Fundamentals of C# Programming for Information Systems

George C. Philip
The University of Wisconsin Oshkosh

Updates for Visual Studio 2019 Users

Including the Preface, Chapter 1 (Sections 1.1–1.3), and Chapter 15 (Section 15.1)

Contributions by Dr. Jakob Iversen
The University of Wisconsin Oshkosh

Prospect Press

Preface

Mission of the Text

Welcome to *Fundamentals of C# Programming for Information Systems*. This book teaches the fundamentals of programming in C# to provide a solid foundation to build business and other real-world applications. Programming concepts are discussed in the context of familiar practical applications that use graphical interfaces.

New to This Edition

A key goal of the new edition was to add significant content so that the book could be used in a two-course sequence in programming. Four new chapters have been added: Two chapters that provide a comprehensive introduction to web applications development, and two other chapters on the concepts and practice of object-oriented programming, including inheritance.

Several other updates were made based on feedback from instructors: a second set of three comprehensive assignments (in Chapters 3, 8, and 12), additional end-of-chapter exercises, and learning objectives for each chapter.

Target Audience

This book is designed for introductory programming courses in IS/MIS, CIS and IT. This book also would fit into a computer science curriculum with an introductory course that uses a GUI-based application-oriented approach to teach programming concepts. The breadth and depth of coverage make this book suitable for a two-course sequence, particularly when students come to the first course with no programming background and a slower pace is desired. An approach in a two-course sequence would be to do in-depth coverage of topics like collections, databases, object-oriented programming, web development, and others presented in later chapters only in the second course.

Key Features

A key feature of the book is that programming concepts are introduced in small chunks through examples and illustrations accompanied by hands-on tutorials. The tutorials, which are interspersed with the concepts, help students apply and explore what they learn immediately. Additionally, review questions and exercises within the chapters enhance student interest and learning.

Although the book is written for beginners, it is thorough and concise. Graphical illustrations and screenshots are used throughout the book to enhance learning for both beginners and experienced students.

Windows forms are used from the beginning to provide GUI-based as opposed to console-based interface. Graphical user interfaces and code are built in the .Net environment using Visual Studio.

Supplements

For Students: Tutorial_Starts.zip file that contains

- Partially completed projects for tutorials
- Data files/databases used in projects

You may download the Tutorial_Starts.zip file from
http://www.prospectpressvt.com/titles/c-sharp-programming/student-resources/

For Instructors: Instructor resources include

- Completed tutorials
- PowerPoint slides for all chapters
- Test bank
- Partially completed projects for tutorials
- Data files/databases used in projects

To access instructor resources, please complete the request form at
http://www.prospectpressvt.com/faculty-resources/instructors-material/

Installing Visual Studio

You are encouraged to use the free version, Visual Studio Community 2019 or Visual Studio Community 2017. The 2015 version generally works well, except that you won't be able to run a few programs that use certain features available only in C# version 7.1 or greater. The instructions given in this textbook on using Visual Studio are based on VS Community 2017. Sections 1.1–1.3 of Chapter 1 include instructions and windows that are updated for VS Community 2019.

You may download Visual Studio Community from the following website:

https://www.visualstudio.com/downloads/

To install Visual Studio, open the downloaded .exe file and run it.

Overview of the Content and Organization

Every possible sequence of topics seems to put constraints on the quality of illustrative examples and applications that can be used in a chapter. The organization of chapters in this book attempts to minimize such problems and to enhance the ability to build on prior chapters. However, except for the foundational Chapters 1–5, there is significant flexibility in choosing specific topics and the depth of coverage. As suggested by the dependencies summarized below, there is some flexibility in the sequencing too.

Chapters 1–5 cover the Visual Studio environment and introductory programming concepts, including methods. These chapters, which provide the foundational knowledge, should be covered in sequence before other chapters, though certain topics like working with dates and times (Section 2.4) may be postponed or skipped.

Chapter 6 presents the application of the GUI controls ScrollBars, RadioButtons, CheckBoxes, ComboBoxes and ListBoxes.

Chapter 7 provides a detailed presentation of one-dimensional arrays, and Chapter 8 presents accessing sequential files and using arrays in combination with files. Chapter 6 is not a prerequisite for Chapters 7 or 8. GUI is presented early on in Chapter 6 to motivate students with more interesting graphical interfaces. It should be noted that the comprehensive assignment (Assignment 2) specified at the end of Chapter 8 requires the use of several GUI controls.

Chapter 9 introduces collections, and discusses the List and Dictionary collections in more detail. Chapter 8 ("Sequential Files and Arrays") is a prerequisite for this chapter. Because of the close relationship between collections and arrays, collections are presented in this book immediately following Chapter 8 on arrays.

Chapter 10 discusses the application of ListView and TabControl. The dependency of this chapter on Chapter 9 (“Collections”) is very low. The prerequisite for this chapter includes Chapters 6 and 8.

Chapter 11 presents multiform applications, Menus and ToolStrips. This chapter has some dependency on previous chapters, except Chapter 10.

Chapter 12 provides in-depth coverage of accessing databases from C# programs. Chapter 6 is a prerequisite for this chapter. In addition, the ListView control presented in Chapter 10 is used in an example in the last part of this chapter, and it is required in the third comprehensive assignment (Assignment 3) at the end of this chapter. Assignment 3 also requires the use of MainMenu control discussed in Chapter 11. Other than that, the dependence of Chapter 12 on Chapters 7–11 is relatively low.

Chapter 13 provides an introduction to object-oriented programming (OOP) principles and techniques. The initial part of this chapter may be used for an introduction to OOP early in the semester.

Chapter 14 describes the concept of inheritance, implementation of inheritance, subclasses and super classes, overriding methods and polymorphism. Chapter 13 is a prerequisite for this chapter, and it also relies on collections from Chapter 9.

Chapter 15 presents the concepts of web applications development, and develops simple web applications using the ASP.Net platform in the Visual Studio environment. Only the basic programming knowledge presented in Chapters 1–3 is required for this chapter.

Chapter 16 describes how to access SQL Server databases and develop multipage web projects. Basic programming concepts presented in Chapters 1–6, the concept of collections from Chapter 9, and basic database concepts including binding controls to a database and filtering records (Chapter 12) are used in building the application.

Acknowledgments

I am thankful for the valuable assistance provided by many people in the preparation of this book. I wish to thank Dr. Jakob Iversen, The University of Wisconsin–Oshkosh, for authoring Chapters 13 and 14 on Object-Oriented Programming.

I was fortunate to work with Beth Lang Golub, editor and president of Prospect Press, who was flexible and supportive of my goal to offer a good quality programming textbook at a reasonable price. Special thanks go to Susan Hegedus, Kathy Bond Borie and Rachel Paul for their painstaking attention to detail in editing this book, and to Annie Clark for the cover design.

I wish to acknowledge the contributions of the following reviewers for their valuable guidance in improving the presentation and contents of this book:

Janet Bailey, University of Arkansas at Little Rock
Wei Kian Chen, Champlain College
Clinton Daniel, University of South Florida
Silvana Faja, University of Central Missouri
Joni L. Jones, University of South Florida
David Pumphrey, Colorado Mesa University
Manonita M. Ratwatte, University of Oklahoma (Retired)
Theadora Ross, University of Arkansas at Little Rock
David M. Weber, Northern Arizona University

Thanks are also due to the instructors who provided valuable feedback on the first edition of this book through user surveys:

Janet Bailey, University of Arkansas, Little Rock
Jeff Dickson, Oregon Institute of Technology
Ruth Lamprecht, Virginia Union University
Panos Linos, Butler University
Ziping Liu, Southeast Missouri State University
Robert Pilgrim, Murray State University
Kris Rosenberg, Oregon Institute of Technology
Doug Titus, University of North Florida

About the Author

Dr. George Philip is Professor Emeritus of Information Systems at the College of Business, The University of Wisconsin–Oshkosh. He has more than twenty-five years of teaching and consulting experience in the information systems field, including computer programming in multiple languages. He also served as chair of the Information Systems Team, and director of the M.S. in Information Systems program. He has published numerous articles in this field.

