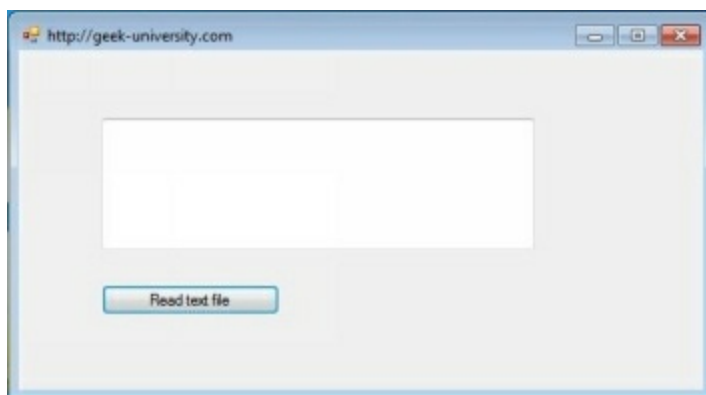
Text: Read text file

**(TextBox)**

Name: txtRead

Text: leave blank

The form should look like this:



DoubleClick the button and type the following code:

```
Dim fileToOpen As String = "C:\test.txt"
    Dim objReader As New System.IO.StreamReader(fileToOpen)
    txtRead.Text = objReader.ReadToEnd
    objReader.Close()
```

Now, run your program, click on the button, and you should see the content of the *test.txt* file inside our TextBox (of course, if the file doesn't already exist, you need to create it and write some text to it):



## Writing to a Text File

Writing to a text file is very similar to reading a text file. When writing to a text file, instead of using *StreamReader*, we use *StreamWriter* to write a stream of strings to a file.

Here is an example. Create a new project and add one button and one textbox object. Change the default properties set for these controls to:

### (Button)

Name: btnWrite

Text: Write to text file

### (TextBox)

Name: txtWrite

Text: leave blank

Add the following code to your write button:

```
Dim fileToWrite As String = "C:\test.txt"
```

```
    If System.IO.File.Exists(fileToWrite) = True Then
```

```
        Dim objWriter As New System.IO.StreamWriter(fileToWrite)
```

```
        objWriter.Write(txtWrite.Text)
```

```
objWriter.Close()  
MessageBox.Show("Text written to file")  
Else  
    MessageBox.Show("File Does Not Exist")  
End If
```

When you click the write button, the text from the *TextBox* object should be saved in a file.

## Copy a File in VB .NET

In order to copy file in Visual Basic 2015, we need to use the *File* object. *File* is an object within *IO* and it has its own properties and methods that you can use. One of these methods is the *Copy* method.

Consider the following example:

```
Dim file1 As String  
    Dim file2 As String  
  
file1 = "C:\test.txt"  
file2 = "C:\test2.txt"  
  
If System.IO.File.Exists(file1) = True Then  
  
    System.IO.File.Copy(file1, file2)  
    MessageBox.Show("File has been Copied")  
  
End If
```

Inside the *if* Statement, we have this piece of code:



```
System.IO.File.Copy(file1, file2)
```

In the code above you can see that I've used the *Copy* method of *System.IO.File* object. In between the round brackets, we first need to type the name of the file you want to copy, and then the name of the new file and its new path.

## Move a File with VB .NET

In Visual Basic 2015, you can move a file in a similar way as copying one - you just need to specify the source file and the new destination for it. The *Move* method of *System.IO.File* object is used to move a file.

Consider the following example:

```
Dim file1 As String
Dim location As String
file1 = "C:\test.txt"
location = "C:\newlocation\test.txt"
If System.IO.File.Exists(file1) = True Then
    System.IO.File.Move(file1, location)
    MessageBox.Show("File Moved")
End If
```

Here is the explanation of the code above:

- *Dim file1 As string* - store the path and name of our text file inside the string variable.
- *Dim location As String* - the new location of our text file.
- *System.IO.File.Move(file1, location)* - one of the Methods available to our new File object is the *Move* method. This method will move our file

to it's new location.

## NOTE

The difference between copying and moving a file is that the original file is retained when the file is copied.

## Delete a file in Visual Basic 2015

To delete a file from your computer using Visual Basic 2015, you need to use the *Delete* method of *System.IO.File* object.

Consider the following example:

```
Dim FileToDelete As String
```

```
FileToDelete = "C:\test.txt"
```

```
If System.IO.File.Exists(FileToDelete) = True Then
```

```
System.IO.File.Delete(FileToDelete)
```

```
MessageBox.Show("File Deleted")
```

```
End If
```

The explanation of the code above:

- *Dim FileToDelete As String* - store the path and name of our file.
- *System.IO.File.Delete(FileToDelete)* - one of the Methods available to our File object is Delete. This method will delete the file.



# **Chapter 9 - Advanced forms**

## **IN THIS CHAPTER**

**Adding menus and sub menus to an application**

**Anchoring and docking**

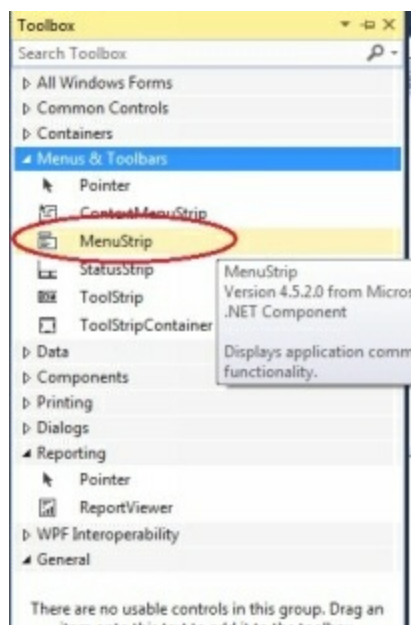
**Creating multiple forms**

**Modal forms**

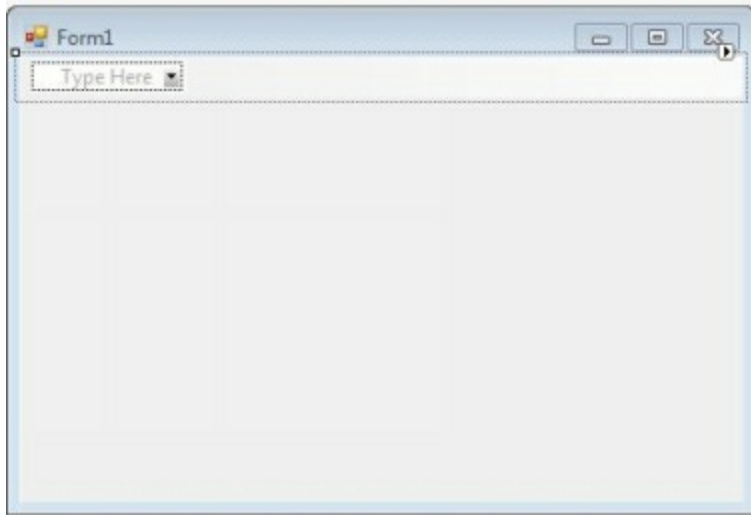
# Adding Menus and Sub Menus in an Application

Menus are the most common and most important elements of the Windows user interface. They are the most popular means of organizing a large number of options.

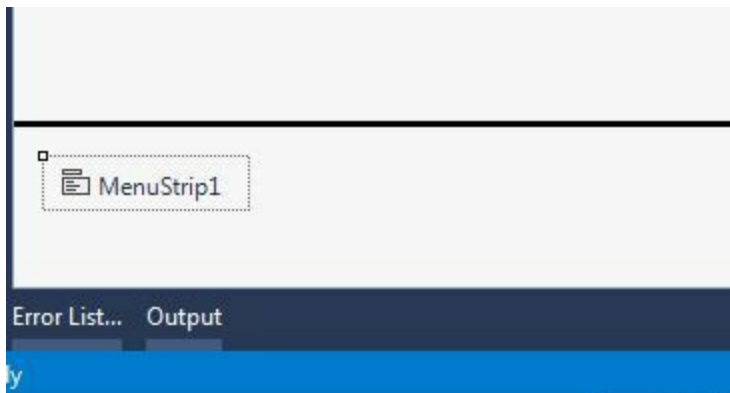
In this section we will go through the process of creating a menu. First, start a new project and create a new Windows Forms application project. Name it anything you like. Now, when your new form appears, you can create a menu bar easily. To do this, create a main menu object on your form using the *MainStrip* control:



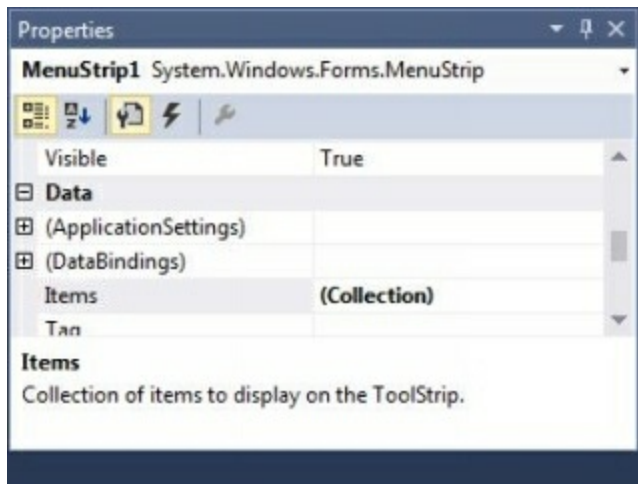
Double click the *MenuStrip* object to add one to your Form. Now, at the top of your form, you should see something like this:



At the bottom of your Visual Studio screen you will see this:



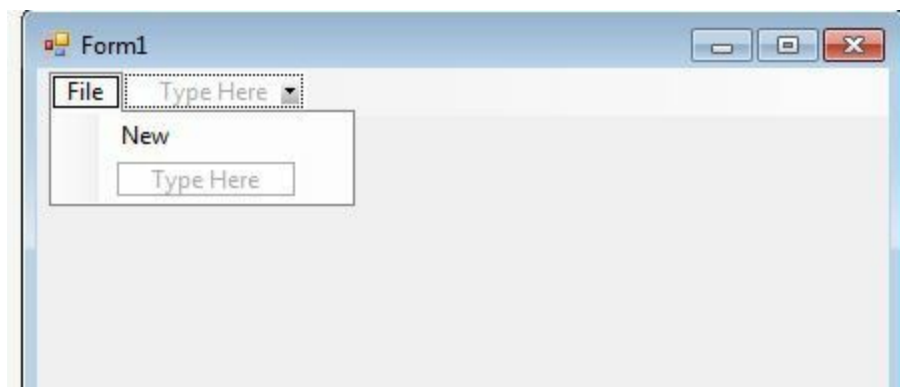
This is the *MenuStrip* toolbox. The default name for this object is *MenuStrip1*. If you click on it, you will see its Properties window on the right side of Visual Studio screen:



Adding items to your menus is very simple. Click inside of the area at the top where it says *Type Here* and enter the word *File*:



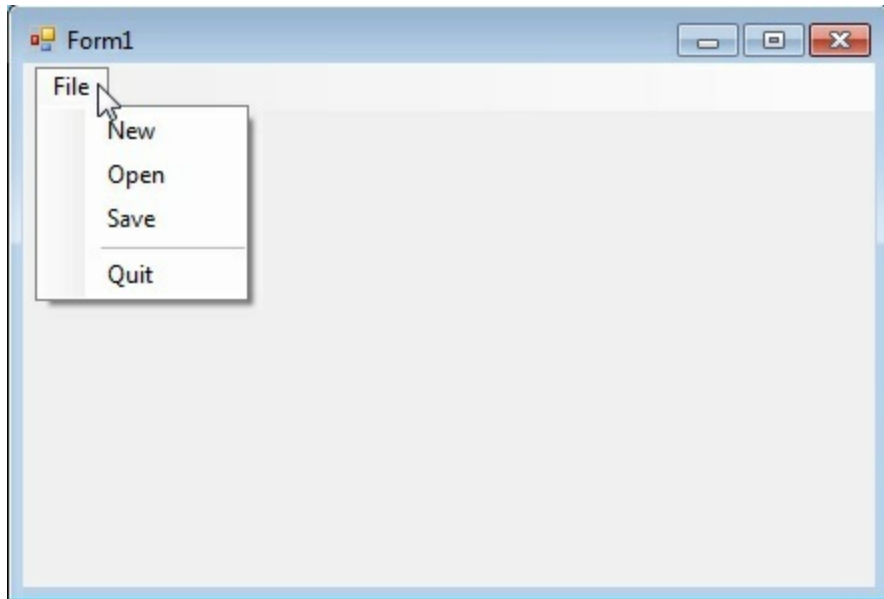
To add items to your *File* menu, click inside the *Type Here* box below the *File* field and type the word *New*. Your menu should look like this:



Create three more menu items called *Open*, *Save* and *Quit* in the same way you've created the *File* menu item. If you need a separator bar, right click on

your menu and select *Insert > Separator*. Also change the *Name* property of the Menu to *mnuQuit*.

Press F5 to run the application. You should now have a menu like the one below:



Click on the *Quit* item. Nothing will happen. This is because we need to add some code so that our *Quit* menu item will do something. Stop debugging by pressing Shift + F7 on your keyboard. Double click on the *Quit* item and add the following in the click event:

```
Me.Close()
```

The code above is simple - the word *Me* refers to the Form and the word *Close* is method. To test your *Quit* item, press F5 key on your keyboard and see what happens.

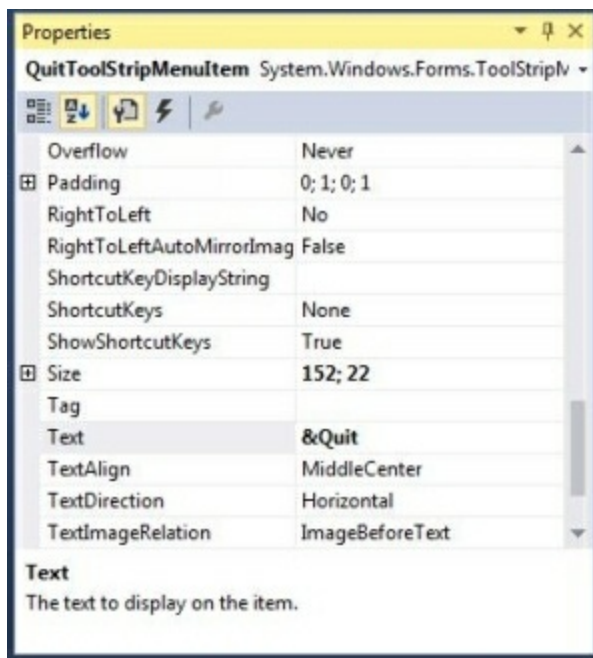
## Underline Shortcut

Menus items usually have their shortcuts. These shortcuts are the underlined

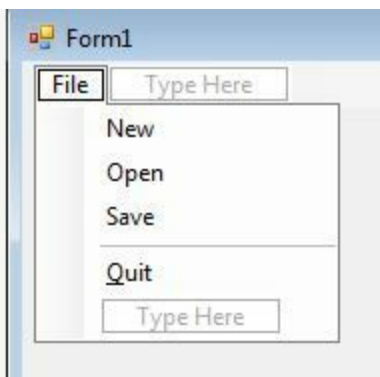


letters that you see when you click a menu. To add an underline to your *Quit* menu item, do this:

1. Click on your *Quit* menu item
2. Locate the *Text* property
3. Type an ampersand symbol (&) before *Quit*



Your menu should now look like this:



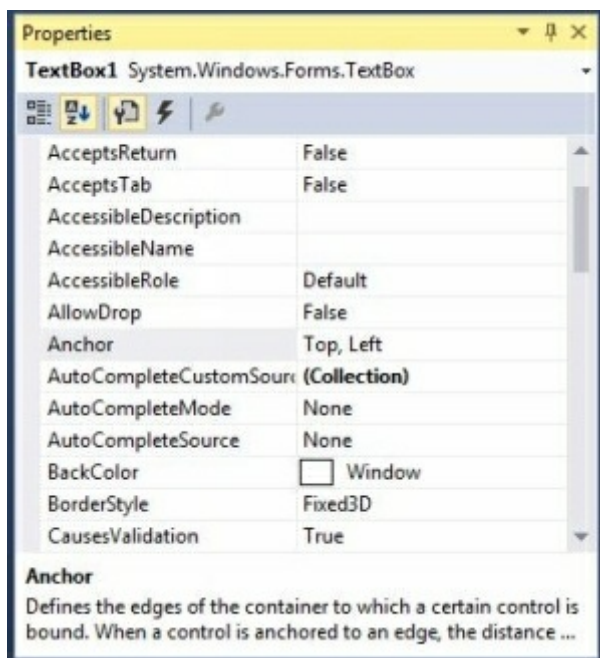
Run your program. To use the underline shortcut, hold down the Alt key on your keyboard and type the underlined character (Q in our case).

## Anchoring and Docking

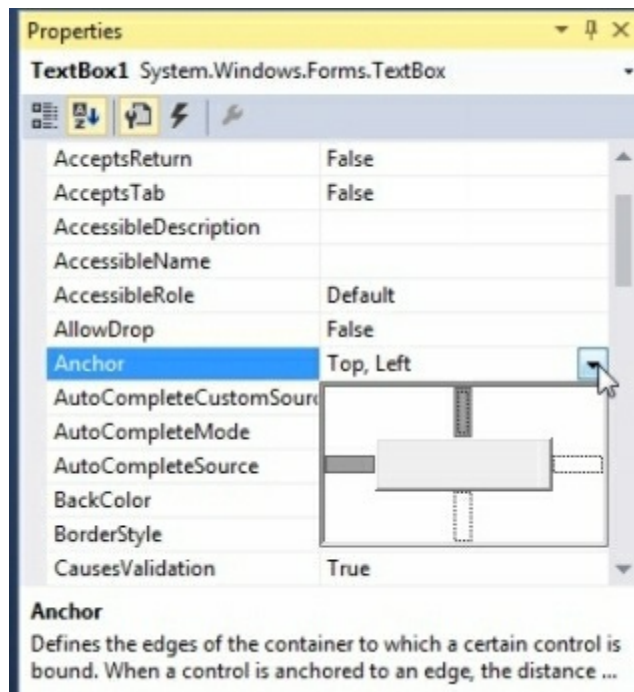
A common issue in form design is the design of forms that can be resized. You might design a nice form for a given size, but when it is resized, the controls are all grouped up in the upper left corner. Visual Studio provides several ways for designing forms that scale properly. The two most important of them are the *Anchor* and *Dock* properties.

The *Anchor* property lets you attach one or more edges of the control to corresponding edges of the form. The anchored edges of the control maintain the same distance from the corresponding edges of the form.