Chapter 1

Introduction to Visual Studio and Programming

Welcome to programming in the C# language. In this chapter, you will learn to develop simple programs in the Visual Studio development environment, and to work with different types of data.

Learning Objectives

After studying this chapter, you should be able to:

- Identify the inputs, processes and outputs of a software system.
- Describe the steps involved in developing a computer program.
- Describe the terms: syntax, logic and runtime errors, machine language, low-level language, high-level language, compiler and interpreter.
- Develop a simple form in Visual Studio to accept user input, do calculations and display formatted output.
- Work with Label, TextBox, Button, ListBox and MessageBox.
- Develop simple programs that use constants, variables and expressions.
- Use the try-catch method to catch errors.

Topics

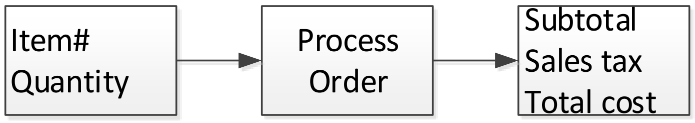
1.1 Introduction to Programming

Programming is the process of developing computer programs. If a computer program seems like a mystery to you, it is just a set of instructions telling the computer how to do a task, like looking up the price of an item or finding the Chinese restaurants in a city.

Unfortunately, computers cannot understand normal English. So, programs have to be written using special commands and statements according to strict rules. A key aspect of programming is breaking down what you want the computer to do, into detailed instructions. Like the directions that a GPS gives you to get to a place, the instructions in a program need to be precise.

Typically, a program uses one or more data items to produce some results. For example, a program that processes an order might use the item number and order quantity to compute the subtotal, sales tax and total cost, as represented in Figure 1-1.

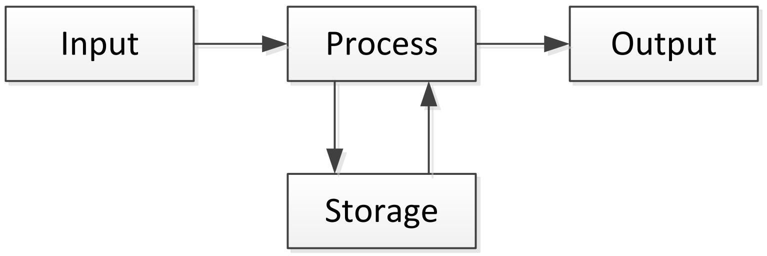
Figure 1-1: Inputs, process and outputs of a program



The program, represented by the block "Process Order," may include multiple subtasks like look up the unit price, check inventory and compute results.

The data that are used by a program are called the **input** to the program**,** and the results produced are called the **output** of the program. In addition to processing input data to produce the output, a program might write data to and read data from **storage** devices like a flash drive or a hard drive, as represented in Figure 1-2.

Figure 1-2: A general representation of a software system



How Do I Go About Developing a Program?

To understand the process of developing a program, let's use a simplified order-processing system as an example. To develop good programs and do so efficiently, follow these steps:

1. Define the purpose, and identify the input, process and output of the program.
2. Design and develop the graphical user interface (GUI).
3. Identify the components and logic of the program.
4. Design and develop files/databases, if any.
5. Write and test the code.

Let's look at these steps in more detail.

1. Define the purpose, and identify input/process/output

Before you can write the program, you need to lay some groundwork. In this step, you identify what the user wants the program to do, including the input, process and output of the program.

Depending on the size and complexity of the program, this could involve extensive analysis of the requirements, including interviews with users; examination of current forms, reports and transactions; and identifying processes like checking inventory and looking up price in an order.

Here is an example of a simplified statement of the **purpose** of the order processing program:

Purpose:	Compute and display the subtotal, sales tax and total cost for an order

The **output** of the program often follows from the purpose. For this example, the output would be

Output:	subtotal, sales tax, total cost

The **process** specifies not only **what** the program should do (e.g., compute total cost) but also **how** it should be done (e.g., how to compute total cost), as follows:

Process: (What?)	Look up unit price, look up sales tax rate, compute subtotal, sales tax and total cost	
(How?)	subtotal:	unit price * order quantity
	sales tax:	subtotal * sales tax rate
	total cost:	subtotal + sales tax

Specifying the process also would include identifying the sources of data, like the product file to get the unit price and sales tax file to get the sales tax rate.

The **input** specifies the data items that are needed to carry out the process to produce the output. The input for this order-processing system would be

Input: item number, order quantity

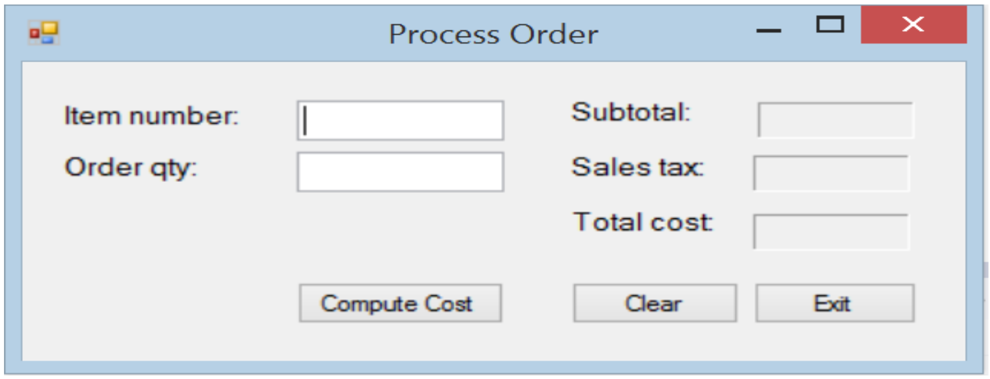
Note that unit price and sales tax rate are not included in the input because the program looks them up. A real-world system would be a lot more complex. Typically, the process would include additional subtasks like handling orders when inventory is insufficient, and output may include various reports. In such systems, graphical methods like **Data Flow Diagrams** and **UML diagrams** are used to represent the processes and the data accessed by them.

2. Design and develop the graphical user interface (GUI)

After identifying the input, process and output of a program, you design and create the user interface—that is, how the user would interact with the program, and how the program would communicate with the user. This is the fun part where you bring in your creativity.

Typically, you use forms to interact with the program. As you will learn in the next section, forms have various types of objects, called **controls**, such as Button, TextBox and Label. In this step, you identify the type of controls to be used, specify their names and captions as appropriate, and design the layout. Figure 1-3 shows an example of the GUI for a simplified order-processing system where the user doesn't provide the unit price and sales tax rate.

Figure 1-3: GUI for an order-processing system



3. Identify the components and logic of the program

This step identifies the major subtasks of the program. For example, in order to process an order, the program needs to do the following subtasks:

```
Get Item# and quantity
Look up unit price and tax rate
Compute subtotal, sales tax and total cost
Display subtotal, sales tax and total cost
```

Again, a real-world program may have to do additional subtasks, like checking the inventory to make sure there is sufficient quantity on stock.

For relatively simpler programs, after identifying the subtasks, you may go directly to writing the program for each subtask. However, for tasks involving more complex logic, it might help to develop an outline of the logic of performing the subtasks. The representation of the logic of a program in plain English is called **pseudo code**. You also may represent the logic graphically using a **flowchart**, as discussed in Chapter 3.