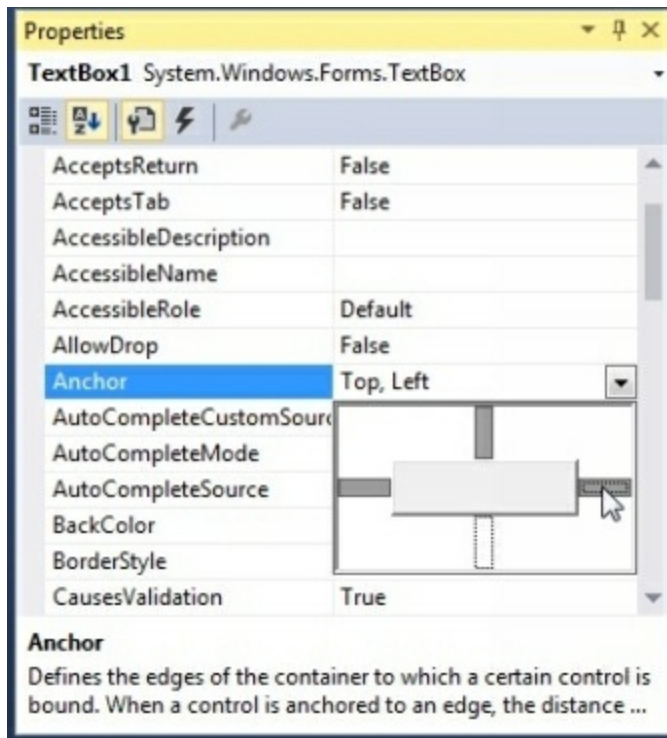
Create a new project. Place two textboxes on the new form and set the *MultiLine* properties of both to *True*. Next, change the height of the boxes. Click on *TextBox1* and select the *Anchor* property in the *Properties* window:



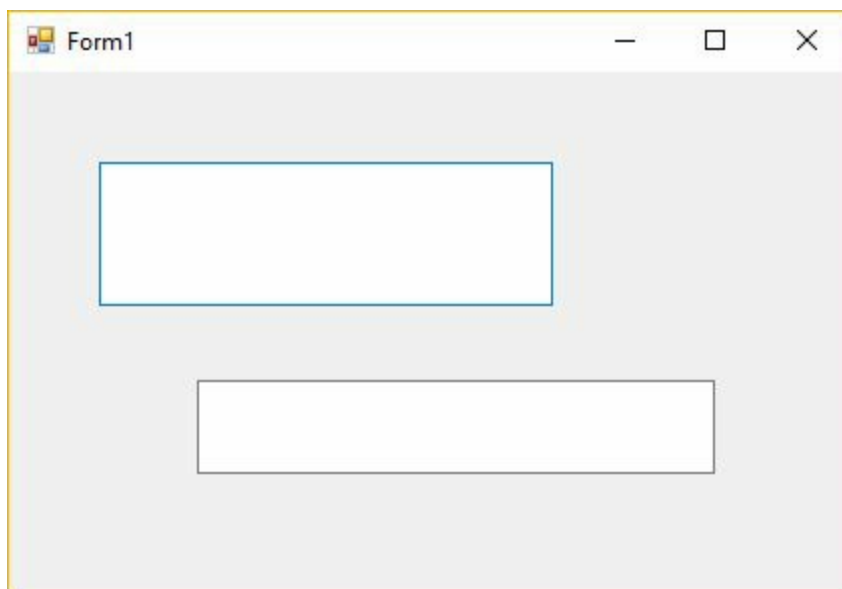
When the form is resized, the control retains its size and its distance from the upper-left corner of the form. Click the arrow on the drop down box:



The object in the middle represents your object. If you want to change the property, just click the smaller grey or white rectangles between the big white rectangle. In the image below the *Anchor* property has been changed so that the text box is at the *Top Left* and *Right* side of the form:



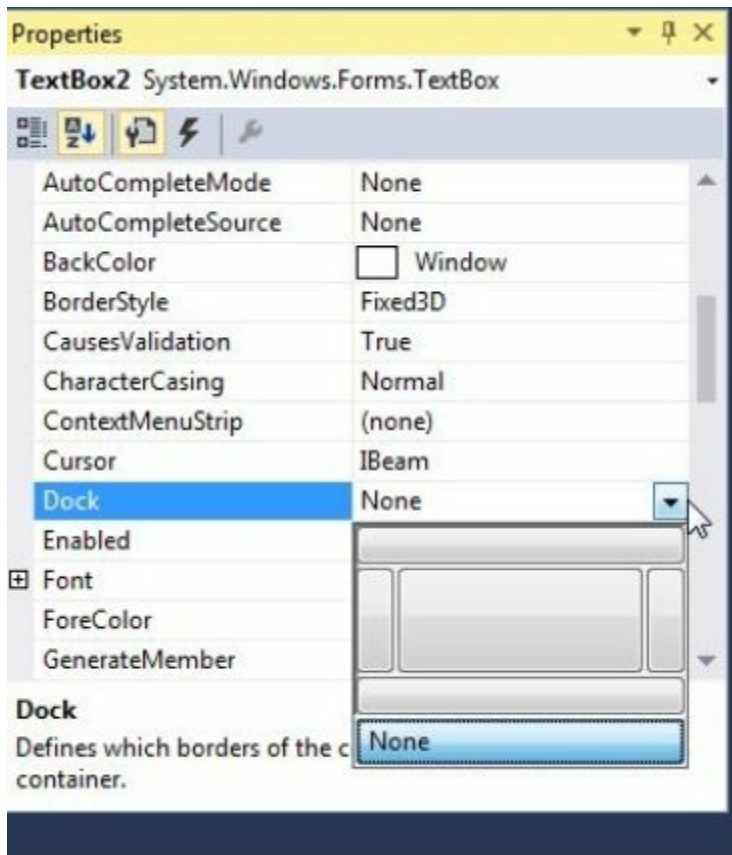
If you want to see the effect of the changes made above, set the anchor property of *TextBox1* to the default (*Top, Left*) and the Anchor property of *TextBox2* to *None*. Run your program and drag the edges of the Form in every direction. This will resize your Form object. You will see that *TextBox1* stays where it is and that the left edge of *TextBox2* moves:



# Docking

In addition to the *Anchor* property, most controls provide a *Dock* property, which determines how a control will dock on the form. The default value for the *Dock* property is *None*.

Create a new form and place a TextBox control on it. Next, locate the *Dock* property for the control:



A control can be docked to one edge of its parent container or can be docked to all edges and fill the parent container. Use the Dock property to define how a control is automatically resized as its parent control is resized.

Try all combinations and see what will happen.

## NOTE

You can dock only to one side at a time.

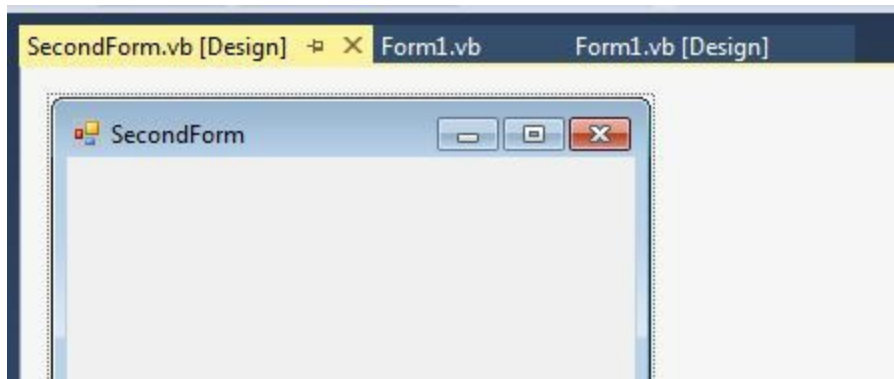
# Creating multiple forms

Most applications in Visual Basic 2015 are made up of multiple forms. In this section you will learn how to create an application that has more than a single form.

Select *File > New Project* from the main menu and create a *Windows Form Application*. A form will be created with a default name of *Form1*.

Click on *Project > Add Windows Form* from the menu bar. Change the default form name to *SecondForm* and click *Add*.

Your new form should look like this:



You can switch between forms by clicking on the tabs.

Now, we will write some code to get this new form to display.

In Visual Basic 6.0, if you had a second form in your project, then displaying it was as easy as using *Form2.Show*. However, that code is no longer valid in Visual Basic 2015 due to some changes in how forms are handled.

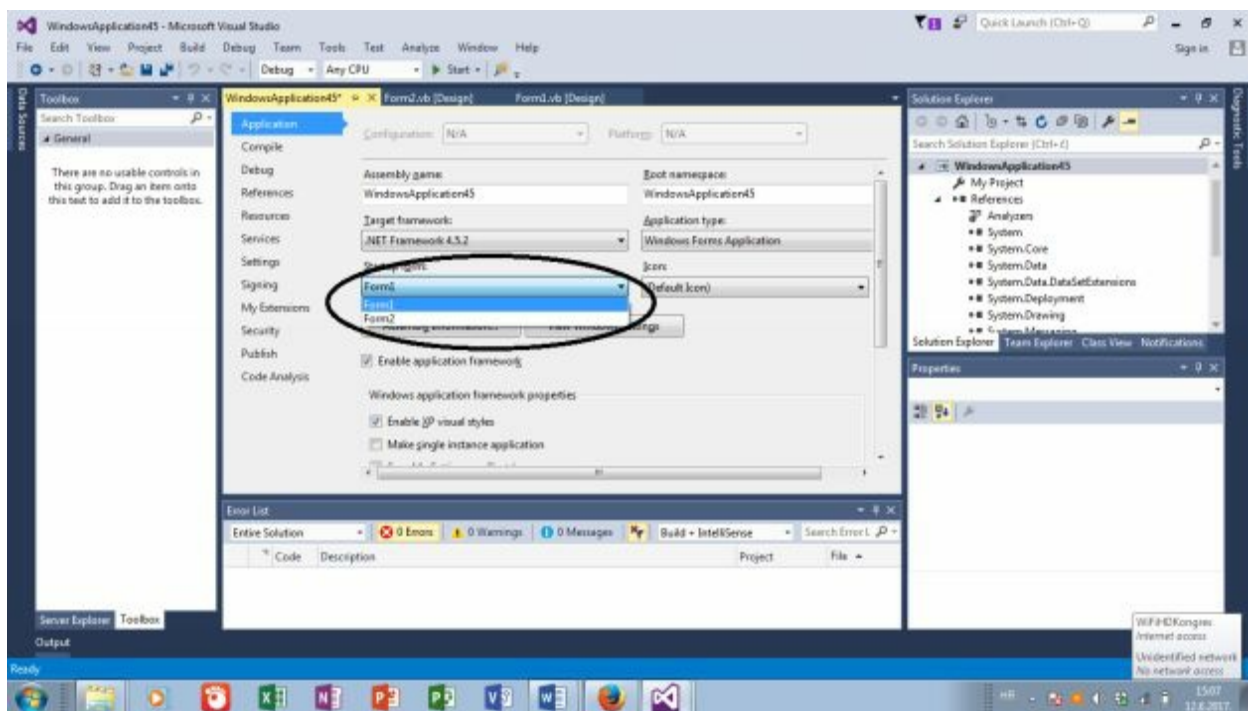
We have already learned that forms are Classes. So, you first have to create a new object called *SecondForm*. Add the button to the first form and type this code:

```
Dim myForm As New SecondForm  
myForm.Show()
```

The first line tells Visual Basic 2015 to define an object called *myForm* that will represent the form that you want to open, in this case *SecondForm*. The second line instructs Visual Basic to show the form represented by the object *myForm*.

When you are finished adding the code, run the application and see how it works. When an application starts, the main form (*Form1*) is loaded. When the button is clicked, the second form will open.

You can control which form is initially loaded by setting the startup object in the project Properties window:



## Modal forms

Modal forms are forms that need to be closed or hidden before a user can continue working. Some examples of modal forms are dialog boxes and message boxes.

You can call a modal form in two ways:

- by calling the *ShowDialog* method

- by calling the *Show* method

Let's start our example by creating two forms, three labels, three buttons and one textbox.

The properties set for these controls are as follows:

### **Form 1**

#### **(Button 3)**

Name: btnName

Text: Enter Your Name

#### **(Label 1)**

Name: lblWelcome

Text: Welcome to Geek University

### **Form 2**

#### **(Button1)**

Name: btnOK

Text: OK

#### **(Button 2)**

Name: btnCancel

Text: Cancel

#### **(Label 2)**

Name: lblName

Text: Enter Your Name

#### **(TextBox1)**

Name: txtName

Text: leave default



Now, double click on your Form2's OK button and add the following code:

```
Private Sub btnOK_Click(sender As Object, e As EventArgs) Handles btnOK.Click
```

```
    Form1.lblName.Text = txtName.Text
```

```
    Me.Hide()
```

End Sub

Inside your Cancel button add the following code:

```
Private Sub btnCancel_Click(sender As Object, e As EventArgs) Handles btnCancel.Click
```

```
    MessageBox.Show("Please enter your name")
```

End Sub

Type the follow code inside the button on the first form:

```
Private Sub btnName_Click(sender As Object, e As EventArgs) Handles btnName.Click
```

```
    Dim frmSecond As New Form2
```

```
    frmSecond.Show()
```

End Sub

When the above code is executed, you should get the following:



Click on the Enter your Name button:



A Windows-style dialog box titled "Form2". It has a light gray background and a blue border. Inside, the text "Enter Your Name" is displayed above a white text input field. Below the input field are two buttons: "OK" on the left and "Cancel" on the right.

Clicking on the OK button takes the control and information back from the Form2 (**modal form**) to the first form. You should get something like this:



A Windows-style application window with a title bar showing the URL "http://geek-university.com" and standard minimize, maximize, and close buttons. The window content area has a light gray background. It displays the text "Welcome to Geek University" followed by "John" on the next line. At the bottom, there is a button labeled "Enter your Name".

# **Chapter 10 - Error Management**

## **IN THIS CHAPTER**

**Types of errors**

**Using breakpoints in your code**

**Try...catch statement**

Bugs in programming happen. Programmers aren't perfect and we can't anticipate all the possible conditions that might happen in our applications. Fortunately, Visual Basic offers a rich set of debugging tools that can help you in discovering the bugs.

Errors in Visual Basic can be categorized into three types:

- Design Time errors
- Run Time errors
- Logical errors

## Design Time errors

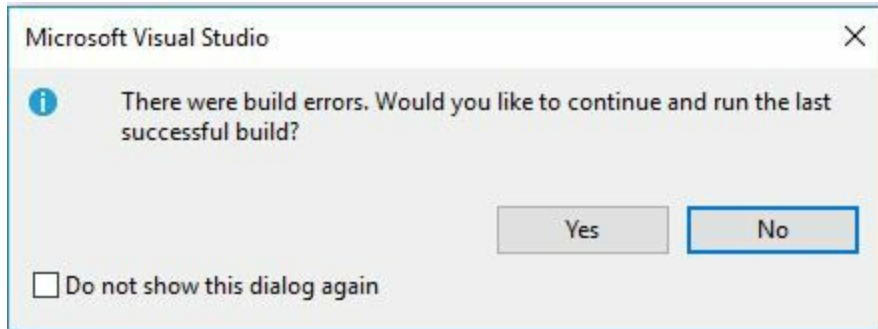
Design Time errors also called syntax errors. The most common errors of this type are the result of typos. These errors are easy to track, because you will get a red line in Visual Studio pointing to them.

Here is an example:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim firstName As Stirng

End Sub
```

As you can see, the (red) wavy line alerted you that this line of code contains a typing (syntax) error.



## Run Time errors

Run Time errors are errors that occur when you ask Visual Basic to run the application. Before the program can be run, the source code must be compiled into the machine code. If the conversion can not be performed, you will get a notification from Visual Basic informing you that your application can not be run before the error is fixed. An example of a runtime error is the division by zero. The following code will produce a runtime error, so try it and see what happens:

```
Dim x As Integer
```

```
Dim y As Integer
```

```
x = 2
```

```
y = 0
```

```
MsgBox(x / y)
```

## Logical errors

Logical errors are errors that occur after the code has been compiled and the program is running. The error of this type will cause your program to behave unexpectedly or even crash. Logical errors can be difficult to find because they do not produce an error message and unlike a program with syntax errors, a program with logic errors can be run.

The following code is an example of a logical error, so try it and see what happens:

```
Dim x As Integer
Dim y As Integer
Dim res As Integer
x = 3
y = 2.5
res = x * y
MsgBox(res)
```

## Breakpoints

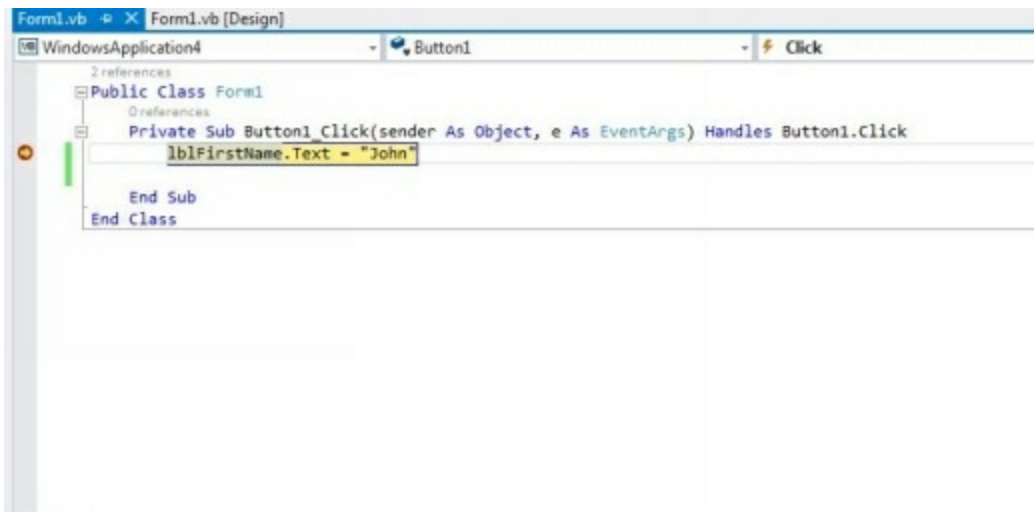
While debugging a program, you can interrupt its execution by inserting a **breakpoint**. When the breakpoint is reached, the program's execution is paused. Any statement that can appear in your VB code can also be executed in the Immediate window.

You can set a breakpoint by clicking on the margins. The following image shows how to add one:

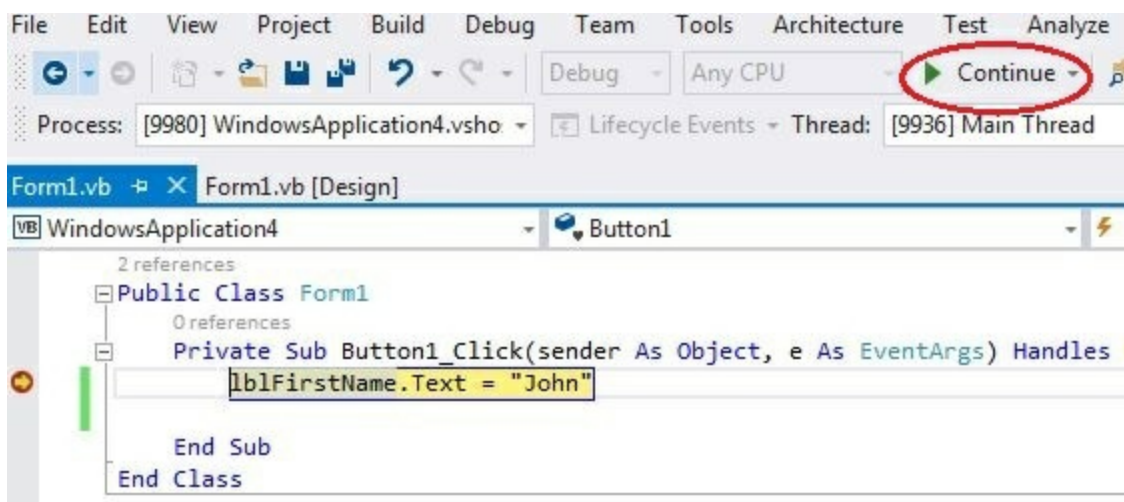


The line where you want Visual Basic to break is highlighted in brown. Press

F5 to start the application and click on the button. The computer begins processing the code contained in the *Button1\_Click* procedure. It stops processing the code when it reaches the *breakpoint* statement. The highlighting indicates that the statement is the next one to be processed. The yellow arrow now appears in the red dot next to the breakpoint.



The next line of code will be executed when you click *Continue* or F10 on your keyboard.



Here is our form with the label showing the word *John*:



## NOTE

To remove the breakpoint, simply click on the breakpoint circle.

## Try...Catch

The *Try...Catch* statements are used to handle errors (also known as exceptions) in Visual Basic. These statements tell Visual Basic 2015 what to do when an exception is encountered; for example, when a file you are trying to open doesn't exist. This act of detecting and processing an exception is called **exception handling**. In this section, I will explain how to use the *Try...Catch* statement to trap errors and exceptions in an application.

You can tell Visual Basic to *Try* some piece of code. An error that occurs while an application is running is called an *Exception*. *Exception* is inbuilt class that deals with errors.

The basic syntax of the Try...Catch statement is:

Try

' Code that might trigger an exception

Catch ex As Exception

'one or more statements to execute when an exception occurs

End Try



Take a look at the following example:

Try

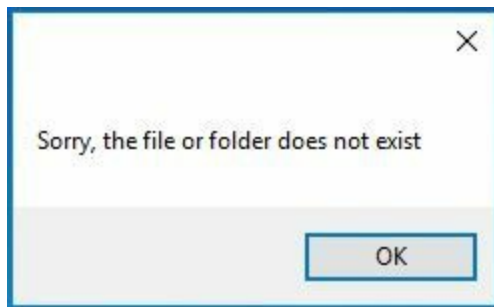
```
Dim sr As StreamReader = File.OpenText("C:\\test.txt")
Console.WriteLine("The first line of this file is:", sr.ReadLine())
sr.Close()
Catch ex As Exception
    MessageBox.Show("Sorry, the file or folder does not exist")
```

End Try

We also need to use the following *imports* statement on the top of our code:

```
Imports System.IO
```

Enter the code above and press F5 on your keyboard to run the application. The code above uses a *Try...Catch* block to catch a possible exception. The method contains a Try block with a *StreamReader* statement that tries to opens a data file called *test.txt* and read a string from the file. When the application is run, you will see a message like this:



Let's break down the code above:

Inside the Try block, we've placed the code that could possibly generate an exception:

```
Dim sr As StreamReader = File.OpenText("C:\\test.txt")
    Console.WriteLine("The first line of this file is:", sr.ReadLine())
    sr.Close()
```

When an exception occurs in the *Try* block's code, the computer processes the code contained in the *Catch* block, which prevents the program from crashing.

```
MessageBox.Show("Sorry, the file or folder does not exist")
```

# **Chapter 11 - Function and Subs**

## **IN THIS CHAPTER**

**Explaining functions and subs**

**Passing arguments to functions**

**Using modules in your code**

A procedure is a block of Visual Basic statements that together perform a task when they are called.

In Visual Basic, there are two types of procedures:

- Subs (shortened for Subroutine)
- Functions

The main difference between functions and subs is that functions return a value while subs do not.

## Sub procedures

Subs are procedures that don't return a value. In this lesson you will learn how to create your own subs.

Each time a sub procedure is called, its statements are executed, starting with the first executable statement after the *Sub* statement. You can define a sub procedure in classes, modules and structures. By default, a sub is *Public*, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it.

### NOTE

The term *method* describes a sub or function procedure that is accessed from outside its defining module, class, or structure.

Let's create an example sub. Start a new Visual Basic project and add a button to the form.

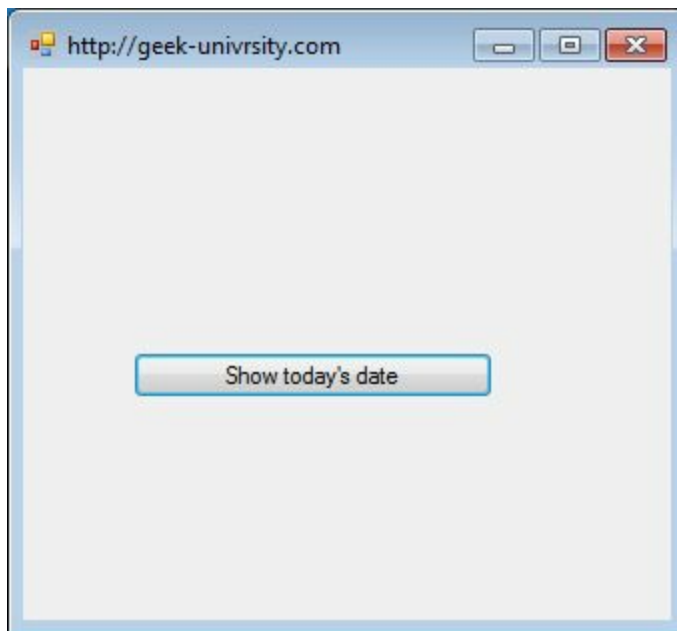
Change the default properties set for this button:

**(Button)**

Name: btnDate

Text: Show today's date

The form should look like this:



To write your own sub, your cursor needs to be outside of the button code.



This is the sub's code:

```
Private Sub ShowDate()
```

```
    MessageBox.Show("Today's date is " & Now().ToShortDateString)
```

```
End Sub
```

Now, run your program and see what happens. Nothing, right? Why is that? Well, to invoke your sub procedure, you need to call it. Double-click the button and paste the following code inside the button click event:

Call ShowDate()

## NOTE

You don't need to use word *Call* to call a sub We are using this word because it makes our code easier to read.

When you've finished typing it all, your code window should look like :

```
Public Class Form1
```

```
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
    End Sub
```

```
    Private Sub btnDate_Click(sender As Object, e As EventArgs) Handles btnDate.Click
```

```
        Call ShowDate()
```

```
    End Sub
```

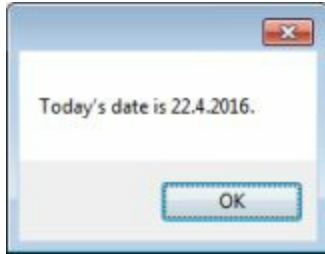
```
    Private Sub ShowDate()
```

```
        MessageBox.Show("Today's date is " & Now().ToShortDateString)
```

```
    End Sub
```

```
End Class
```

Save the program and start the application. Click the button and see what happens. You should get the **MessageBox** displaying the today's date.



## ByVal and ByRef

In Visual Basic, you can pass an argument to subs and functions by value or by reference. This is known as a passing mechanism, and it determines whether the procedure can modify the programming element underlying the argument in the calling code.

*ByVal* means that you are passing a copy of a variable to your sub or to your function. You can make changes to the copy and the original will not be altered.

*ByRef* is short for *By Reference*. This means that you are not handing over a copy of the original variable but pointing to the original variable.

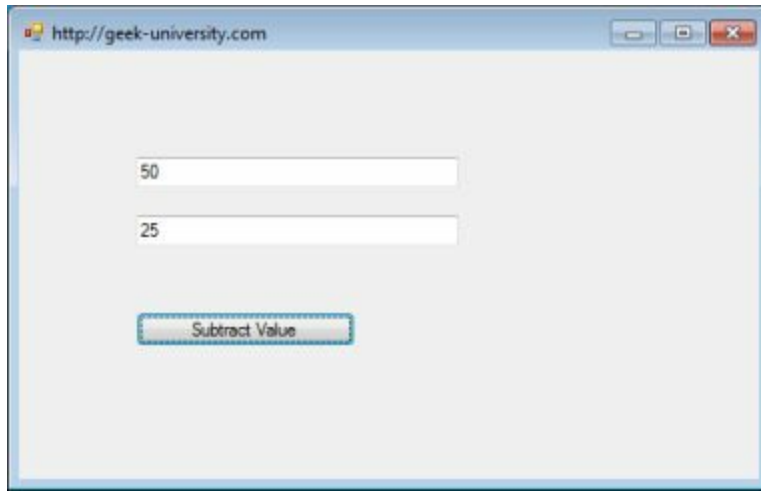
Let's look at an example for *ByVal*. Add a button and two textboxes to your form. Name the button *btnSubtract*. Double click the button and type the following code inside the click event:

```
Private Sub btnSubtract_Click(sender As Object, e As EventArgs) Handles  
    btnSubtract.Click  
        MessageBox.Show(subtractNumbers(TextBox1.Text, TextBox2.Text))  
    End Sub
```

Next, create a simple function outside the button:

```
Private Function subtractNumbers(ByVal num1 As Integer, ByVal num2 As Integer)  
    Return num1 - num2  
  
End Function
```

Press F5 on your keyboard to test your application. Enter some values inside the textbox:



Now, click on your button and see what will happen.



As you can see in the picture above, when we clicked on the button, the values from the textboxes were forwarded to the *subtractNumbers* function which then returned the result of subtraction and showed it in the messagebox.

*ByRef* is an alternative to *ByVal*. Consider the following example:

```
Public Class Form1
```

```
    Dim num1 As Integer = 4
```

```
    Sub addnumber(ByRef num2 As Integer)
```

```
        num2 = num2 + 1
```

```
    End Sub
```



```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Call addnumber(num1)
    MessageBox.Show(num1)

End Sub

End Class
```

If you run a program and click on **Button1**, the *addnumber* sub will be called and you will get number 5 as a result.

## Functions

Functions are similar to subs. The main difference between functions and subs is that a function returns a value while a sub does not.

A function can be a part of a Module, Class or Structure. It can be called from other functions, subs or properties. A function can be used multiple time throughout a program.

The syntax of a function:

```
[ accessibility ] Function function_name [ paramaters ] [As return_type ]
    statements
End Function
```

The accessibility of a function can be Public, Protected, Friend, Protected Friend, or Private and determines which pieces of code can invoke the function.

Let us write some code as an example:

```
Public Class Form1
    Dim labeltext As String = "Working with functions in Visual Basic 2015"
```

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
End Sub
```

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
    Label1.Text = changelabeltext()
```