4. Design and develop files/databases

If data is read from or written to files and/or databases, these are designed and developed prior to writing the program. Depending on the application, this step may have to be done in parallel with previous steps.

5. Write and test the code

The final step is to write and test the code. You can program in a variety of languages. C#, Java, Visual Basic, Python and PHP are among the popular languages. You will use C#, which is a popular language for developing desktop and web applications.

Programing may involve iteratively developing an application by going through the above steps multiple times.

Syntax, Logic and Runtime Errors

The programming statements you write have to follow strict rules of the language, called the **syntax**. The program wouldn't **compile** if it had any **syntax error**, like a missing semicolon at the end of a statement or a misspelled key word. Compiling is the process of translating the program you write, called the **source code**, to another language before running a program, as described in the next section. So, your first task is to make sure that there are no syntax errors. The good news is that Visual Studio provides a lot of help in identifying syntax errors.

After the syntax errors are eliminated, the program may run. But, it's still too early to celebrate because the results could be incorrect due to errors in the program logic, just like you can write a grammatically correct sentence that doesn't convey the intended message. Errors that cause a program to produce incorrect or unintended results are called **logic errors**. A tax-filing software using the wrong tax rate and a billing software overcharging a customer are examples of logic errors.

There are errors other than logic errors that can occur at runtime. These are called **runtime errors**. Runtime errors cause the program to crash (unless the program catches and handles such errors) because the program asks the computer to do something it is unable to do, like accessing a file with an invalid path or dividing a number by zero.

The process of identifying errors (bugs) is called **debugging**. Testing programs to identify and eliminate errors is an extremely important part of developing software.

Review Questions

1.1 Consider Google as a software system. What would be the input, process and output for Google?

1.2 Consider a software system that enrolls students into classes. Identify some key inputs that the system needs every time a student enrolls in a class, and the subtasks (process) that need to be performed. What are some outputs the system should produce for students and instructors?

1.3 List the major steps in developing a program.

1.4 Incorrect punctuation in a program is an example of what type of error?

1.5 A payroll program uses the wrong formula to compute overtime pay. What type of error is it?

1.6 True or false: A program that doesn't have any syntax errors should produce the correct results.

1.2 Introduction to Visual Studio

In this section, you will learn how to use Visual Studio (VS) to create the user interface and write C# programs. Visual Studio is an integrated development environment (IDE) for developing applications in a variety of languages, including C#, Visual Basic, C++, Python and HTML/JavaScript. VS supports development of desktop and web applications for Windows, Android and iOS. A major strength of Visual Studio is that it provides a user-friendly environment for developing applications.

Installing Visual Studio

You are encouraged to use the free version, Visual Studio Community 2019 or Visual Studio Community 2017. The 2015 version generally works well, except that you won't be able to run a few programs that use certain features available only in C# version 7.1 or greater. The instructions given in this textbook on using Visual Studio are based on VS Community 2017. This document gives the updated instructions for VS Community 2019. Please note that windows that are different only in appearance, not in content, are not updated.

You may download Visual Studio Community from the following website:
https://www.visualstudio.com/downloads/

To install Visual Studio, open the downloaded .exe file and run it.

Components of Visual Studio

Though you have the choice to write programs in a variety of languages, the CPU can understand only **Machine Language,** which is extremely difficult for humans to understand. Machine Language requires detailed instructions that consist of patterns of bits (0 and 1), like 10001010, and are dependent on the machine (the specific type of computer). Because of the need to write detailed machine-dependent instructions, Machine Language is called a **low-level language**.

Except in special cases, programs are written in **high-level languages,** like C#, which require fewer statements, are less dependent on the hardware and are easier to understand because they use words rather than patterns of bits.

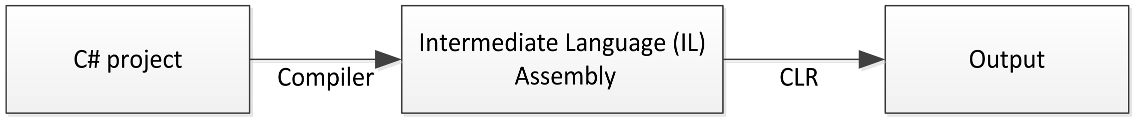
Programs written in high-level languages are translated to Machine Language before they are run. Different languages use different methods to translate and run programs. Many languages use a special software called a **compiler** to translate the source code to Machine Language. The compiler typically produces a separate executable Machine Language program that can be run any number of times without having to compile every time it is run.

Some programming languages use an **interpreter** that translates each statement to Machine Language and runs it without producing an executable program. So, every time the program is run, it needs to be translated to Machine Language.

Visual Studio uses a compiler, but it translates the source code to an **intermediate language**, which is translated to Machine Language and run using another software, as described later in this section. Thus, Visual Studio translates your statements to Machine Language in two steps:

First, a **compiler** converts the source code into a language called Microsoft Intermediate Language (IL). The compiled code, along with references to prebuilt programs (called classes), is stored in an executable file called **Microsoft Intermediate Language (IL) Assembly**. Such files have the extension .exe or .dll.

Next, another software called **Common Language Runtime (CLR)** translates the assemblies from Intermediate Language to Machine Language and executes the programs. The process of translating and running the source code may be represented as follows:



The products that support developing and running programs within the Visual Studio family include the following:

1. An **Integrated Development Environment (IDE)**
   An IDE provides an environment to develop programs, which includes code editors for **Visual C#, Visual Basic, Visual J#, Visual C++, HTML and XML,** and designers for **Windows forms** and **web forms.**
   In Visual Studio, a software application typically is organized into **Projects** that may contain one or more **forms.**
   Forms provide the **user interface** that allows users to input data for the program, to interact with the program and to display results.

2. **A compiler** that translates the source code into **Microsoft Intermediate Language (MSIL)**

3. **.Net Framework**, which includes
   a. the **Common Language Runtime (CLR)** that translates the assemblies from Intermediate Language to Machine Language and executes the programs, and
   b. .Net Framework **Class Library** that includes a large number of prebuilt programs called **classes**.

Running a program, as described earlier, consists of (1) the compiler translating the source code (the project) into Microsoft Intermediate Language Assembly and (2) the Common Language Runtime translating the assemblies from Intermediate Language to Machine Language and executing the program to produce the output. Next, we will look at how to work with Visual Studio to develop C# programs.

Review Questions

1.7 What is the only programming language that the CPU can understand?

1.8 What is a compiler?

1.9 What is an interpreter?

1.10 What is Microsoft Intermediate Language Assembly?

1.11 What is the function of Common Language Runtime in Visual Studio?

Creating an Application in Visual Studio

To become familiar with the Visual Studio environment, let's create an application for an ice cream parlor, consisting of a simple Windows form to compute and display the cost for an order of ice cream. We will follow the five steps identified earlier to develop it:

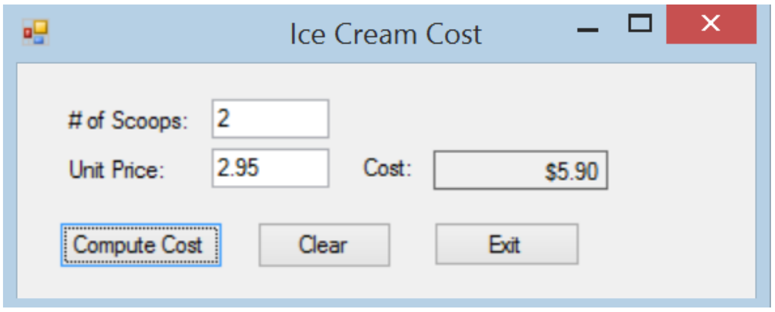
1. Define the purpose

Purpose:	Compute and display the total cost for an order of ice cream
Input:	Unit price, number of scoops
Process:	Compute total cost
	Total cost: unit price * number of scoops
Output:	Total cost

2. Design and develop the Graphical User Interface (GUI)

Figure 1-4 presents the design of the form that shows the user interface to let the user enter the number of scoops and unit price to display the cost. You will create the form in Tutorial 1.