```
End Sub
```

```
Public Function changelabeltext()
```

```
    Return labeltext
```

```
End Function
```

```
End Class
```

After you finish adding the code, click on *Debug > Start* to have a look at your program. When you click on the button, the *changelabeltext* function is called, which will alter the default *Label1* text property to *Working with functions in Visual Basic 2015*.

## Modules in Visual Basic 2015

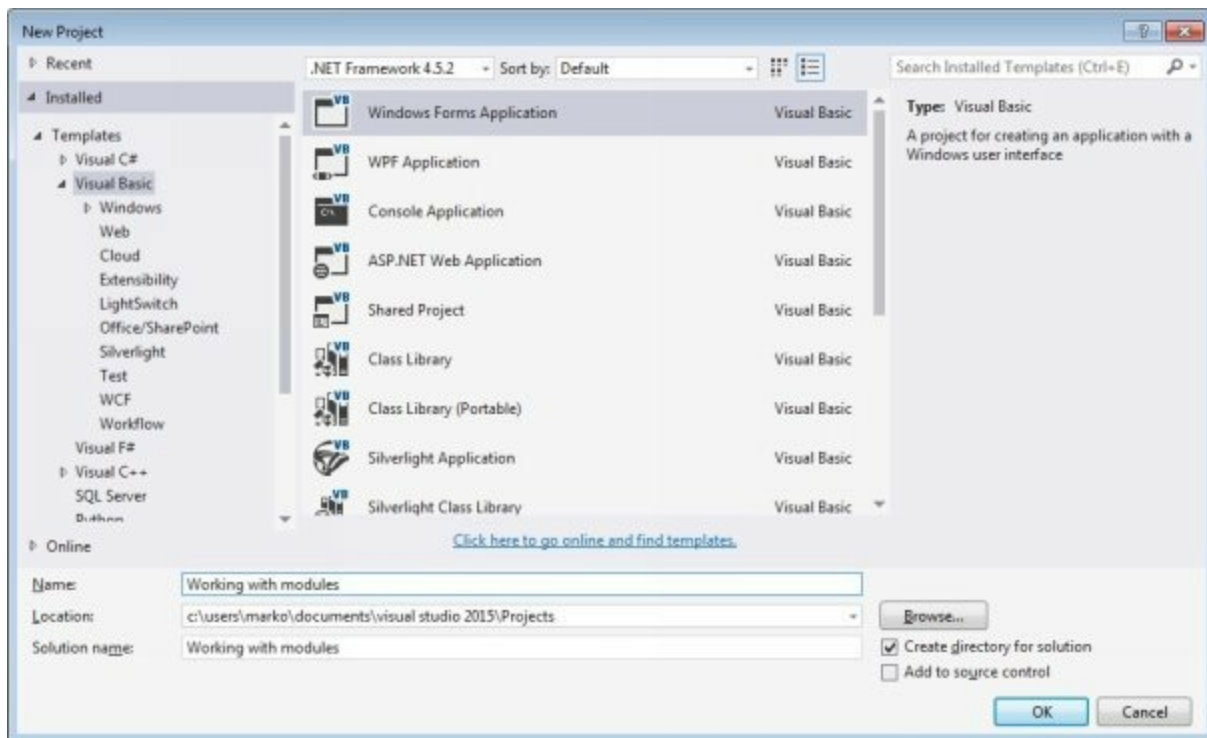
In this section we will learn about modules - what they are, how to create one, how to store information inside a module and then access that information in your application, etc.

A module can be used to store functions, subs or even variables to sort out your code so that you can access things more easily. If you have a complex application where you have to store a lot of functions, subs or variables, it can be useful to sort them into different categories using modules, instead of writing code over and over again.

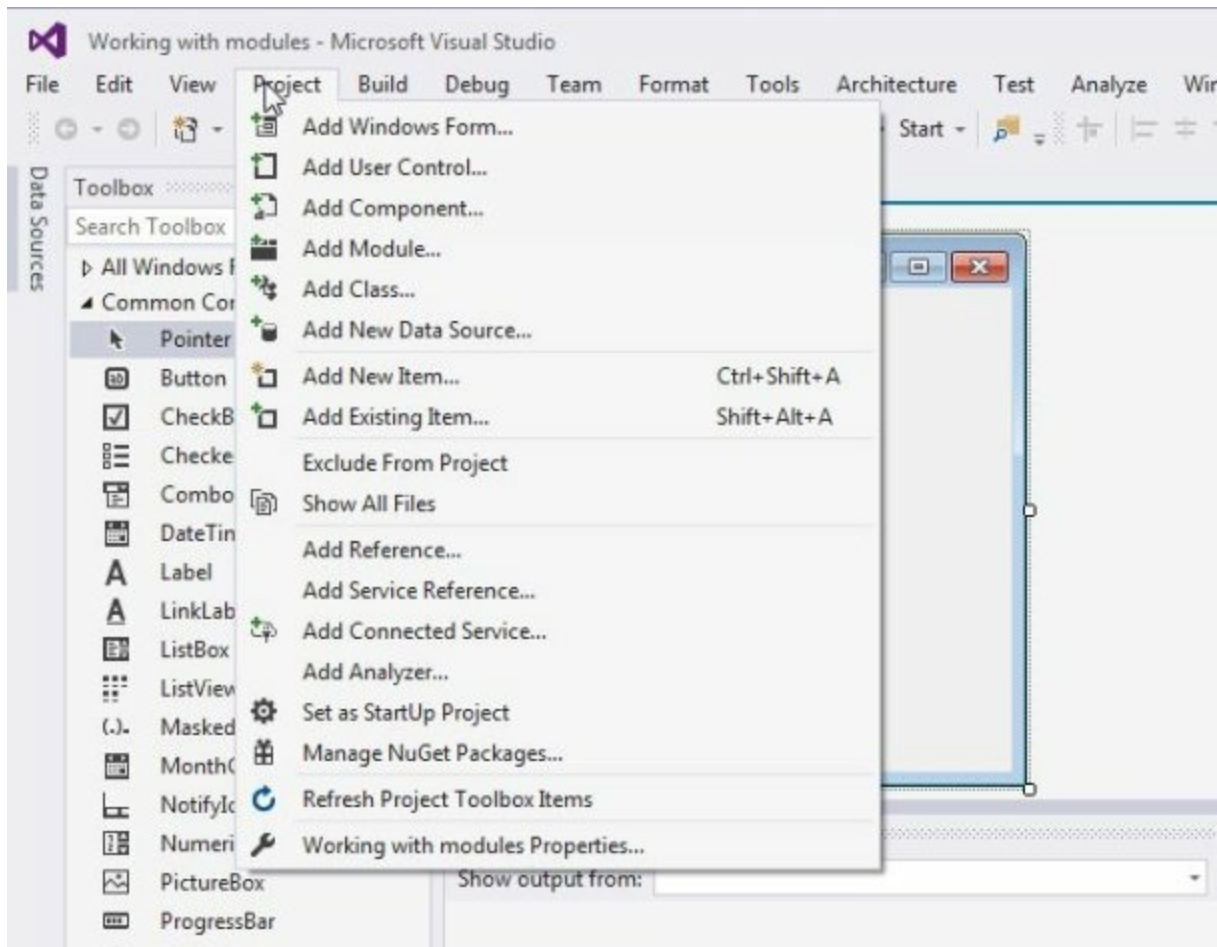
A module is different than a class because all the members of a module are

shared. A module also does not have a constructor, which means that you can not create an instance of a module.

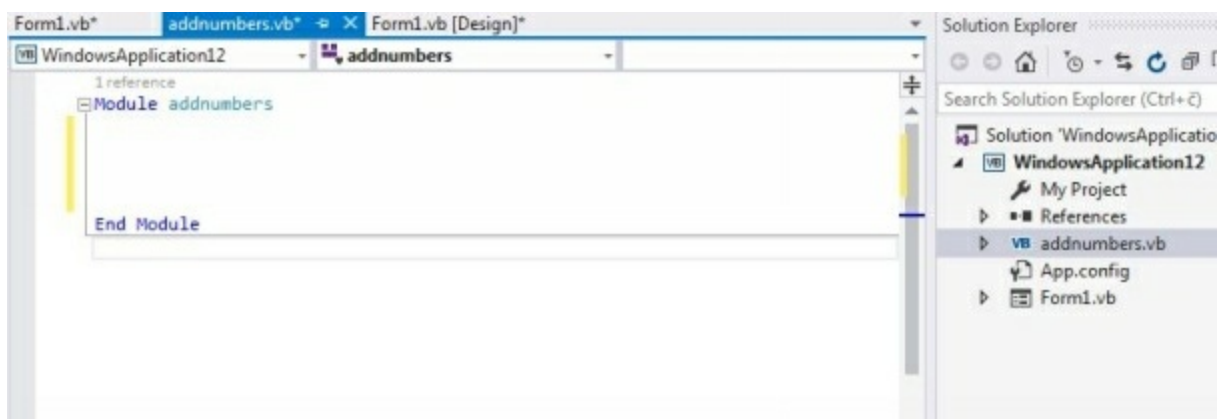
Let's create our first module. Create a new project and name it *Working with modules*:



Once the new project has been opened, select *Project* > *Add* module:



Name the new module *addnumbers.vb* and click the *Add* button. The new module will be added to the project and a new tab called *addnumbers.vb* for accessing the module code should appear in the design area:



Put the following controls on the Form and change their properties to the following:

**(Button)**

**Name:** btnAddNumbers

**Text:** Add

**(TextBox1)**

**Name:** leave default

**Text:** leave default

**(TextBox2)**

**Name:** leave default

**Text:** leave default

Now, inbetween the *Module addnumbers* and *End Module* keywords, write the following function:

Module addnumbers

Public Function sumnumbers(ByVal number1 As Integer, ByVal number2 As Integer)

Return number1 + number2

End Function

End Module

Put the following code for your button:

MessageBox.Show(addnumbers.sumnumbers(TextBox1.Text, TextBox2.Text))

Press F5 to build and run the application. When the application is running, pressing the button will cause a message window to appear. The window

should display the sum of two numbers that you've entered in the textboxes:



# **Chapter 12 - Classes and Objects**

## **IN THIS CHAPTER**

**Explaining classes and objects**

**Creating your own classes**

**Creating properties and methods in classes**

In this section you will learn how to write your own Visual Basic classes and how to create objects from them. A class is the code that defines an object, and all objects are created based on a class. Each object has its own set of properties and methods that it will respond to.

Classes are the core of any object-oriented language. You will find yourself using classes whenever you write any of programs in Visual Basic 2015. The Windows Form itself is a Class and when you run your application you are creating Objects: a Form object, button object, label object etc.

## NOTE

A big advantage of creating classes is that you can reuse their code whenever you need it in some other projects.

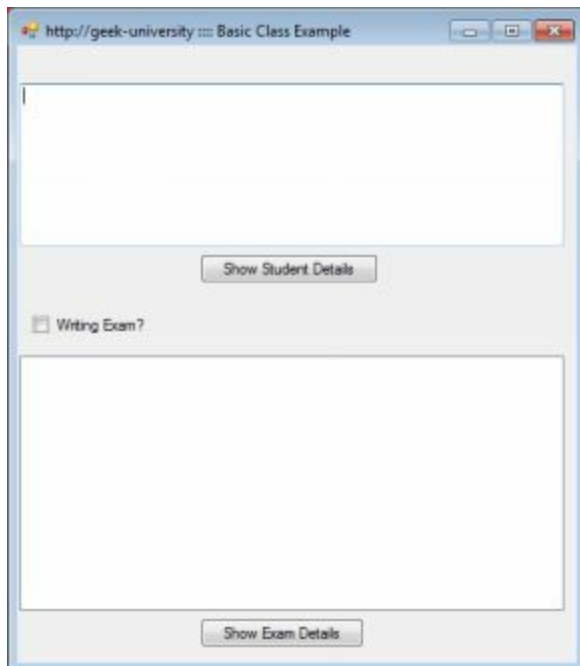
## Create a Class

In this example we will create a class called *Student*. At first, the class will have minimum functionality, but we will keep on adding features to it. The name of a class can be anything - just make sure that it's suggestive of the class's functionality.

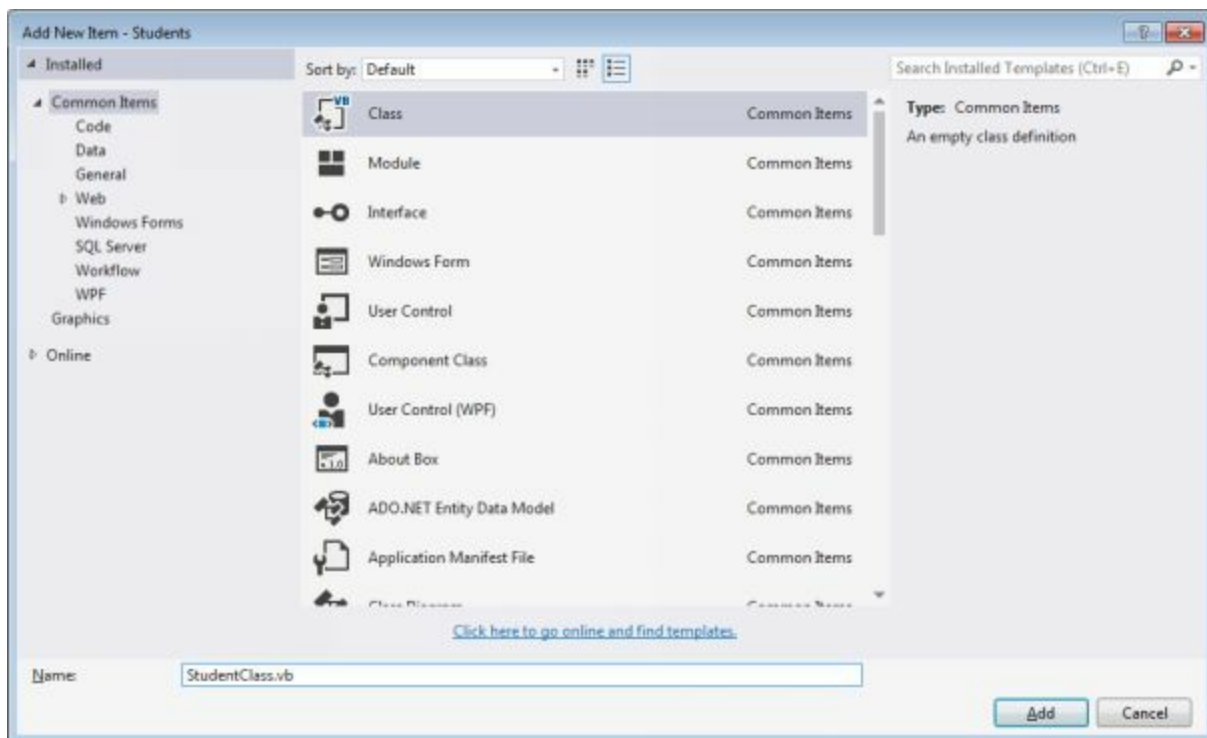
Create an application and name it *Student*. Add a text box, a label, a listbox, two buttons and one check box to the form. Leave the default properties, only change the text properties.

Your form should look something like this:

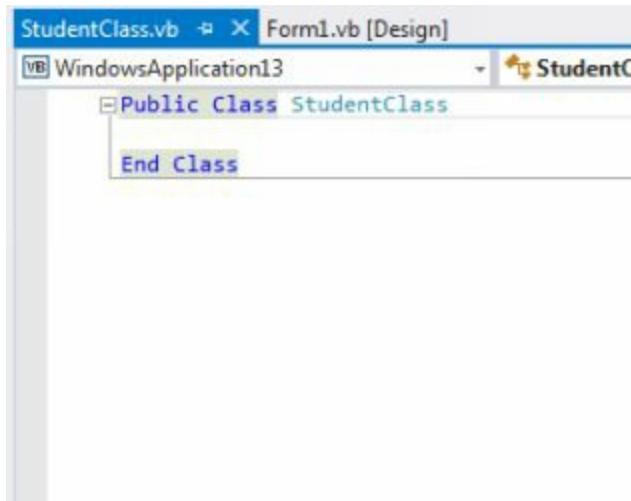




Next, go to *Project > Add Class*. You should get the following window:



Change the name of the class from *Class1.vb* to *StudentClass.vb* and click the *Add* button. You should see the default code window for your *StudentClass*:



When you are designing a Class, your code has to go inside functions and subs. In the next sections I will describe how you can create your own properties for the objects.

## Create Properties in VB .NET Classes

Properties are used in Visual Basic to describe objects. For example, a Text is a property of a Button object, just like its height, font, and size. You can have different properties set up in your class and provide different settings for each object.

In your class you can either create a field (e.g. a Public variable), or you can create a Private variable and expose the Private variable using a Property statement.

Edit the existing class in your project so it looks like the code shown below:

```
Public Class StudentClass 'Represents A Student
    Private strStudentName As String ' Student Name
    Private strStudentSurname As String ' Student Surname
    Private strStudentNumber As String ' Student Number
    Private blnWriting As Boolean 'Is Student Writing The Exam?
```

Public Property StudentName As String 'Student Name Property

Get

Return strStudentName

End Get

Set(value As String)

strStudentName = value

End Set

End Property

Public Property StudentSurname As String ' Student Surname Property

Get

Return strStudentSurname

End Get

Set(value As String)

strStudentSurname = value

End Set

End Property

Public Property StudentNumber As String ' Student Number Property

Get

Return strStudentNumber

End Get

Set(value As String)

strStudentNumber = value

End Set

End Property

Public Property WritingExam As Boolean 'Writing Exam Property

Get

Return blnWriting 'Get Setting

End Get

Set(value As Boolean)

blnWriting = value ' Provide Setting

End Set

```
End Property  
End Class
```

The code above is a little bit long, but don't worry - here is an explanation of the most interesting parts:

*StudentName*, *StudentSurname* and *StudentNumber* - these are the names of our properties.

Within the Get and Set parts we have the following code:

```
Get  
    Return strStudentName  
End Get  
Set(value As String)  
    strStudentName = value  
End Set
```

*Get / End Get* - inside this block we are telling Visual Basic that we want to read from a property.

*Set / End Set* - the code inside Set and End Set keywords allows us to Set a value for our property.

## Create Methods in your VB .NET Classes

You have already used methods in your code. Remember, a method is a predefined procedure that you can invoke when needed. For example, if you want the computer to close the current program when the user clicks the Exit button, you need to use the *Me.Close()* method in the button's Click event.

A method created in a Class is nothing more than a function or a sub. So add the

following code to the Class you've created in the previous lesson:

```
Public Function WriteExam()
```

```
Dim strWriting As String = "" 'String To Be Returned
```

```
If blnWriting Then 'If Writing
```

```
strWriting = strStudentName & " " & strStudentSurname & ", " & strStudentNumber & " Is  
Still Writing Exam." 'String to inform about progress of exam
```

```
Else
```

```
strWriting = strStudentName & " " & strStudentSurname & ", " & strStudentNumber & " Has  
Finished Writing Exam." 'String to inform about progress of exam
```

```
End If
```

```
Return strWriting ' Return Writing String
```

```
End Function
```

This method above is very basic - it simply checks the Boolean property to see if the student is still writing his exam.

Now, let us create a new object from our Class. Add the following line of code under *Public Class Form1*:

```
Private objStudent As New StudentClass
```

Now we can now refer to our object from the Form load event. Paste this code inside Form1 load event:

```
objStudent.StudentName = "John"
```

```
objStudent.StudentSurname = "Doe"  
objStudent.StudentNumber = "123456"
```

This will set the objStudent object's properties. Add the next code inside the *Show Student Details'* button click event:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
    TextBox1.Text = objStudent.StudentName & Environment.NewLine _  
        & objStudent.StudentSurname & Environment.NewLine _  
        & objStudent.StudentNumber
```

```
End Sub
```

Now, add this code for the CheckBox:

```
Private Sub CheckBox1_CheckedChanged(sender As Object, e As EventArgs) Handles  
CheckBox1.CheckedChanged
```

```
    objStudent.WritingExam = CheckBox1.Checked  
End Sub
```

By checking the Checkbox, the *WritingExam* property will be changed to *True*. By unchecking the checkbox, the *WritingExam* property gets set to *False*. Add the final code for the *Show Exam Details'* button:

```
Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
```

```
    ListBox1.Items.Add(objStudent.WriteExam())  
End Sub
```

This code above will call the *WriteExam* method from the *Student* object.

And that's it for our introduction to classes. Note that there is an awful lot more to learn about Classes and Objects and we have just scratched the surface.



# **Chapter 13 - Visual Basic and Databases**

## **IN THIS CHAPTER**

**Database terms and concepts**

**Installing SQL Server 2014**

**Connecting applications to databases**

In this section you will learn about the basic mechanisms of interacting with databases. As you will see, it is fairly easy to write VB statements to execute SQL queries and fetch and modify rows in a database. The real challenge is the design and implementation of functional interfaces that display the data. You also need to know how to allow the user to navigate through the data, modify it, and submit the changes to the database.

## Database Terminology

Before we create some database project in Visual Basic, I will describe some basic database terms and concepts.

So, what actually is a database? A **database** is a file (container) that contains an organized collection of related information. Databases are maintained by special programs called **database management systems (DBMSs)**. Some of the most popular management systems are Microsoft SQL Server, MySQL, Oracle and Microsoft Access.

We will work with the data stored in Microsoft SQL Server 2014 databases. Databases created using Microsoft SQL Server are relational databases. A **relational database** stores information in tables composed of columns and rows.

A **field** is a single item of information about something: a person, a place, a thing... A **record** is a group of related fields that contains all data about something.

A **table** is used for storing data in the database. A relation database can contain one or more tables.

## SQL Server 2014 Installation

In the following section, you will create the simple database project which

uses a Microsoft SQL Server 2014 database named **salesdb**. But first, we need to learn how to install an SQL Server database.

First, you will need to download SQL Server 2014. You can download a 180 day trial version from Microsoft here: <https://www.microsoft.com/en-us/evalcenter/evaluate-sql-server-2014>

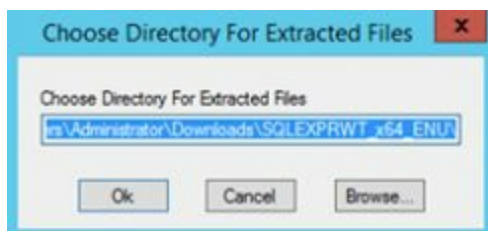
## NOTE

SQL Server requires .NET Framework 3.5 SP1. if your Windows system does not have .NET Framework 3.5 SP1 installed, you will need to download it and install it from this link: <https://www.microsoft.com/net>

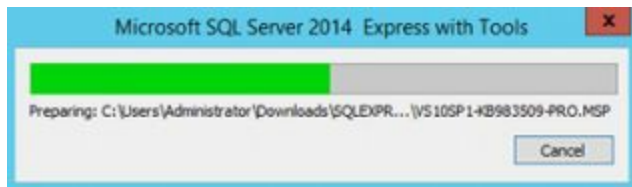
Once you've obtained the SQL Server 2014 installation file, you can start the installation. Here are the steps:

Double click on the installation file. Click *Yes* when asked *Do you want to allow the following program to make changes to this computer?*

Click **OK** to use the default directory to extract files or choose another directory:



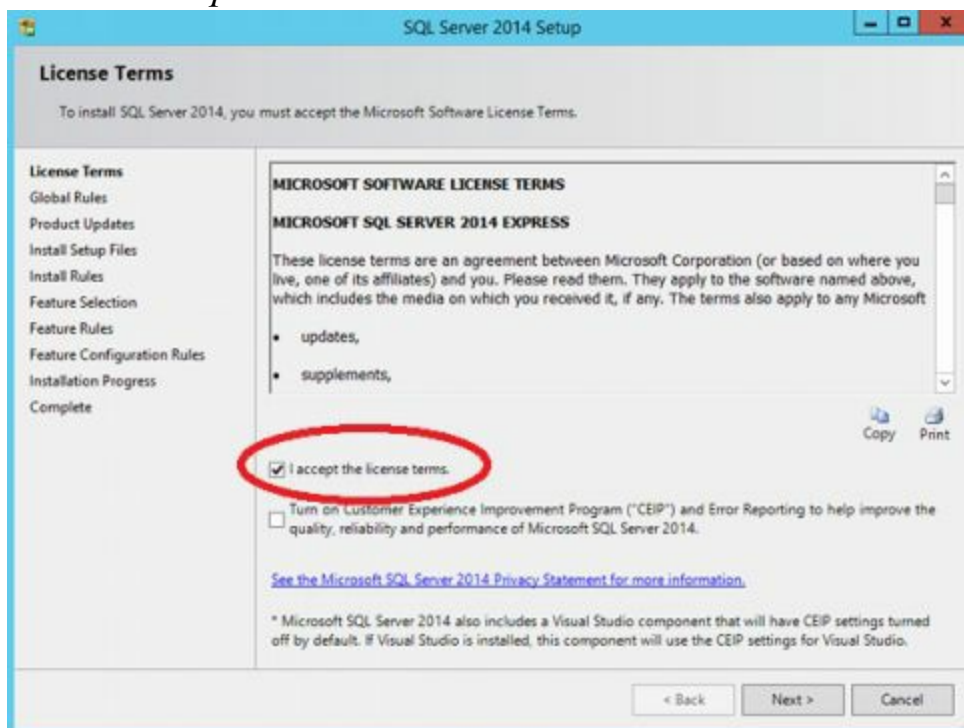
The installation should start:



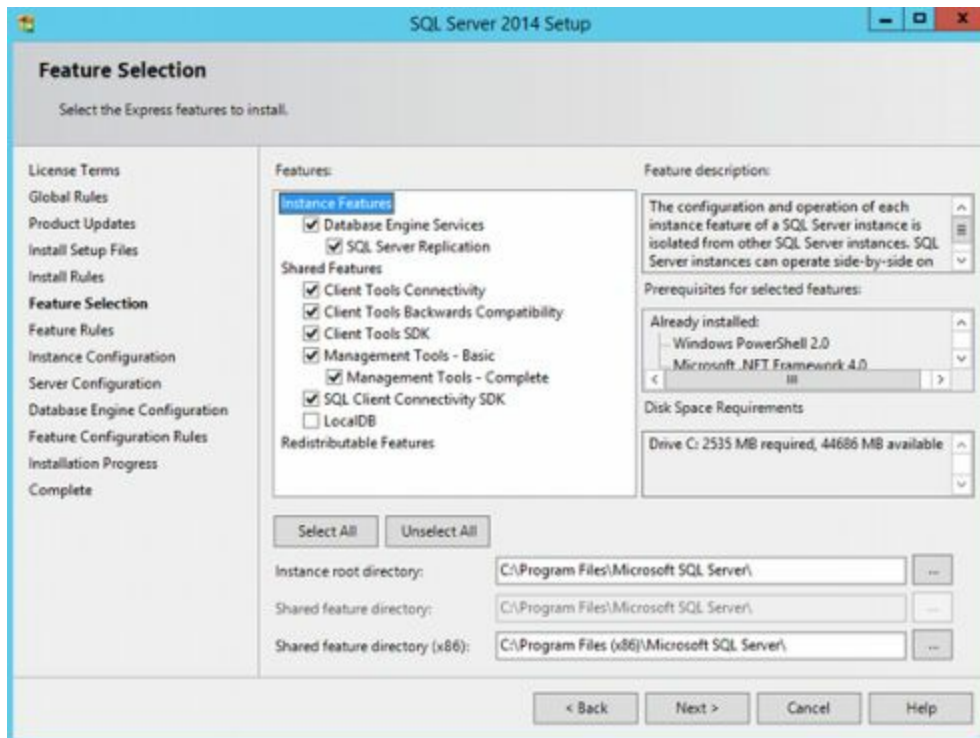
You will be asked if you want to perform a new stand-alone installation or upgrade from a previous version of SQL Server. To perform a new stand-alone installation, click *New SQL Server stand-alone installation*:



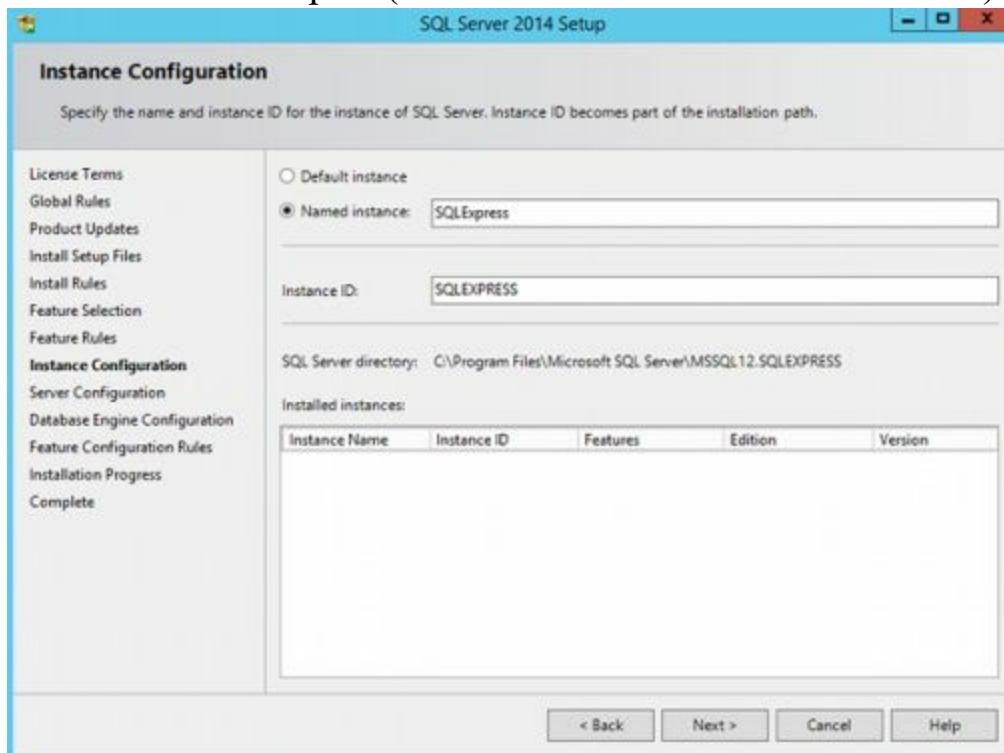
Select *I accept the licence terms* and click *Next*:



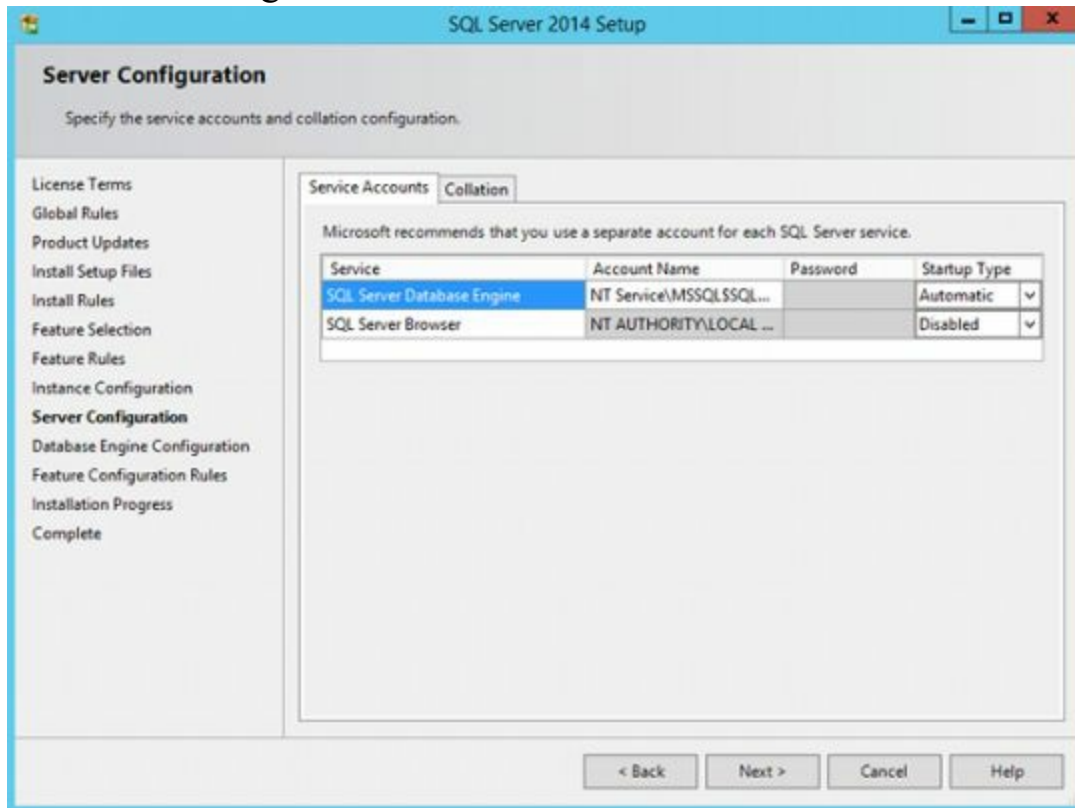
Here you can select or deselect the features you'd like to include or exclude. I recommend that you leave the defaults:



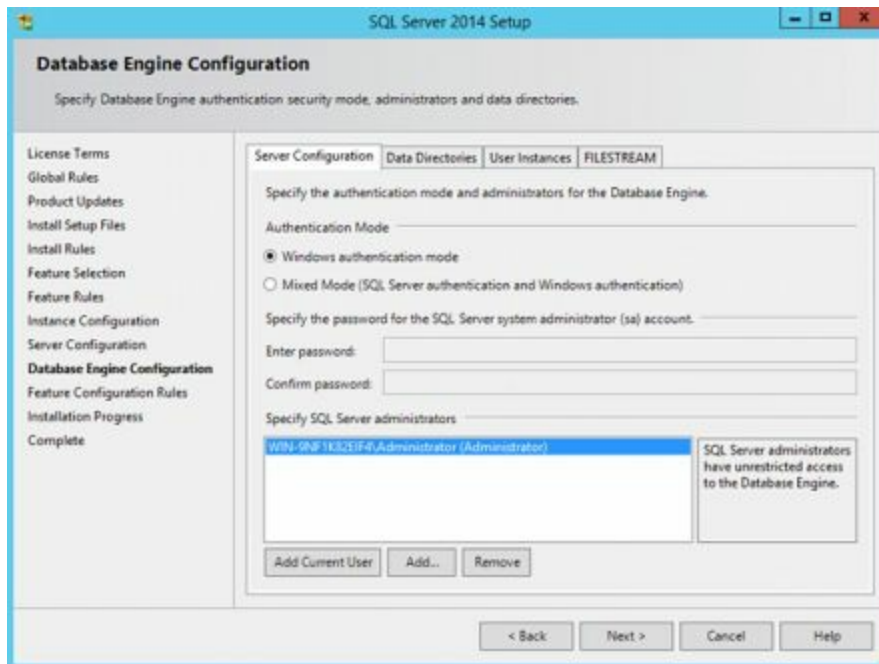
Chose the instance path (I recommend to leave it at the default) and click Next:



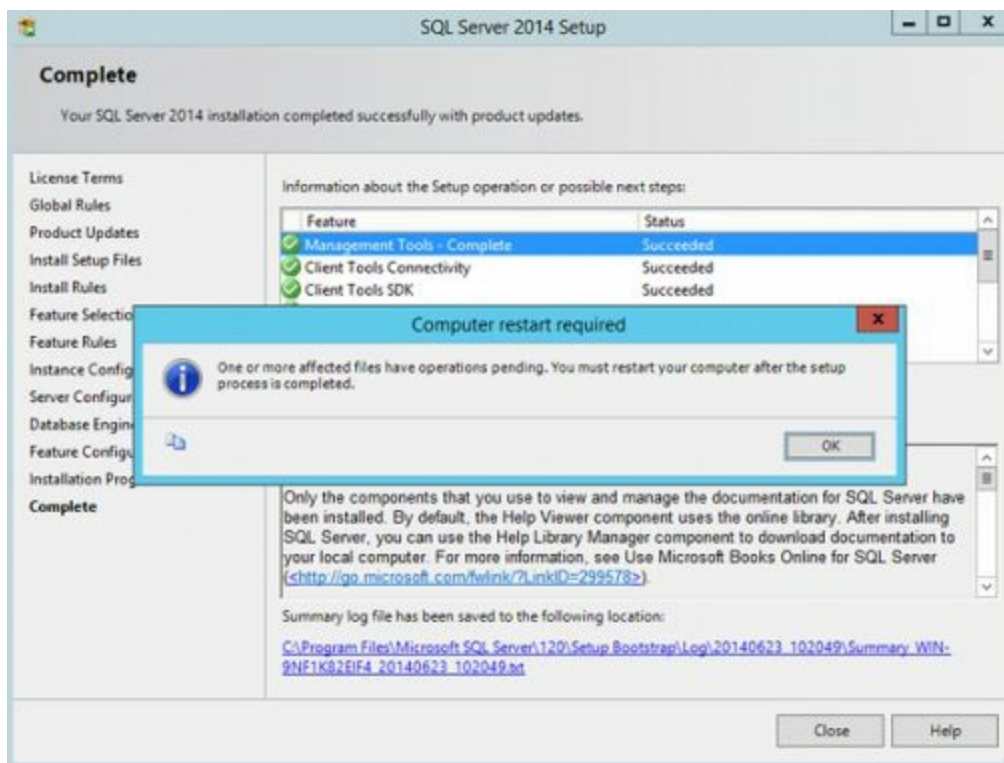
Here you can specify the service accounts and startup type, or simply accept the default configuration:



On the following screen, you can either leave the default settings or change according to your requirements. You can specify the authentication mode that SQL Server will use, as well as the SQL Server administrator. At least one system administrator must be provided. Click *Next*.



Once the installation is completed, you may be asked to restart your computer. Click OK and restart the computer:



That's it, SQL Server 2014 should now be installed. Now we can start building our database project.

## Connecting an application to an SQL Server 2014 Database

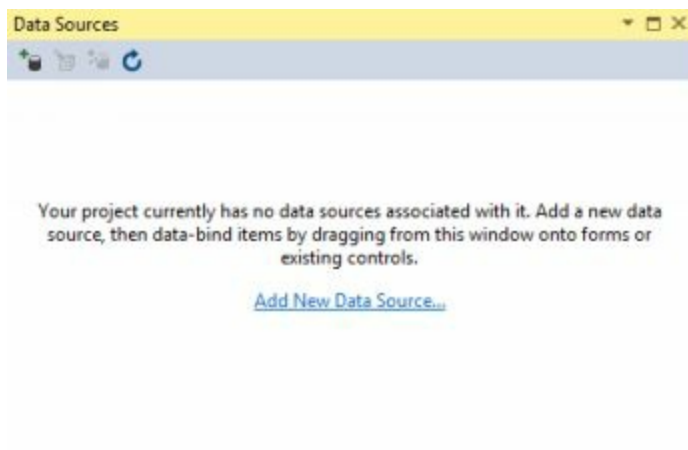
In this section we are going to create a simple Sales database application. Before starting the lesson, download the sample database:

<http://geek-university.com/uploads/SalesDBData.mdf>

Before an application can access the database data, it needs to be connected to the database. You can make the connection using a wizard. The wizard will do most of the job for us.

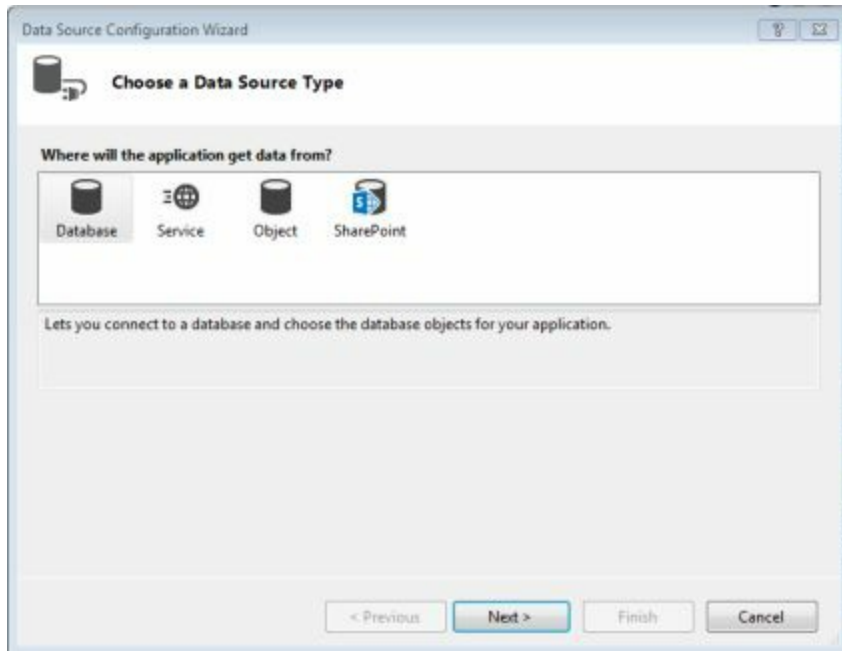
Let start with our simple database project. To connect the application to the *Salesdb* database, follow these steps:

In the *Solution Explorer*, click on *Data Sources* at the bottom:



Click **Add New Data Source** in the Data Source Configuration Windows to start the wizard.





Click the *Next* button to display the *Choose a Database Model* screen. Now, click on *Dataset*. Click *Next*.

Click *New Connection...*



Because we want to connect to an SQL Server 2014 database, make sure to select *Microsoft SQL Server Data Source*.

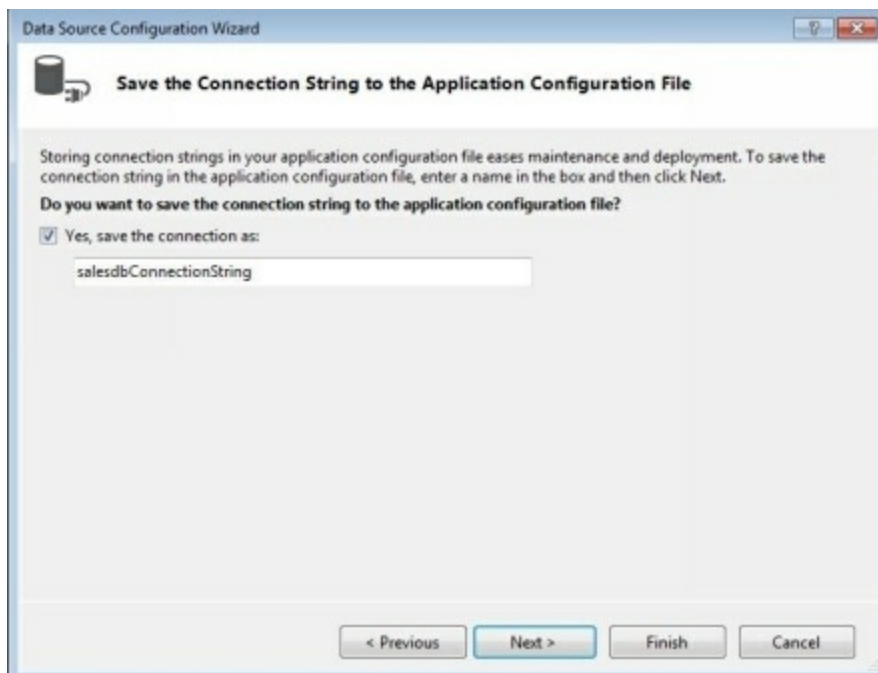


The 'Add Connection' dialog box is shown. It contains the following fields and controls:

- Data source:** A dropdown menu set to 'Microsoft SQL Server (SqlClient)' with a 'Change...' button to its right.
- Server name:** A text input field with a 'Refresh' button to its right.
- Log on to the server:** A section containing:
  - Authentication:** A dropdown menu set to 'Windows Authentication'.
  - User name:** A text input field.
  - Password:** A text input field.
  - ☐ Save my password
- Connect to a database:** A section containing:
  - ☒ Select or enter a database name: A dropdown menu.
  - ☐ Attach a database file: A text input field with a 'Browse...' button to its right.
  - Logical name:** A text input field.
- Buttons:** 'Test Connection', 'OK', 'Cancel', and 'Advanced...'.