Figure 1-4: Windows form to compute ice cream cost



3. Identify the components and logic of the program

The major subtasks include

Get number of scoops and unit price
Compute cost
Display cost

4. Design and develop files/databases

This application doesn't involve the use of files or databases.

5. Write and test the code

To understand the process of developing the entire application, including the code, let's create the form presented in step 2 and write the code for the tasks identified in step 3.

Creating a Windows Form

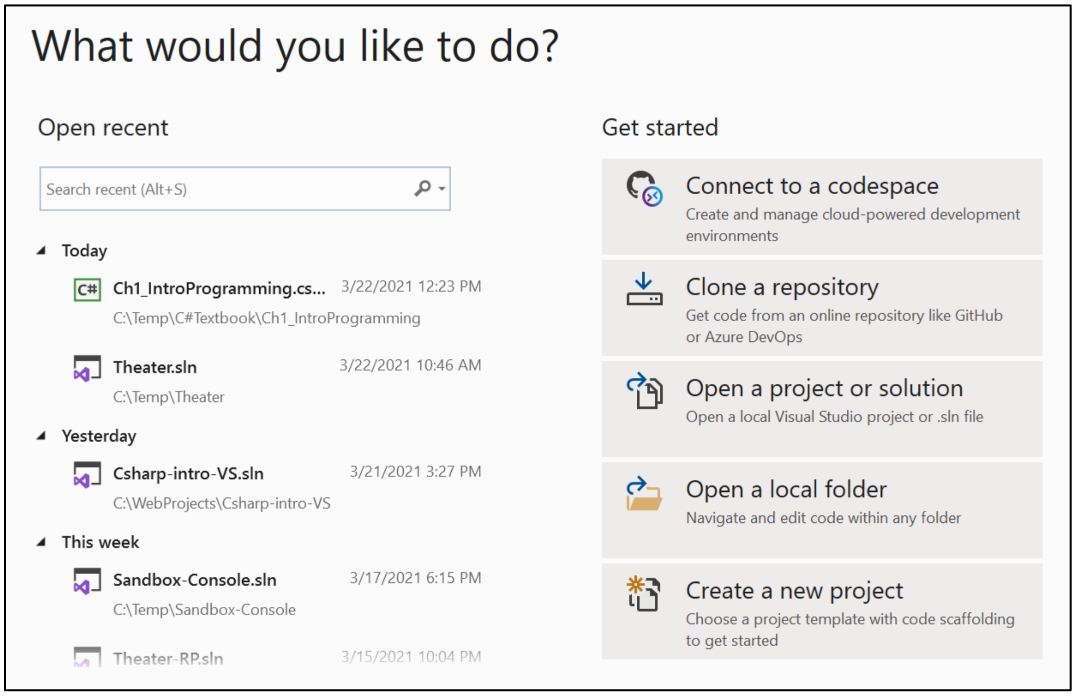
In Visual Studio, forms are created within a **Project**. Typically, all forms within a Project relate to a common task. Creating a project also creates a **Solution,** which is a container for one or more projects. Each project we discuss in this book is in a separate solution that has the same name as the project**.**

Tutorial 1 creates the project and the form to compute the cost of an order of ice cream.

Tutorial 1: Creating a Form: Ice Cream Cost

Step 1-1: Open Visual Studio Community. You will see a start page that resembles the one in Figure 1-5, except that the list of recently opened projects will be different.

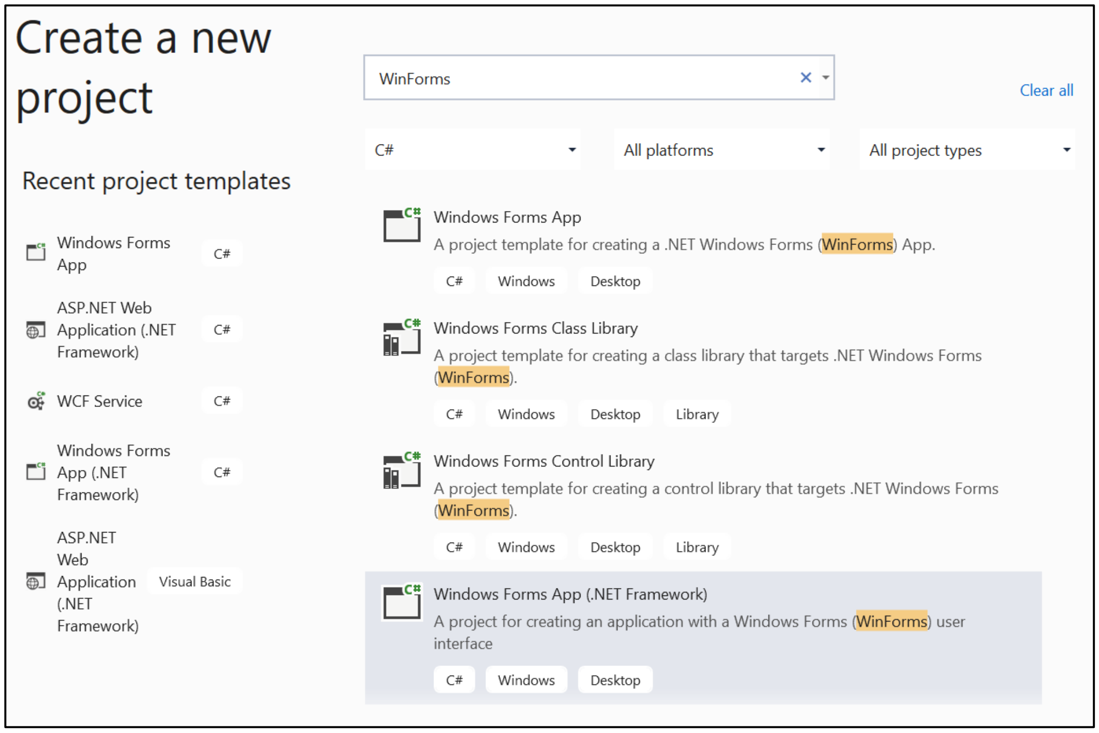
Figure 1-5: Visual Studio start page


What would you like to do?
Open recent
Search recent (Alt+S)
Today
Ch1_IntroProgramming.cs... 3/22/2021 12:23 PM
C:\Temp\C#Textbook\Ch1_IntroProgramming
Theater.sln 3/22/2021 10:46 AM
C:\Temp\Theater
Yesterday
Csharp-intro-VS.sln 3/21/2021 3:27 PM
C:\WebProjects\Csharp-intro-VS
This week
Sandbox-Console.sln 3/17/2021 6:15 PM
C:\Temp\Sandbox-Console
Theater-RP.sln 3/15/2021 10:04 PM
Get started
Connect to a codespace
Create and manage cloud-powered development environments
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started

Step 1-2: Create a new project.

Click *Create a new project* on the start page. You will see the *Create a new project* window that displays a list of available project templates on the right side of the window. To narrow down the list, type *WinForms* in the search box and select *C#* for language type, as shown in Figure 1-6.

Figure 1-6: *Create a New Project* window


Create a new project

WinForms

Clear all

C# | All platforms | All project types

Recent project templates

Windows Forms App C#

ASP.NET Web Application (.NET Framework) C#

WCF Service C#

Windows Forms App (.NET Framework) C#

ASP.NET Web Application (.NET Framework) Visual Basic

Windows Forms App

A project template for creating a .NET Windows Forms (WinForms) App.

C# Windows Desktop

Windows Forms Class Library

A project template for creating a class library that targets .NET Windows Forms (WinForms).

C# Windows Desktop Library

Windows Forms Control Library

A project template for creating a control library that targets .NET Windows Forms (WinForms).

C# Windows Desktop Library

Windows Forms App (.NET Framework)

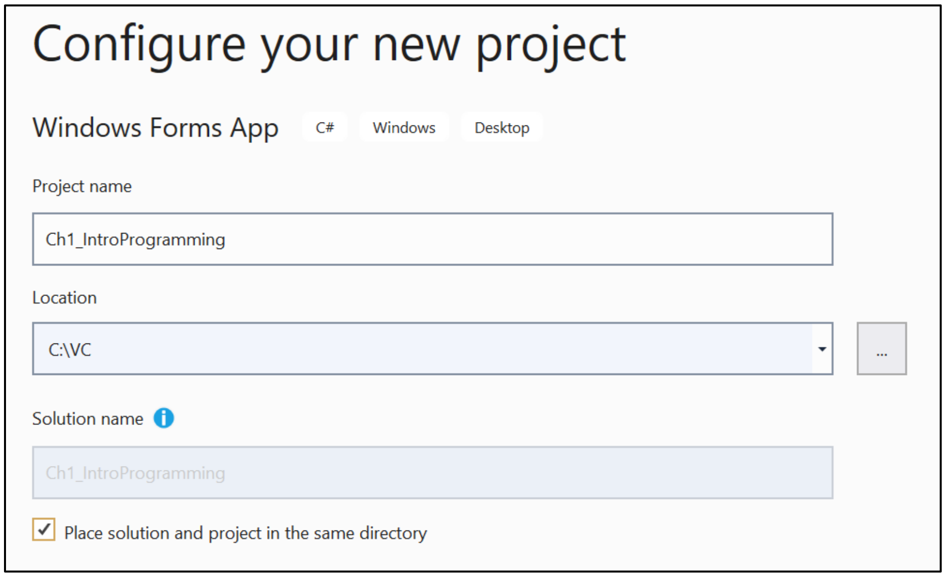
A project for creating an application with a Windows Forms (WinForms) user interface

C# Windows Desktop

Select *Windows Forms App (.NET Framework).*

Click *Next*. You will see the *Configure your new project* window, shown in Figure 1-7.

Figure 1-7: *Configure your new project* window


Configure your new project

Windows Forms App C# Windows Desktop

Project name

Ch1_IntroProgramming

Location

C:\VC

Solution name

Ch1_IntroProgramming

[✓] Place solution and project in the same directory

Step 1-3: Configure the project

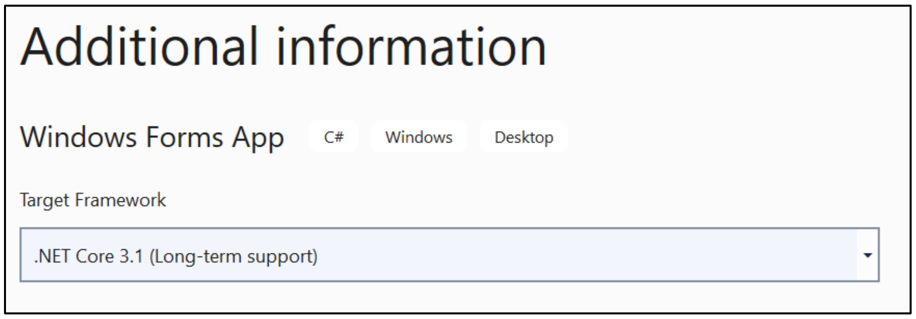
For *Project name*, enter Ch1_IntroProgramming.

Select the *Location* where you want to store the project. The instructions in this book assume that your project folder is C:\VC.

Check the CheckBox for *Place solution and project in the same directory*, so that both the Project and the Solution will be created in the same folder (If there are multiple Projects within a Solution, it would be better to have separate directories for the Solution and Projects.) Note that the default name for the Solution is the same as that of the Project.).

Click *Next*. You will see the Additional information window, as shown in Figure 1-8

Figure 1-8: *Additional information* window


Additional information

Windows Forms App

Target Framework

.NET Core 3.1 (Long-term support)

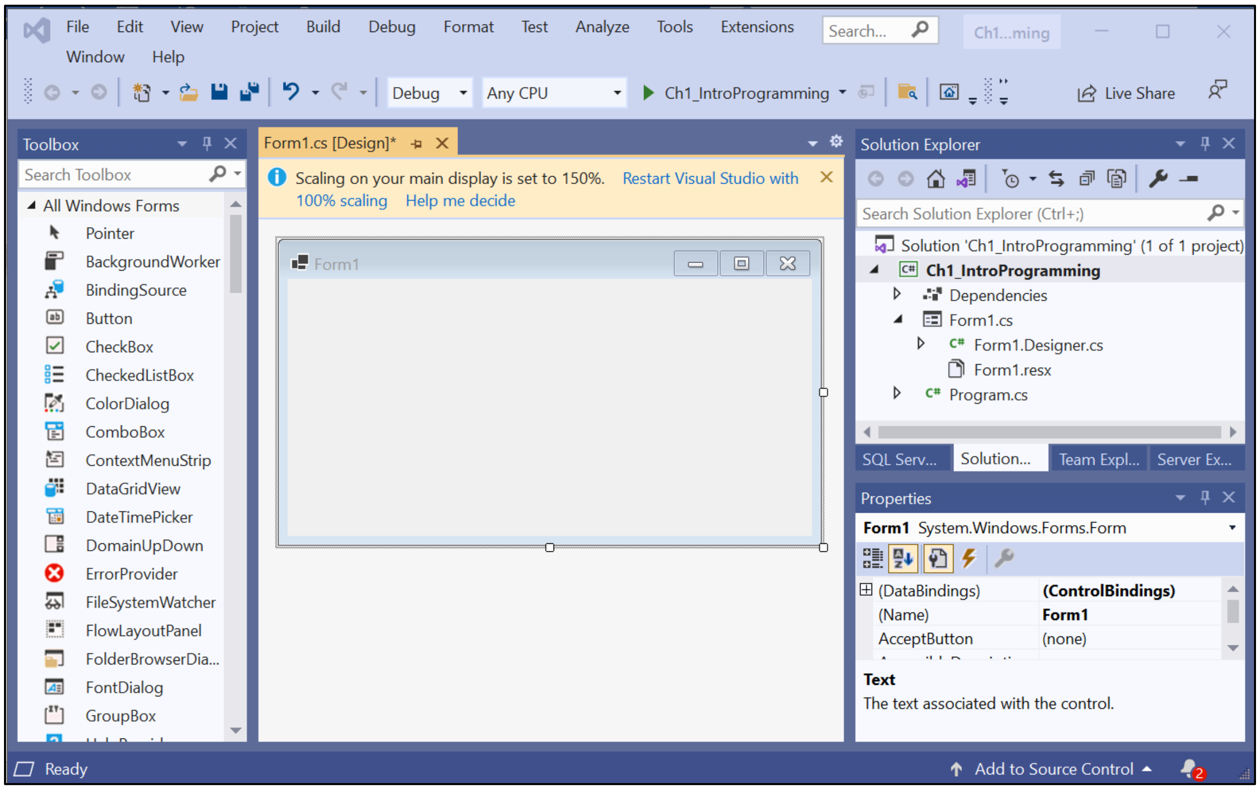
Click *Create*. The Visual Studio development environment appears, as shown in Figure 1-9, except that the Toolbox and Properties windows may not be open.