Click **Test Connection** to test the connection.

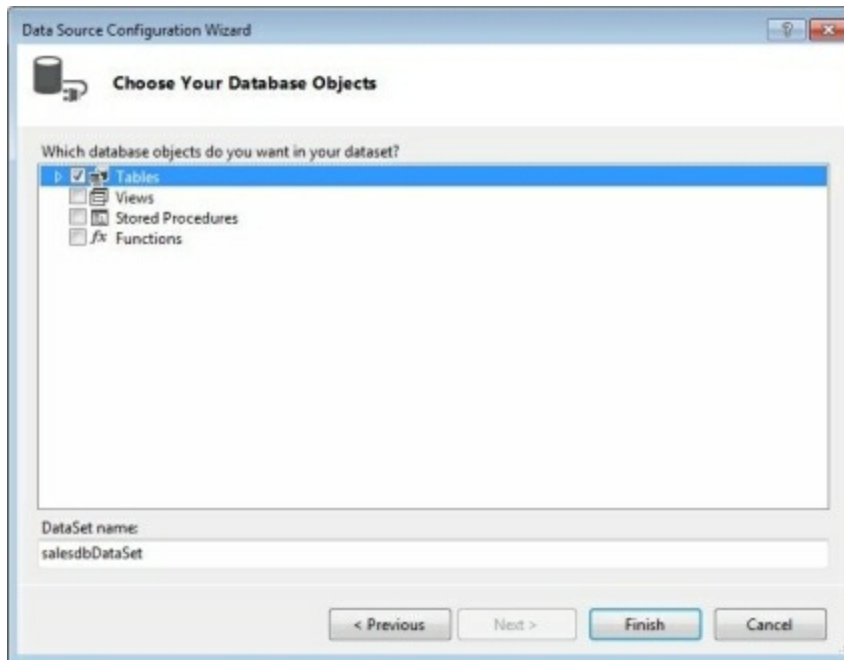
Next, click **Next** to save the connection string to the application configuration file:



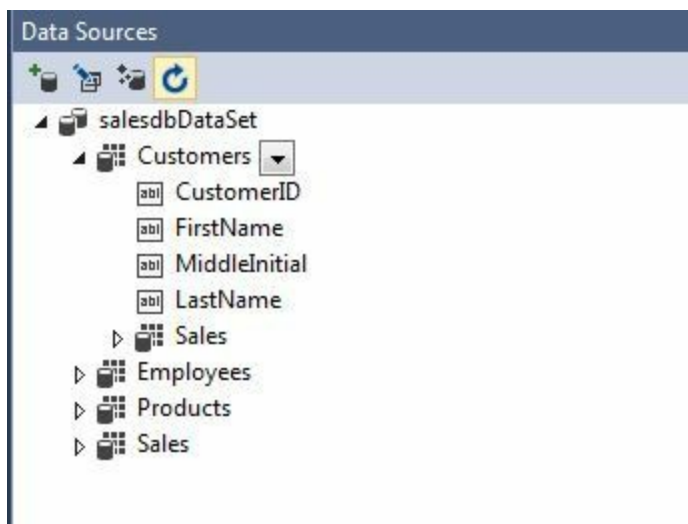
The 'Data Source Configuration Wizard' is shown, specifically the 'Save the Connection String to the Application Configuration File' step. It contains the following elements:

- Icon:** A database cylinder icon.
- Title:** 'Save the Connection String to the Application Configuration File'.
- Text:** 'Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.'
- Question:** 'Do you want to save the connection string to the application configuration file?'
- Options:** ☒ Yes, save the connection as:
- Text Input:** A text box containing 'salesdbConnectionString'.
- Buttons:** '< Previous', 'Next >', 'Finish', and 'Cancel'.

Now, you can select which tables and fields you want. Check the Tables box to include them all. You can give your dataset a name. Click *Finish* and you are done.



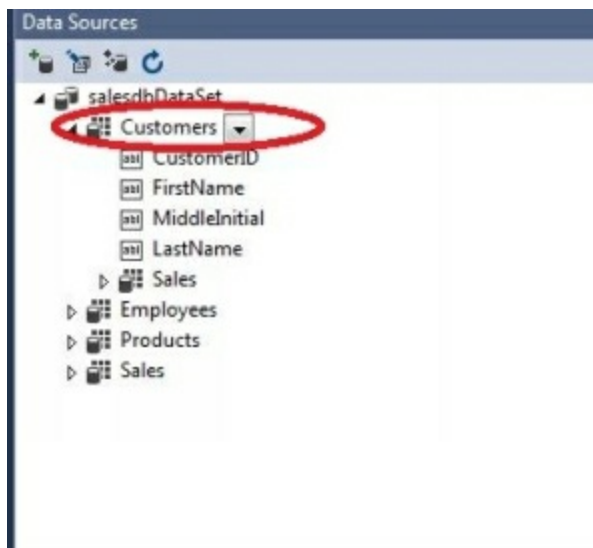
The Data Source of the Solution Explorer now display information about your database.



All the tables and the fields in the database should now be displayed.

## Previewing the Contents of a Dataset

To add a field or a table to your Form, click on the one in the list, hold down your left mouse button and drag it over to your form:



The table and all of its fields should be dragged to the Form. You should get something like this:

|   | CustomerID | FirstName | MiddleInitial | LastName |
|---|------------|-----------|---------------|----------|
| * |            |           |               |          |

As I already mentioned before, the wizard did most of the work for us, even the navigation control.

Press F5 key on your keyboard and see what will happen. Your form might look like this one:

|   | CustomerID | FirstName | MiddleInitial | LastName  |
|---|------------|-----------|---------------|-----------|
| ▶ | 1          | Aaron     |               | Alexander |
|   | 2          | Aaron     |               | Bryant    |
|   | 3          | Aaron     |               | Butler    |
|   | 4          | Aaron     |               | Chen      |
|   | 5          | Aaron     |               | Coleman   |
|   | 6          | Aaron     |               | Con       |
|   | 7          | Aaron     |               | Edwards   |
|   | 8          | Aaron     |               | Flores    |

Click the navigation buttons to scroll through the database.

# **Chapter 14 - Network programming**

## **IN THIS CHAPTER**

**Explaining sockets**

**Writing our own TCP/IP server and client**

**Programming FTP, HTTP, and SMTP clients**

In this chapter I will show you how to network applications with Visual Basic 2015. A network program is an application that uses a computer network to send data to another application on a remote computer. The Microsoft .NET Framework provides a layered, extensible, and managed implementation of Internet services that can be quickly and easily integrated into your applications.

## Socket programming

Sockets are defined as communication end-points, and relate to a means of computer communication using IP addresses and ports. Sockets are generally used to send data between two computers on a network, but can also be used for communication between two programs on the same computer.

As you probably know, an IP address defines both the network and the computer identity. The first part of the IP relates to the network, the second part the computer. Just where the first part ends and the second part starts is determined by the subnet mask.

Each computer also has a number of ports. These are not physical ports (e.g. like physical USB sockets) but simple numbers within the 64k range (0 to 65536). So it is possible for a computer to talk to a number of other devices at the same time by using dedicated ports for each conversation.

There are two types of communication protocol used for Socket Programming: **TCP/IP** (Transmission Control Protocol/Internet protocol) communication and **UDP/IP** (User Datagram Protocol/Internet protocol) communication .

In the following tutorial we are going to write a Server Socket Program and Client Socket Program through Visual Basic 2015 using TCP/IP Communication.

# Writing a simple TCP/IP server

For this example we are going to write a simple server program that waits and listens for the connection. The purpose of the TCP server program is to detect incoming data sent from the client. Any new data will be displayed in a textbox. The program will listen on port 12345.

To get started, create a new Visual Basic 2015 project. Set the project name to *myserver*.

Place the following controls on the Form1:

**(Button1):**

Name: btnListen

Text: Listen

**(Button2):**

Name: btnSend

Text: Send

**(TextBox1):**

Name: TextBox1

Text: leave the textbox blank

**(TextBox2):**

Name: TextBox2

Text: leave the textbox blank

**(Label1):**

Text:Receive

**(Label2):**



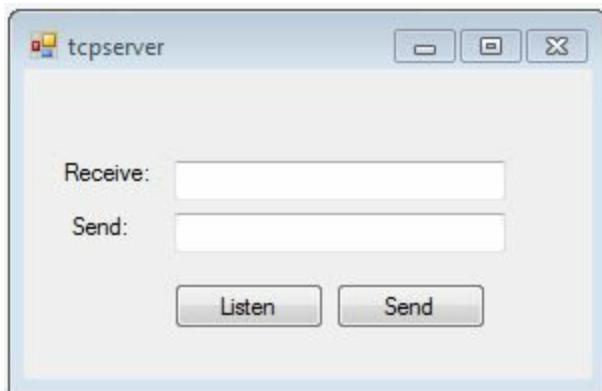
Text:Send

**(Timer1):**

Enabled:False

Interval:100

Your Form should look like this:



Add the following *Namespaces* at the top of the code:

Imports System.Net

Imports System.Net.Sockets

Now we need to declare two global variables. To do so, just bellow *Public Class Form1*, type the following:

Dim TCPServer As Socket

Dim TCPListener As TcpListener

## NOTE

Now we need to start the listening process. This opens the specified socket to receive any packets of data sent to it. The following code example creates a *TcpListener*. In the load event of the Listen button paste the following code:

```
TCPListenerz = New TcpListener(IPAddress.Any, 1234)
```

```
TCPListenerz.Start()
```

```
TCPServer = TCPListenerz.AcceptSocket()
```

```
TCPServer.Blocking = False
```

```
Timer1.Enabled = True
```

The listener would be useless without a timer. We need the timer to constantly update the listener to receive data:

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles
```

```
Timer1.Tick
```

```
Try
```

```
Dim rcvbytes(TCPServer.ReceiveBufferSize) As Byte
```

```
TCPServer.Receive(rcvbytes)
```

```
TextBox1.Text = System.Text.Encoding.ASCII.GetString(rcvbytes)
```

```
Catch ex As Exception
```

```
End Try
```

```
End Sub
```

Now, to send a message to a particular client, paste the following code inside the *Send* button:

Private Sub btnSend\_Click(sender As Object, e As EventArgs) Handles btnSend.Click

Dim sendbytes() As Byte =

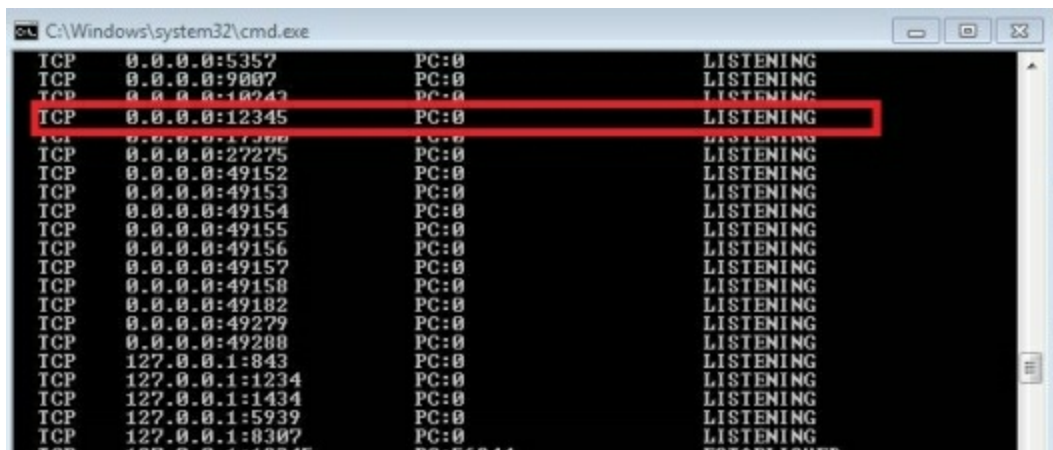
System.Text.Encoding.ASCII.GetBytes(TextBox2.Text)

TCPServer.Send(sendbytes)

End Sub

Run the program and click on the *Listen* button. You can test that the program is working using the following command in the Windows Command Prompt:

netstat -at



As you can see, the port 12345 is in the *LISTEN* state.

In the following chapter we are going to create a client socket program to

connect to our server.

## Writing a simple TCP/IP client

The TCP/IP Client will be a Windows based application with its main function to send message to Server application. To get started, create a new Visual Basic 2015 project and name it *myclient*.

Place the following controls on Form1:

**(Button1):**

Name: btnConnect

Text: Connect

**(Button2):**

Name: btnSend

Text: Send

**(TextBox1):**

Name: TextBox1

Text: leave the textbox blank

**(TextBox2):**

Name: TextBox2

Text: leave the textbox blank

**(TextBox3):**

Name: TextBox3

Text: leave the textbox blank

**(Label1):**

Name: Label1

Text: Server IP

**(Label2):**

Name: Label2

Text: Send

**(Label3):**

Name: Label3

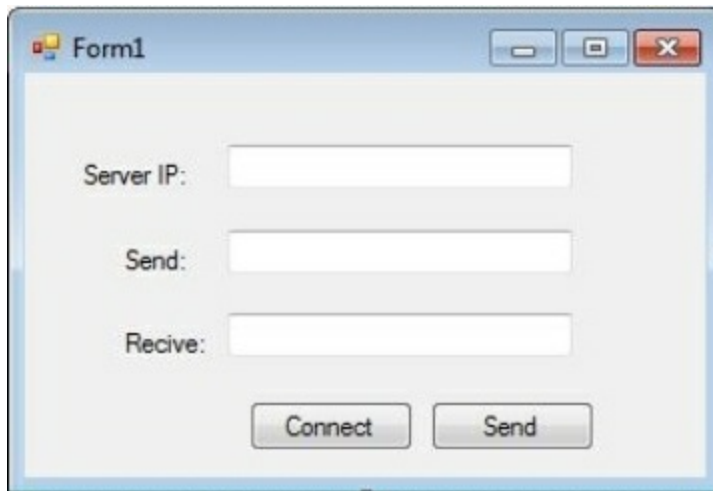
Text: Recive

**(Timer1):**

Interval: 100

Enabled: False

When you've finished setting properties, your Form should look something like this:

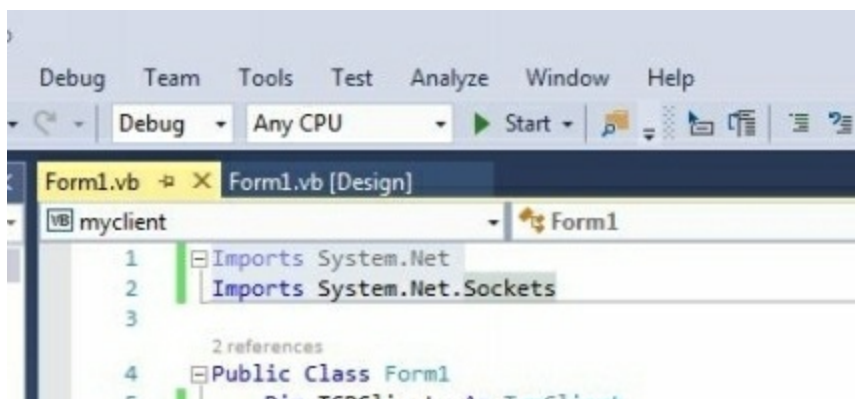


In this example we are going to use the *TcpClient* class, so we need to include the following statement in the program code:

Imports System.Net

Imports System.Net.Sockets

The statements has to be placed on top of the program, above the Public Class Form1:



The word *Imports* indicates that we are importing the namespace *System.Net* and the namespace *System.Net.Sockets* inside the application.

Next we need to declare two global variables. To do so, just bellow Public Class Form1 type the following:

```
Dim TCPClientz As TcpClient
```

```
Dim TCPClientStream As NetworkStream
```

To initiate the connection with server side application, place the following code inside the *btnConnect* click event:

```
TCPClientz = New TcpClient(TextBox1.Text, 12345)
```

```
Timer1.Enabled = True
```

```
TCPClientStream = TCPClientz.GetStream()
```

From the code above we can see that the first step is to create a TCP Client object. Here, the port 12345 is used simply because it is easy to remember and it is not in the first 1024 port numbers which are reserved for special use by IANA. To send and receive data, the *GetStream* method is used to obtain a *NetworkStream*.

The program would be useless without a timer. We need the timer to constantly update the listener to receive data:

```
Private Sub Timer1_Tick(sender As Object, e As EventArgs) Handles  
Timer1.Tick
```

```
If TCPClientStream.DataAvailable = True Then
```

```
Dim rcvbytes(TCPClientz.ReceiveBufferSize) As Byte
```

```
TCPClientStream.Read(rcvbytes, 0, CInt(TCPClientz.ReceiveBufferSize))
```

```
TextBox3.Text = System.Text.Encoding.ASCII.GetString(rcvbytes)
```

```
End If
```

```
End Sub
```

In the code above we are using *Read* method of the *NetworkStream* class to

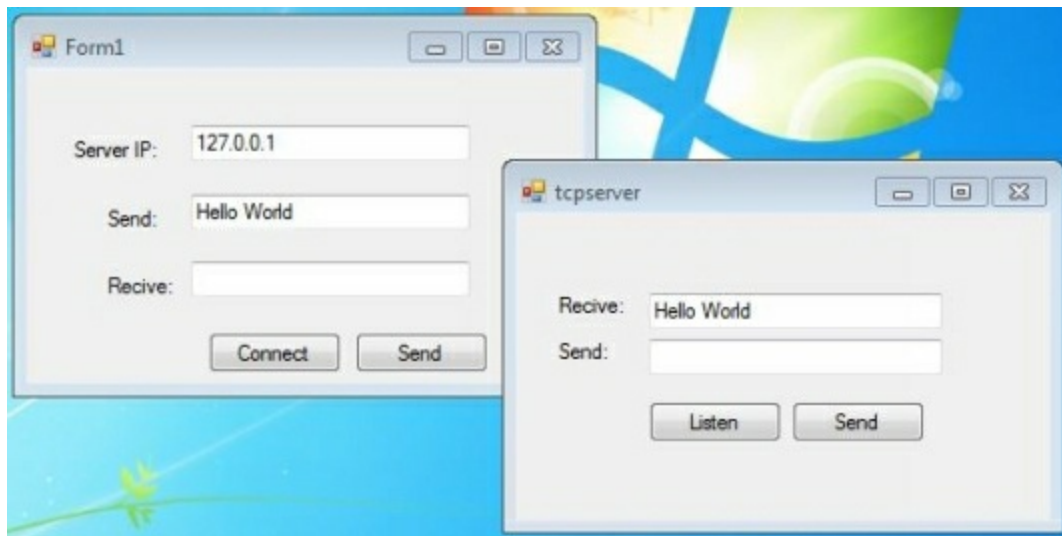
receive data from the remote host.

To send a message to the particular server application, place the following code inside the *Send* button:

```
Dim sendbytes() As Byte =  
System.Text.Encoding.ASCII.GetBytes(TextBox2.Text)  
TCPClientz.Client.Send(sendbytes)
```

To test the application, execute it from Visual Studio. On the same PC, open the TCP server application and execute it. Type *127.0.0.1* in the textbox and press the *Connect* button.

Next, try to input *Hello world* inside the *TextBox2* and click the send button on the client. The server should receive and display the *Hello World* message.



Congratulations! You have successfully used Visual Basic 2015 to send data across a network.





# HTTP protocol

HTTP (Hypertext Transfer Protocol) is an client-server protocol that allows clients to request web pages from web servers. It is an application level protocol widely used on the Internet. Clients are usually web browsers. When a user wants to access a web page, a browser sends an HTTP Request message to the web server. The server responds with the requested web page. Web servers usually use TCP port 80.

Clients and web servers use the request-response method to communicate with each other, with clients sending the HTTP Requests and servers responding with the HTTP Responses. Clients usually send their requests using GET or POST methods, for example *GET /homepage.html*. Web servers responds with a status message (200 if the request was successful) and sends the requested resource.

In this section we will learn how to create an simple HTTP Client for downloading file using the Visual Basic programming language.

So, let us start our new example by adding one button and two textboxes on our form.

The properties set for these controls are as follows:

## **(Button)**

Name: btnDownload

Text: Download file

## **(TextBox1)**

Name: TextBox1

Text:

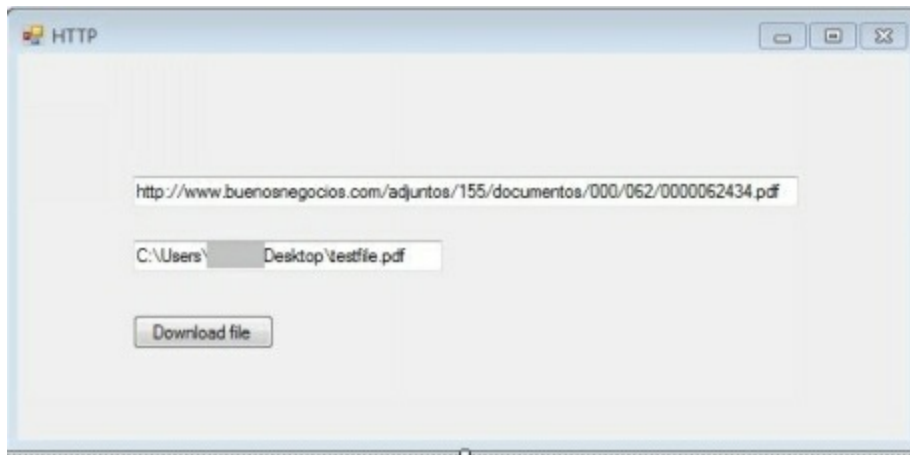
<http://www.buenosnegocios.com/adjuntos/155/documentos/000/062/00000624>

## **(TextBox2)**

Name: TextBox1

Text: *your file location*

The form should look like this:



We are going to use the *WebClient* class so we need to add the following Namespace at the top of the code:

Imports System.Net

The *WebClient* class provides common methods for sending and receiving data from a resource identified by a URL.

First we need to define a *WebClient* object. This client allows you to connect with HTTP servers like Apache. Double click on the *Download file* button and add the following code:

Try

```
Dim sURL As String = TextBox1.Text
Dim sFile As String = TextBox2.Text
Dim wc As WebClient = New WebClient()
wc.DownloadFile(sURL, sFile)
MessageBox.Show("Download has been completed")
```

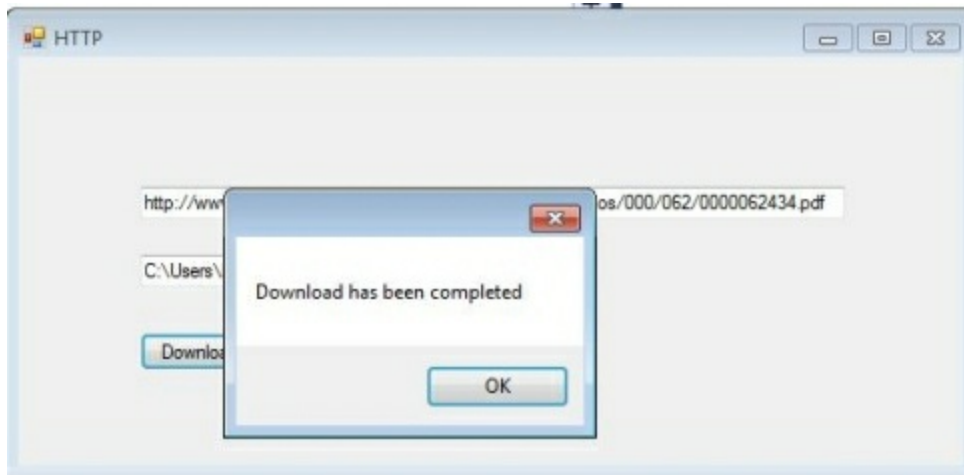
Catch ex As Exception

```
MessageBox.Show("The file does not exist. Please try again")
```

End Try

We are using the *DownloadFile* method to download a remote file and store it to a specific location.

Now, press F5 and test your application:



If everything went alright, you should get a message like the one above.

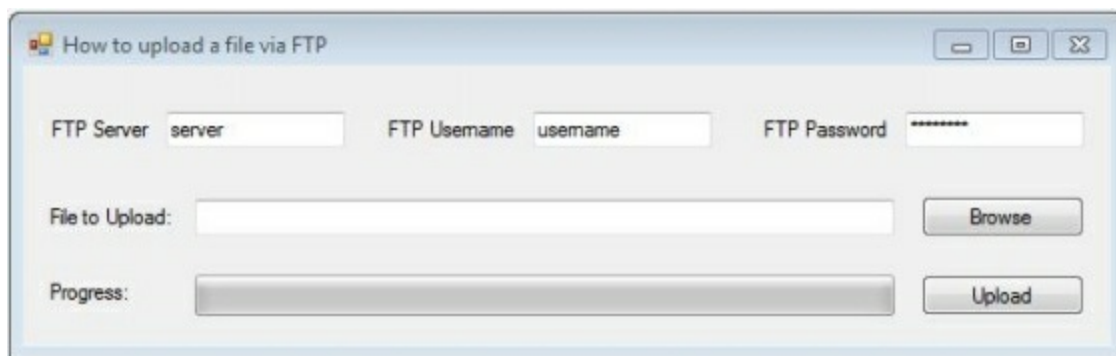
## FTP protocol

FTP (File Transfer Protocol) is a network protocol used to transfer files from one computer to another over a TCP network. Like Telnet, it uses a client-network architecture, which means that a user has to have an FTP client installed to access an FTP server running on the remote machine. FTP software is freely available for all major operating systems, including Windows, UNIX, and MAC OS X. After establishing an FTP connection, the user can download or upload files to and from the FTP server.

FTP uses two TCP ports: port 20 for sending data and port 21 for sending control commands. FTP can use authentication, but like Telnet, all data is sent in clear text, including usernames and passwords.

In this article you will learn how to create an simple FTP Client using Visual Basic.

To begin, open a new project, and then create the interface similar to the below:



Now change the object's properties as follow:

**(Button1):**

Name: btnBrowse

Text: Browse

**(Button2):**

Name: btnUpload

Text: Upload

**(TextBox1):**

Name: txtServer

Text: server

**(TextBox2):**

Name: txtUsername

Text: username

**(TextBox3):**

Name: txtPassword

PasswordChar:\*

**(TextBox4):**

Name: txtFile

**(Label1):**

Name: FTP Server

**(Label2):**

Name: FTP Username

**(Label3):**

Name: FTP Password

**(Label4):**

Name: File to Upload

**(Label5):**

Name: Progress

**(ProgressBar):**

Name: pb1

**(BackgroundWorker):**

Name: bWorker

In this example we are going to use the *FtpWebRequest* class, so first we need to add the following Namespace at the top of the code:

```
Imports System.Net
```

Next we need to define two global variables:

```
Dim ftpFilePath As String = Nothing
```

```
Dim fileUpload() As Byte
```

For the Browse button's event, write the following code:

```
Dim newFile As New OpenFileDialog
```

```
If newFile.ShowDialog = DialogResult.OK Then
```

```
txtFile.Text = newFile.FileName
```

```
ftpFilePath = txtServer.Text & "/" & IO.Path.GetFileName(txtFile.Text)
```

```
End If
```

In the code above the *ftpFilePath* global variable contains a remote directory where the file will be uploaded.

Next, inside the *bWorker* click event, add the following code:

```
Dim request As System.Net.FtpWebRequest =  
DirectCast(System.Net.WebRequest.Create(ftpFilePath),  
System.Net.FtpWebRequest)
```

```
Dim user As String = txtUsername.Text
```

```
Dim pass As String = txtPassword.Text
```

```
request.Credentials = New System.Net.NetworkCredential(user, pass)
```

```
request.Method = System.Net.WebRequestMethods.Ftp.UploadFile
```

```
Dim file() As Byte = System.IO.File.ReadAllBytes(txtFile.Text)
```

```
Dim strz As System.IO.Stream = request.GetRequestStream()
```

```
For offset As Integer = 0 To file.Length Step 1024
```

```
    bWorker.ReportProgress(CType(offset / file.Length * pb1.Maximum,  
Integer))
```

```
    Dim chSize As Integer = file.Length - offset
```

```
    If chSize > 1024 Then chSize = 1024
```

```
    strz.Write(file, offset, chSize)
```

```
Next
```

```
strz.Write(file, 0, file.Length)
```

```
strz.Close()
```

```
strz.Dispose()
```



The code above first attempts to log in using the username and password supplied. Then the program opens a TCP connection to the FTP server on port 21. Once the connection has been made, a stream to the remote host should be established.

Stream to the remote host is initialized through the *btnUpload* click event:

```
bWorker.RunWorkerAsync()
```

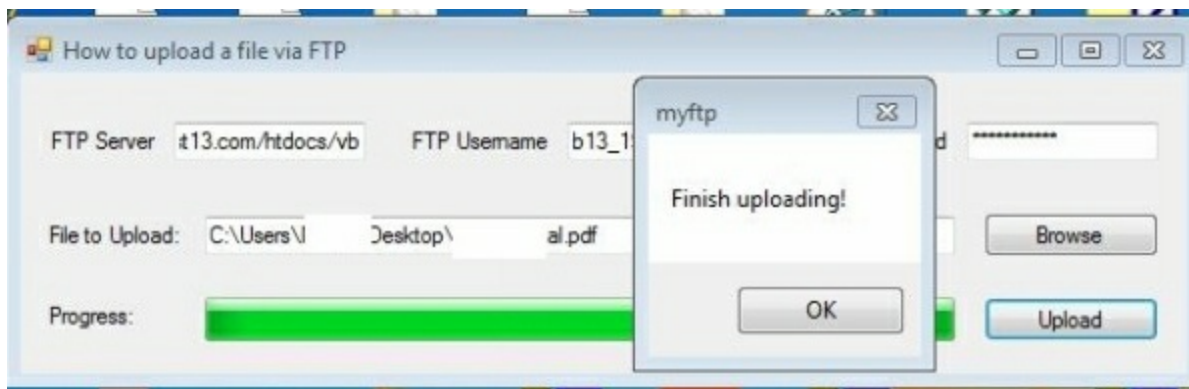
BackgroundWorker performs a task in the background. In the *ProgressChanged* event handler, add the code to indicate the progress - in this case updating the progress bar percentage:

```
Private Sub bWorker_ProgressChanged(sender As Object, e As  
ProgressChangedEventArgs) Handles bWorker.ProgressChanged
```

```
    pb1.Value = e.ProgressPercentage
```

```
End Sub
```

To test this application, ensure that you have an FTP server running and then execute the application from Visual Studio:



If everything went alright, you should be able to upload the file. After the process is completed, you should get the message like the one above.

## SMTP protocol

Simple Mail Transfer Protocol (SMTP) is the standard protocol for email services on a TCP/IP network. SMTP provides the ability to send and receive email messages.

SMTP is an application-layer protocol that enables the transmission and delivery of email over the Internet. SMTP is created and maintained by the Internet Engineering Task Force (IETF).

#### NOTE

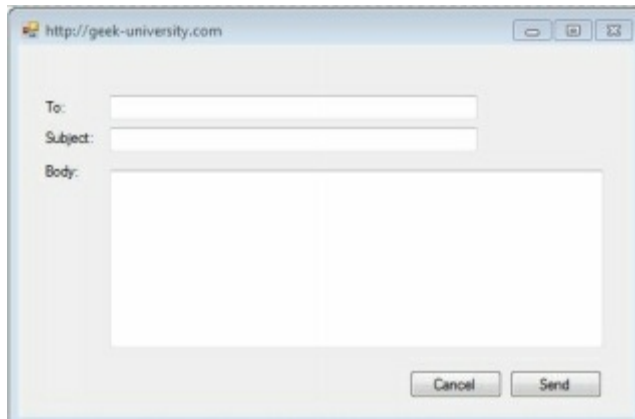
Simple Mail Transfer Protocol is also known as RFC 821 and RFC 2821.

In this section, I will show you how to send email using Gmail account in Visual Basic 2015.

The SMTP server address is *smtp.gmail.com*. It requires implicit SSL or explicit SSL (TLS) connection, and you should use your Gmail email address as the user name for authentication.

| Server         | Port    | SSL/TLS |
|----------------|---------|---------|
| smtp.gmail.com | 25, 587 | TLS     |
| smtp.gmail.com | 465     | SSL     |

To begin, open a new project, and then create the interface similar to the below:



In order to use the SMTP class, we need to import the following namespace:

Imports System.Net.Mail

Now change the object's properties as follow:

**(Button1):**

Name: btnCancel

Text: Cancel

**(Button2):**

Name: btnSend

Text: Send

**(TextBox1):**

Name: txtTo

Text: To

**(TextBox2):**

Name: txtSubject

Text: Subject

**(TextBox3):**

Name: txtBody

Text: Body

**(Label1):**

Text: To

**(Label2):**

Text: Subject

**(Label3):**

Text: Body

When you've finished setting properties, paste the following code inside the send button click event:

```
Dim MyMailMessage As New MailMessage()  
Try  
    MyMailMessage.From = New MailAddress("youremail")  
    MyMailMessage.To.Add(TextBox1.Text)  
    MyMailMessage.Subject = TextBox2.Text  
    MyMailMessage.Body = TextBox3.Text  
    Dim SMTP As New SmtpClient("smtp.gmail.com")  
    SMTP.Port = 587
```