1.3 Visual Studio Environment

This section introduces the various windows and the environment within Visual Studio.

Figure 1-9 shows the default form named **Form1** in the **Designer** window, the **Toolbox** on the left, and the **Solution Explorer** window and the **Properties** window on the right.

Figure 1-9: Visual Studio environment



Scaling on your main display is set to 150%.

Form1 System.Windows.Forms.Form

(DataBindings)	(ControlBindings)
(Name)	Form1
AcceptButton	(none)

Text
The text associated with the control.

The appearance of the form will vary with the version of Visual Studio and Windows you use.

The **Designer** window allows you to create the user interface by adding controls to a form.

The **Toolbox** contains the components to build the user interface. The items within the Toolbox (e.g., ComboBox, TextBox and Label) are called **controls**. If the Toolbox is not open, select *View* > *Toolbox* from the menu.

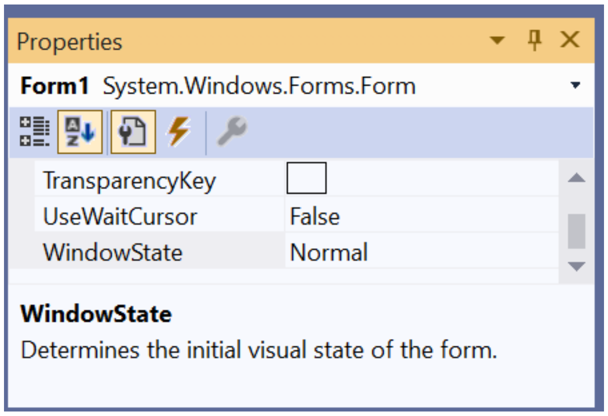
The **Solution Explorer** window shows the name of the Solution, which is a container of projects; the name of the project (or projects, if there are multiple projects); and the names of the files within each project. If the name of the Solution is not displayed, select

Tools, *Options*, *Projects and Solutions*, *General* and check *Always Show Solution*

The **Properties** window displays and lets you set the properties of the currently selected object in the *Designer* window. For example, the *Name* field in the Properties window shows that the name of the form is **Form1**. If the Properties window is not open, select *View* > *Properties Window*.

A brief description of each property is displayed at the bottom of the properties window when you click on the property, as shown in Figure 1-10. Make sure you expand the description area to make the description visible. If the description is not displayed, right click the property and check *Description*.

Figure 1-10: Description of properties



TransparencyKey	
UseWaitCursor	False
WindowState	Normal

WindowState
Determines the initial visual state of the form.

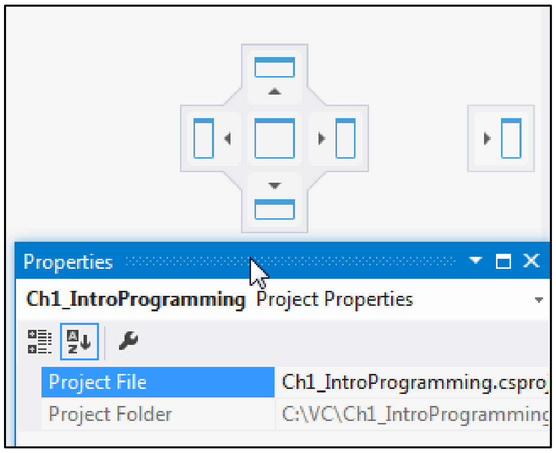
Displaying a closed window

If any window is not displayed, select *View* from the menu, and select the corresponding window.

Changing the docking position of a window

You may change the docking position of a window, such as the Properties window, by grabbing it with the pointer and then dropping it in one of the four positions (top, bottom, left, right), shown in Figure 1-11.

Figure 1-11: Docking positions



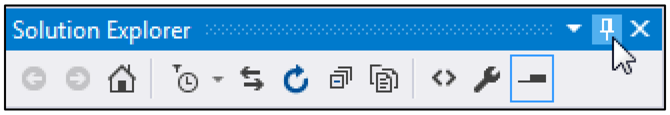
Project File	Ch1_IntroProgramming.cspro
Project Folder	C:\VC\Ch1_IntroProgramming

Undocking (floating) a window

By default, all windows are attached to a side of the Visual Studio window. To undock a window, right click its title bar, and select float. To dock a floating window, grab it with the pointer and drop it in one of the positions shown in Figure 1-9.

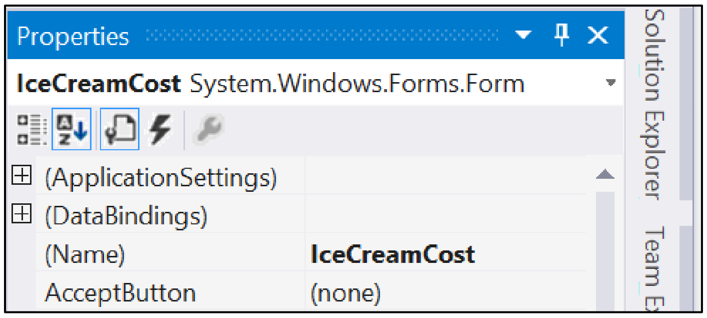
Autohide

The pushpin icon on a window, as shown here, allows you to hide the window and show it as a tab on the side.



Click the pushpin to turn autohide on and reduce the window to tab, as shown in Figure 1-12. Notice that the Solution Explorer is now a vertical bar to the right of the Properties window.

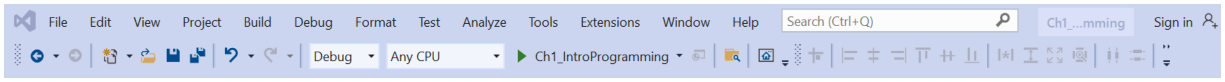
Figure 1-12: Solution Explorer with autohide on



To turn autohide off, click on the tab and click the pushpin.

Menu and toolbar items

The menu and toolbar, shown here, provide access to an extensive set of functions.



The function of each toolbar item is displayed in a **ToolTip** box when you hover the mouse pointer over the item. For a general understanding of the functions, you may view the descriptions of the Toolbar items by displaying the Tooltips. You will become more familiar with them through the projects in this book.

Adding and deleting forms

You may add a form to the project as follows:

Select: *Project, Add Form (Windows Forms)*. You will see the *Add New Item* window.

Make sure Visual C# Items and Windows Forms are selected.

Type in a name for the form. Click *Add.*

To delete a form,

Right click the name of the file (.cs) in the Solutions window, and select *Delete*.

Deleting a file deletes the form and all files associated with the form permanently.

Excluding a form from the project

You may exclude a form from the Solution Explorer but still keep the files associated with the form, in the project folder, as follows:

Right click the file name (.cs) in the *Solution Explorer*, and select *Exclude from Project*.

To bring back a form that was excluded from the project,

Select *Project, Add Existing Item*.

Select the file (.cs) from the project folder.

The form will be added to the Solution Explorer.

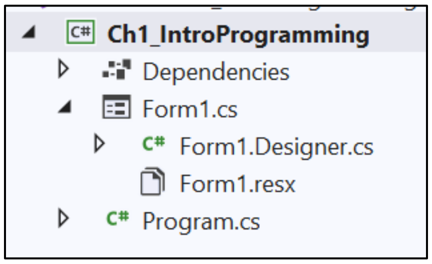
Changing the name and title of a form

The *Name* field in the Properties window shows that the current name of the form is **Form1**.

The Solution Explorer also displays a file named **Form1.cs**, which is the file that contains the code**.** Expanding this file, by clicking the triangle to its left, displays two other files,

Form1.Designer.cs that contains the generated code representing the design of the form and **Form1.res** that contains additional resources associated with the form,

as shown below:



Guidelines to name a form

Names of forms (and classes, in general) must start with a letter, and can contain only letters, numbers or the underscore character (_). However, the use of the underscore character is not recommended.

The convention in C# is to use the **Pascal Case** to name forms: The first letter of the form name and the first letter of subsequent words, if any, are capitalized, as in IceCreamCost and FinancialPlanner.

Use a meaningful name and avoid using abbreviations or acronyms that are not widely recognized. A noun or noun phrase is recommended for a form name, though you will find a few form names in this book that violate this guideline.

Now that you are familiar with the environment, let's develop the application using the default form, Form1, in the project that you created.

Step 1-4: Change the form name.

Right click *Form1.cs,* and rename it **IceCreamCost.cs**. On the window that asks whether you want to change all references, select *Yes*. Notice that the name of the form in the Properties window and the names of all three files under Form1.cs are changed. Note that the title of the form still remains *Form1*.

Step 1-5: Change the form title.

To change the title, select the Text property of the form in the Properties window and change it to **Ice Cream Cost**.

Specifying the startup form

A project typically contains multiple related forms. The Program.cs file contains the startup program called Main that includes the following statement, where *FormName* represents the name of the startup form.

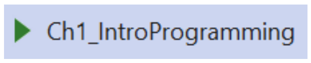
Application.Run(new *FormName*());

Step 1-6: Double click on Program.cs in the Solution Explorer window, and make sure that the name of the startup form is IceCreamCost, as shown below:

```
Application.Run(new IceCreamCost());
```

Running a form

Step 1-7: Run the form by clicking the *Start* button (shown below) from the Toolbar.



Visual Studio displays the empty form. Because you haven't added any controls to the form, there is not much you can do with it. Close the form by clicking the Close button ("x") at the top right of the form.

Additional properties of forms

Let's look at a couple of properties that apply to the form only when it is run.

Click on the *start position* property of IceCreamCosts.cs, and view its description displayed at the bottom of the Properties window. If the description is not visible, expand the description area to make it visible.

Click the dropdown arrow to the right of StartPosition to view the options. Select *Center Screen* from the list. Run the form and see how the property impacts the position of the form. Change the property back to *WindowsDefaultPosition*.

Repeat for *WindowState* property, setting it to *Maximized*.

Review Questions

1.12 What property of a form would you use to change the title displayed on the title bar of the form?

1.13 When you run a project containing multiple forms, which form does C# run?

Please proceed to Section 1.4 in the textbook.

Chapter 15

Introduction to Web Applications Development

In previous chapters, you used C# to build Windows form applications, which are installed on individual client computers and run on Windows. This chapter applies the knowledge you gained earlier to develop web applications that are installed on a web server and run on browsers on client computers. You will build web applications using the ASP.Net platform, which is part of the .Net Framework, in the familiar Visual Studio environment.

Learning Objectives

After studying this chapter, you should be able to:

- Describe the terms *web server*, *HTTP request and response*, *Web Forms* and *MVC approaches*, *code-behind file* and *Redirect*, *Transfer and Cross-page PostBack*
- Describe the differences between web application and Windows form application, static and dynamic web pages, client-side and server-side scripts, stateless and stateful modes, HTML controls and web server controls.
- Create a web page with web server controls.
- Use C# code-behind files to respond to events on a web form.
- Validate data using Validation controls.

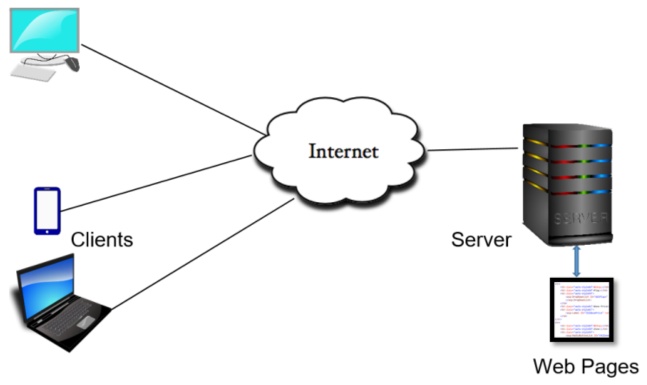
Topics

15.1 Introduction to Web Applications

Windows form applications, like those you developed in previous chapters, are installed and run on client computers with the Windows operating system. Hence, to access a Windows form application from a computer, the application must be installed on that computer.

Web applications, on the other hand, are installed and run on a server(s) and are accessed through a browser from clients like laptops, smartphones and tablets over the Internet. Figure 15-1 shows the components of a simple web application that consists of clients, the Internet and a **web server** that stores, maintains and delivers web pages to the clients. To carry out these functions, the server is configured with *web server software* like the Microsoft Internet Information Server (IIS), Apache or Nginx. The term *web server* is often used to refer to the server hardware and/or the software. Because web applications are accessed through a browser, the clients don't have to be Windows machines.

Figure 15-1: Components of a web application



How Does a Client Display a Web Page from the Server?

From a user's perspective, displaying a web page on a client is simple. You type the URL of the page into a browser, select a bookmark, or click a link. For instance, the following URL would display the *ordering* page on the **domain** *prospectpressvt* within the top-level domain *.com*:

http://www.prospectpressvt.com/ordering

The first two components of the above URL are:

http:	The **web server protocol** used by the client and the server to communicate with each other. Other types of protocols include SMTP (e-mail server protocol) and FTP (File Transfer Protocol).
www	The default **host name**. If you don't specify a host name, it is assumed to be www. Thus *yahoo.com* is the same as www.yahoo.com. However, *finance.yahoo.com* finds the domain *yahoo.com* on the host *finance*.

Now let's look at what happens behind the scenes when you display a web page. We will start with the simpler case of **static** pages and then look at the more general **dynamic** pages.

Static and Dynamic Web Pages

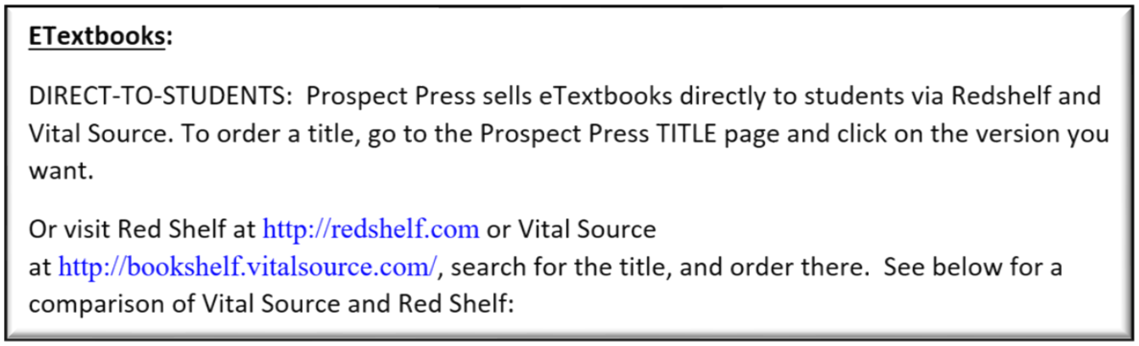
A **static** web page remains the same each time it is displayed, unless it is manually altered. Thus a web page that displays product prices would be a static page if the prices are "hard-coded" into the contents of the page. However, it would be a dynamic page if the prices are read from a file/database and incorporated into the web page each time it is displayed. A dynamic web page typically is created on the fly when the page is requested. Dynamic pages may also accept user input and process it.