```
SMTP.EnableSsl = True
SMTP.Credentials = New System.Net.NetworkCredential("youremail",
"yourpassword")
SMTP.Send(MyMailMessage)
MsgBox("Your Mail has been Successfully sent")
TextBox1.Clear()
TextBox2.Clear()
TextBox3.Clear()
TextBox1.Focus()
Catch ex As Exception
MsgBox(ex.ToString)
End Try
```

Change *youremail* with your current email address and *yourpassword* with your email ID's password.

And that should be it! Run your program and try to send a test email.

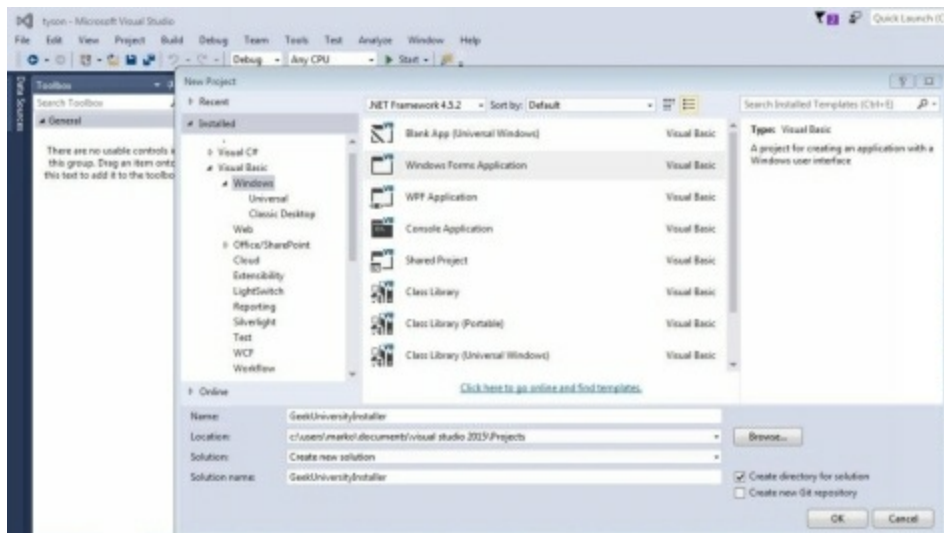
And with this, we've reached the final section of this book. There is an awful lot more to learn about Visual Basic 2015 and a bit of experimentation is needed before you become a skilled programmer.

# Appendices

## Appendix I: Deploying a Windows-based Application

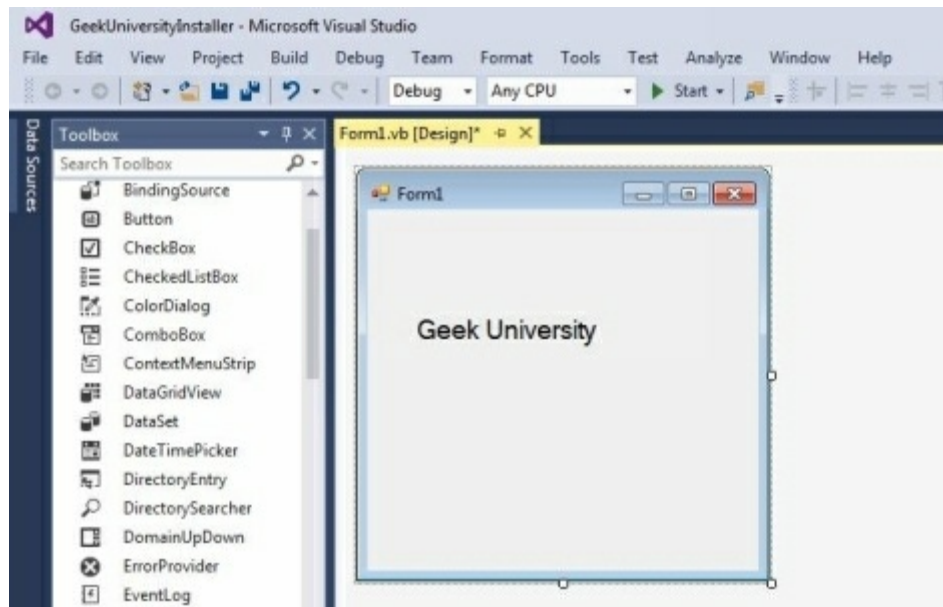
In this bonus chapter I will guide you through a step-by-step procedure for creating a Windows application and a setup installer for it in a way that is easy to understand and follow.

From the File menu select *New Project*. The following dialog box should appear:

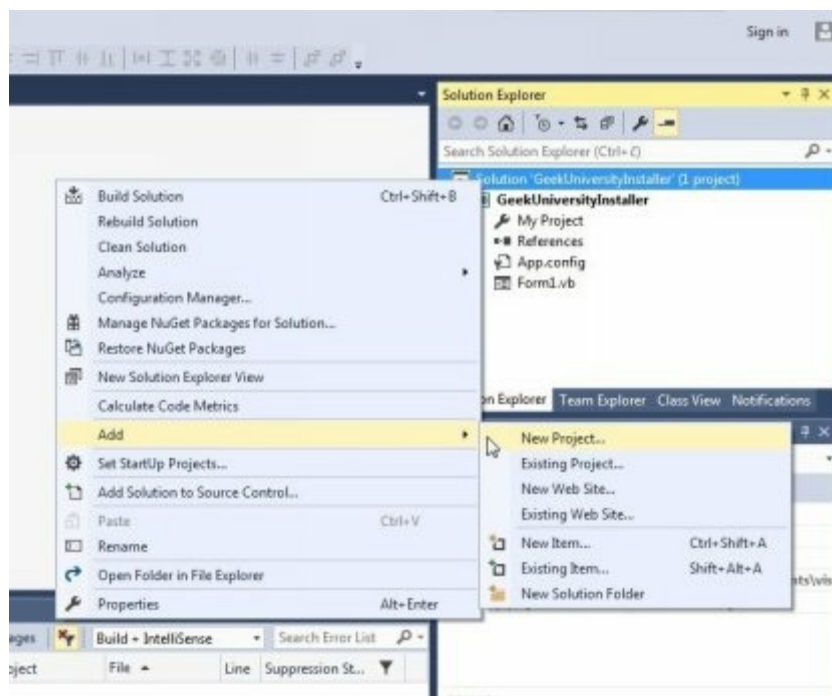


Select *Windows Form Application* and change the default project name to *GeekUniversityInstaller*.

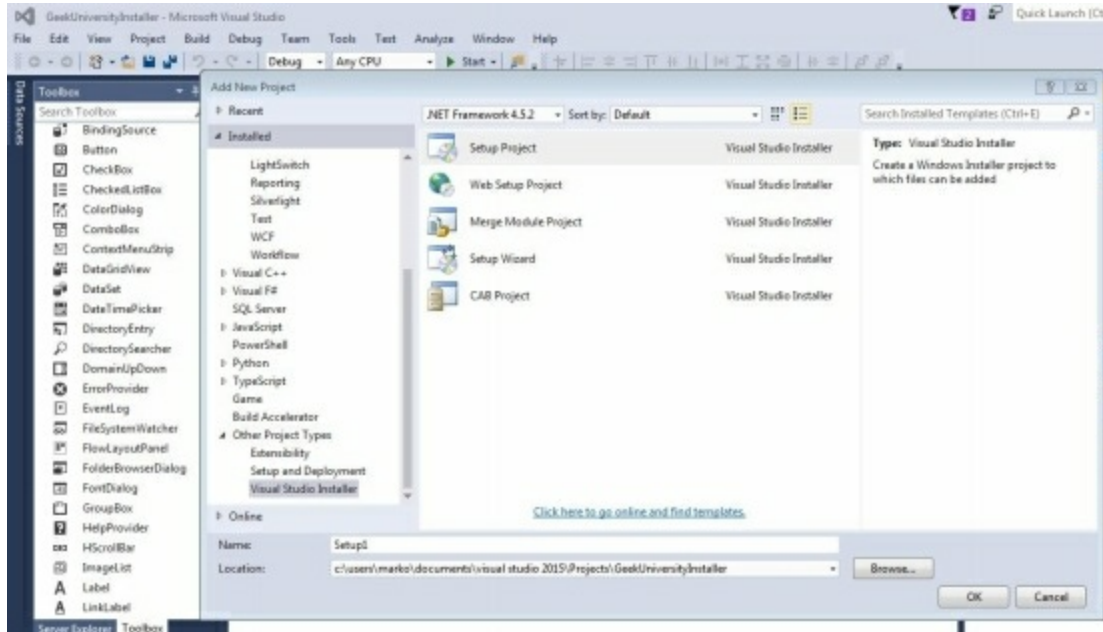
Now, add one label to the project form:



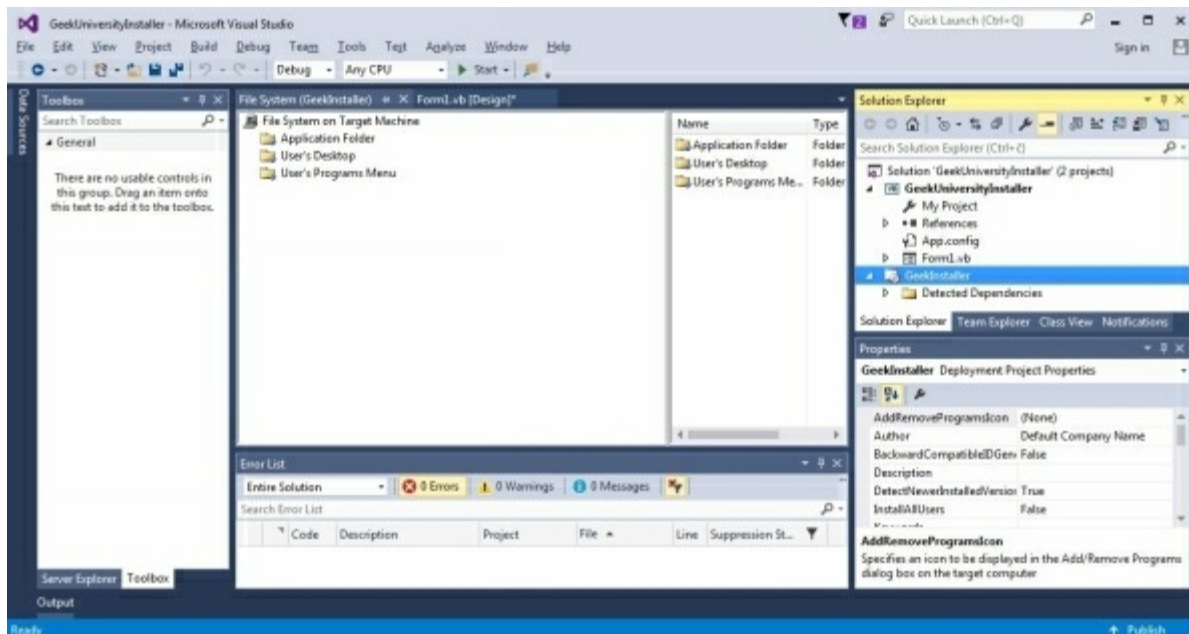
Now we will create an installer for the GeekUniversityInstaller application. Right-click on the solution explorer and add a new project to your solution like in the following picture:



Select a Setup Project by choosing *Other Project Types -> Setup and Deployment -> Visual Studio Installer*:



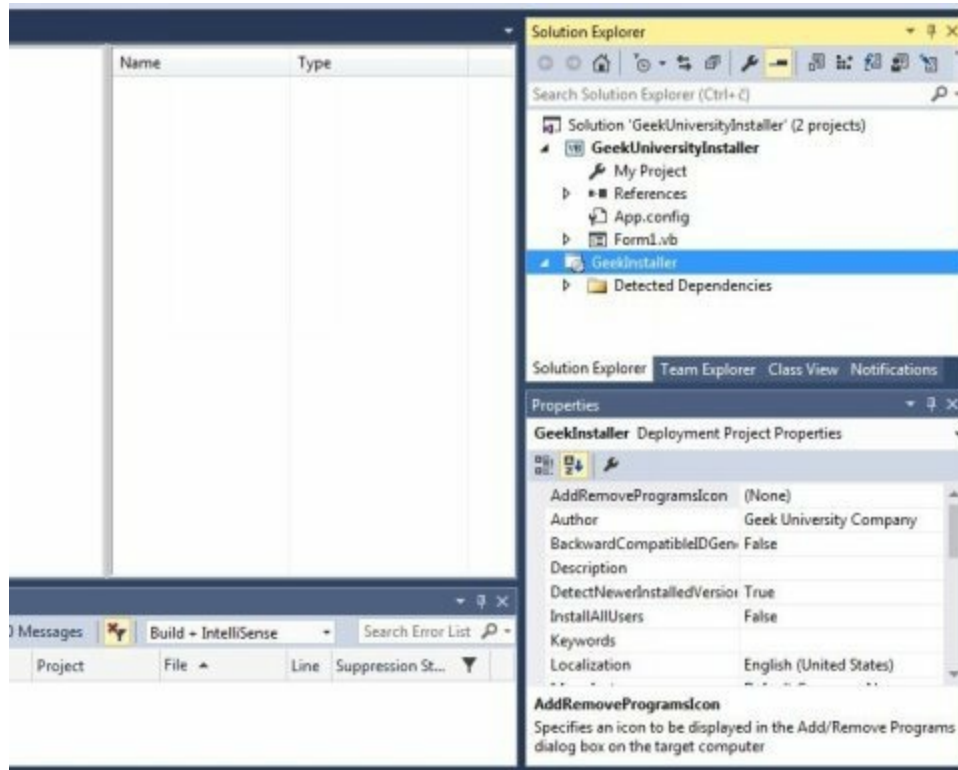
If everything went well you should see this screen:



You can change the properties of the object as you wish.



For example, you can change default Author property to *Geek University Company*:



Finally, save all and rebuild the project.

Now our setup is ready to install our Windows application. Just browse the debug folder location of the GeekUniversityInstaller project. Inside the folder you should find two files: *GeekInstaller.msi* and *Setup.exe*.

DoubleClick on the *GeekInstaller.msi* to start the installer. This installer will guide you through the process of installation:



That's it. You have just created your first simple installer.

# Appendix II: Cheat Sheet

## PROGRAM STRUCTURE

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click  
        MessageBox.Show("Hello World")  
    End Sub
```

```
End Class
```

## COMMENTS

```
Public Class Form1
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click  
        MessageBox.Show("Working with comments") ' This is a comment  
    End Sub
```

```
End Class
```

## RULES FOR NAMING VARIABLES, CONSTANTS, PROCEDURES AND ARGUMENTS

1. The name must start with a letter or an underscore.
2. The name can contain only letters, numbers, and the underscore character. No punctuation characters, special characters, or spaces are allowed in the name.
3. Name can't exceed 255 characters in length..
4. The name cannot be a reserved word, such as Sub or Double.

## TYPE CONVERSION RULES

1. Strings will not be implicitly converted to numbers.
2. Numbers will not be implicitly converted to strings.

3. Wider data types will not be implicitly demoted to narrower data types.
4. Narrower data types will be implicitly promoted to wider data types.

| DATA TYPES |                               |
|------------|-------------------------------|
| Boolean    | a logical value (True, False) |
| Char       | one Unicode character         |
| Date       | date and time information     |
| Decimal    | a number with a decimal place |
| Double     | a number with a decimal place |
| Integer    | integer                       |
| Long       | integer                       |
| Object     | data of any type              |
| Short      | integer                       |
| Single     | a number with a decimal place |
| String     | text                          |

## CONSTANTS

`Const dblPI As Double = 3.141593`

| OPERATORS               |  |
|-------------------------|--|
| <code>^</code>          | Exponentiation   |
| <code>-</code>          | negation   |
| <code>*,/</code>        | multiplication and division  |
| <code>\</code>          | integer division   |
| <code>Mod</code>        | modulus (remainder) arithmetic                                     |
| <code>+,-</code>        | addition and subtraction   |
| <code>&amp;</code>      | concatenation  |
| <code>=,&lt;&gt;</code> | equal to, not equal to   |
| <code>&gt;,&gt;=</code> | greater than, greater than or equal to                             |
| <code>&lt;,&lt;=</code> | less than, less than or equal to                                   |
| <code>Not</code>        | reverses the truth-value of the condition; True                    |
| <code>And</code>        | all subconditions must be true                                     |
| <code>AndAlso</code>    | same as the And operator, except performs short-circuit evaluation |
| <code>Or</code>         |  |

|        |   |
|--------|---|
| OrElse | only one of the subconditions needs to be true  |
| Xor    | same as the Or operator, except performs short-circuit evaluation                         |
|        | only one of the sub-conditions can be true for the compound condition to evaluate to True |

| ARITHMETIC ASSIGNMENT |                           |
|-----------------------|---------------------------|
| +=                    | addition assignment       |
| - =                   | subtraction assignment    |
| *=                    | multiplication assignment |
| /=                    | division assignment       |

## CHOICES

```

Dim x As Integer
x = InputBox("Enter a number:", "http://geek-university.com", "Type your number here.")
If x Mod 2 = 0 Then
    MessageBox.Show("The number you have entered is an even number")
Else
    MessageBox.Show("The number you have entered is an odd number.")
End If

```

```

Dim game As String
game = txtInfo.Text
Dim answer As String
answer = game & "is your favourite game!"
Select Case game
    Case "GTA 4"
        MessageBox.Show(answer)
    Case "Battlefield 3"
        MessageBox.Show(answer)
    Case "GRID"
        MessageBox.Show(answer)
    Case Else
        MessageBox.Show("Looks like your game is not on my list")
End Select

```

## LOOPS

```
Dim number As Integer
For number = 1 To 10
    MessageBox.Show(number.ToString)
Next number
```

```
Dim line As String
FileOpen(1, "C:\test.txt", OpenMode.Input)
Do While Not EOF(1)
    line = line & LineInput(1)
    richTextBox.Text = line
Loop
```

## ARRAYS

```
Dim salesalaries(5) As Integer
salesalaries(0) = 1000
salesalaries(1) = 1500
salesalaries(2) = 2000
salesalaries(3) = 2500
salesalaries(4) = 3000
```

```
richSalaries.Text = "Harry's salary is:" & salesalaries(0) & "$" & vbNewLine _
    & "Jackson's salary is:" & salesalaries(1) & "$" & vbNewLine _
    & "Liam's salary is:" & salesalaries(2) & "$" & vbNewLine _
    & "Ethan's salary is:" & salesalaries(3) & "$" & vbNewLine _
    & "Noah's salary is:" & salesalaries(4) & "$" & vbNewLine
```

## STRINGS

'Special character constants (all also accessible from ControlChars class)

```
vbCrLf, vbCr, vbLf, vbNewLine
vbNullString
vbTab
vbBack
vbFormFeed
vbVerticalTab
```

' No string literal operator

```
Dim filename As String = "c:\test.dat "
```

```
' String comparison
```

```
Dim name As String = "John"
```

```
If (name = "John") Then ' true
```

```
If (name.Equals("John")) Then ' true
```

```
If (name.ToUpper().Equals("JOHN")) Then ' true
```

```
If (name.CompareTo("John") = 0) Then ' true
```

```
' Substring
```

```
s = Mid("testing", 2, 3) ' est
```

```
' Replacement
```

```
s = name.Replace("ohn", "oan") ' s is "Joan"
```

```
' Split
```

```
Dim names As String = "John,Joan,Jim,Jack"
```

```
Dim parts() As String = names.Split(",".ToCharArray()) ' One name in each slot
```

```
' Integer to String
```

```
Dim x As Integer = 7
```

```
Dim y As String = x.ToString() ' y is "7"
```

```
' String to Upper
```

```
word = "HELLO WORLD"
```

```
MessageBox.Show(word.ToLower)
```

## EXCEPTION HANDLING

```
Try
```

```
Dim sr As StreamReader = File.OpenText("C:\\test.txt")
```

```
Console.WriteLine("The first line of this file is:", sr.ReadLine())
```

```
sr.Close()
```

```
Catch ex As Exception 'one or more statements to execute when an exception occurs
```

```
MessageBox.Show("Sorry, the file or folder does not exist")
```

```
End Try
```

## ACCESS FILES

```
'Write to file
```

```
Dim fileToWrite As String = "C:\test.txt"
```

```
If System.IO.File.Exists(fileToWrite) = True Then
```

```
    Dim objWriter As New System.IO.StreamWriter(fileToWrite)
```

```
    objWriter.Write(txtWrite.Text)
```

```
    objWriter.Close()
```

```
    MessageBox.Show("Text written to file")
```

```
Else
```

```
    MessageBox.Show("File Does Not Exist")
```

```
End If
```

```
'Read from file
```

```
Dim fileToOpen As String = "C:\test.txt"
```

```
Dim objReader As New System.IO.StreamReader(fileToOpen)
```

```
txtRead.Text = objReader.ReadToEnd
```

```
objReader.Close()
```

## FUNCTIONS

```
Public Class Form1
```

```
    Dim labeltext As String = "Working with functions in Visual Basic 2015"
```

```
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

```
    End Sub
```

```
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
```

```
        Label1.Text = changelabeltext()
```

```
    End Sub
```

```
    Public Function changelabeltext()
```

```
        Return labeltext
```

```
    End Function
```



End Class

## NAMESPACES

Namespace Harding

    Namespace Compsci

        Namespace Graphics

        ...

    End Namespace

End Namespace

End Namespace

Imports Harding.Compsci.Graphics

## DATABASES

Connect an application to an Access database

1. Open the application's solution file.
2. If necessary, open the Data Sources window by clicking View on the menu bar, pointing to Other Windows, and then clicking Data Sources.
3. Click Add New Data Source in the Data Sources window to start the Data Source Configuration Wizard, which displays the Choose a Data Source Type screen. If necessary, click Database.
4. Click the Next button, and then continue using the wizard to specify the data source

# Conclusion

If you are reading this conclusion and you have read carefully the entire book, then please accept my congratulations! I am certain that you have earned valuable knowledge in the principles of programming that will stick for life. Even when the years pass, even if technology evolves and computers are far from their current state, the fundamental knowledge of data structures in programming and the algorithmic way of thinking as well as the experience gained in solving programming problems will always aid you, especially if you work in the field of information technology.

Also, please check out the free courses available from Geek University (<http://geek-university.com>). These courses are an excellent follow-up to your progress as software engineers and professionals in software development. All materials (lecture slides, exercises, demos) at Geek University Web site are available in English for free!