In addition to static and dynamic web pages, a website might contain additional static files like .css files that contain formatting information, .js (JavaScript) files that contain client-side scripts and .cs files containing server-side C# code.

Displaying Static Web Pages

To understand how static web pages are displayed, let's look at how a static page is stored on the server. Though the browser displays a web page with formatted text and graphics, a static web page is stored on the server as HTML markup that specifies the contents and format of the web page. Note that HTML is referred to as a markup language rather than a programming language, because it is used primarily to control the presentation of information. The files that contain the HTML markup generally have .htm or

.html extensions. The HTML code for a static page typically includes the text displayed on the page. As an example, here is a segment of the text displayed on the Prospect Press website:



ETextbooks:

DIRECT-TO-STUDENTS: Prospect Press sells eTextbooks directly to students via Redshelf and Vital Source. To order a title, go to the Prospect Press TITLE page and click on the version you want.

Or visit Red Shelf at http://redshelf.com or Vital Source at http://bookshelf.vitalsource.com/, search for the title, and order there. See below for a comparison of Vital Source and Red Shelf:

The corresponding HTML markup looks like this:

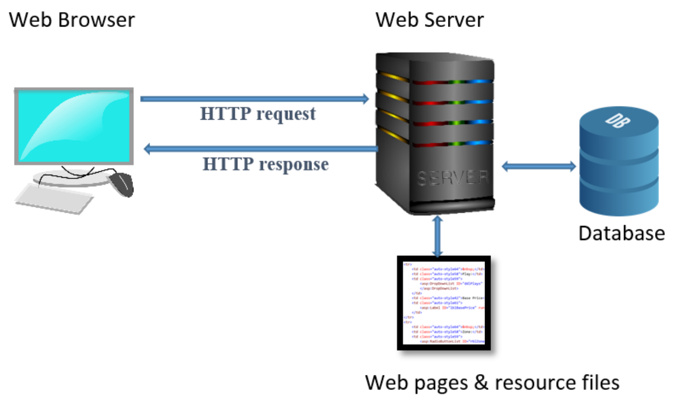
```
<p><strong><u>ETextbooks</u></strong><strong>:</strong></p>
<p>DIRECT-TO-STUDENTS:  Prospect Press sells eTextbooks directly to students via Redshelf and Vital
Source. To order a title, go to the Prospect Press TITLE page and click on the version you want.</p>
<p>Or visit Red Shelf at <a href="http://redshelf.com"
target="_blank">http://redshelf.com</a> or Vital Source at <a
href="http://bookshelf.vitalsource.com/">http://bookshelf.vitalsource.com/</a>, search for the title,
and order there.  See below for a comparison of Vital Source and Red Shelf:</p>
```

We will discuss the structure of HTML later in this chapter.

What happens when you click on a link or type in a URL and press the Enter key to display a static web page? The browser on the client opens the connection to the server and sends a **request** (**HTTP request**, to be more precise) to the web server, which essentially tells the web server, "*Hey, get me the file specified in this URL*." The server retrieves the file that contains the HTML for the page and any related files, and sends a **response** (**HTTP response**) that includes the files back to the browser. Thus a static page is delivered to the browser exactly as stored on the server.

When the browser receives the response, it closes the connection, creates the web page as specified in the HTML code and displays it. The user won't be able to modify or interact with the static page displayed in the browser. The combination of request and response is called a *transaction*. Figure 15-2 shows the process.

Figure 15-2: HTTP request and response



Displaying Dynamic Web Pages

The process of displaying dynamic and static web pages involves an HTTP request by the client and a response from the server. However, displaying dynamic pages involves additional processing on the server because they are created on demand. That is, segments of the page are generated dynamically using **scripts** embedded in HTML or stored in a separate file(s). The scripts may perform tasks like looking up information from a database, displaying dynamic data like inventory levels and account balances and customizing the web page. So, a dynamic page may look different each time it is displayed, depending on various factors like who the user is, the date/time, changes in the data read by the script and the parameter values passed through the script.

What is a script? A script is essentially a type of code that is written in interpretive scripting languages like JavaScript, PHP and CGI, to control the behavior of applications. For instance, you may validate input data on a web form using a script written in JavaScript and executed by the browser. A program, on the other hand, is code written in programming languages like Java and C# that are compiled. Scripts are easier to develop but slower to run. In addition to code written in scripting and programming languages, web pages also use code written in markup languages like HTML and CSS (Cascading Style Sheet), which are described later.

It should be realized that the difference between scripting and programming languages is becoming blurred. Though, strictly speaking, the term *script* means code written in interpretive languages, it is generally used to represent any code that works with a web page, including code written in a programming language like C#.

There are two approaches to running scripts. If the script is run on the server, it is called *server-side scripting* or *server-side coding*. If the script is run on the browser, it is called *client-side scripting* or *client-side coding*. Typical web applications today use both server-side and client-side coding.

Server-Side Scripting

In this chapter, you will develop web applications using the ASP.NET framework, described shortly. You are likely to work in an environment where a single server plays the role of a web server (that handles HTTP requests), as well as an application server that provides an environment to execute scripts and create dynamic web pages. In addition, it becomes a database server by running SQL Server DBMS on it. However, generally, each of these three functions could be performed on a separate physical server to improve security and performance.

When a web server receives a request for a web page that includes a server-side script, the server executes the script, generates the HTML code for the web page and sends it to the browser. Because the server-side script is run on the server and only the result is sent to the browser, server-side scripting has the advantage that the users won’t be able to view the script in the browser. However, it puts an additional burden on the server.

Server-side scripts typically include ASP.NET code and code written in languages like ASP.NET, PHP, CGI and JSP, which are not compiled, and programs written in programming languages like C#. It should be noted that ASP.NET is a framework for developing web applications, and not a programming language. In this chapter, you will use IIS (Internet Information Services) Express as the web server, ASP.NET as the application server to build ASP.NET applications and SQL Server Express as the DBMS for the database server.

Client-Side Scripting

A client-side script could be embedded in the HTML code for a page or in a separate file. But it is run on the client (in the browser) when the browser receives the web page from the web server. That would mean you can also view the script by displaying the source code of a page in the browser. JavaScript is currently the most popular language for client-side scripting.

Client-side scripts are typically used to make web forms more interesting, user-friendly and interactive. Applications of client-side scripts include data validation, running multimedia and creating gadgets like calculators. Running client-side scripts is simpler because it doesn't involve the application server.

Stateless and Stateful Modes

HTTP is a stateless protocol. That means that as soon as a client receives a response for its request, the connection between the client and server is terminated, and the server keeps no information about the client. So, if the same client makes another request, the server has no way to know whether it is the same client or another client. The stateless mode of communication between the server and the client is fine for applications involving only simple static pages. However, it does not meet the requirements of today's typical dynamic applications.

ASP.Net provides several features to maintain the state, including view state, session state, application state and server-side caching. You will learn more about view state and session state later. In the stateful mode, the server keeps track of information, such as who the user is and what he does from one page to the next, until the user logs out.

ASP.NET and ASP.NET Core

ASP.NET is a traditional framework for building web applications. It is based on the .NET Framework that you have used in previous chapters; thus all .NET Framework features like type safety, inheritance and compatibility with .NET languages are available for ASP.NET development.

As described earlier, ASP.NET applications are typically developed in a network environment using an Internet or Intranet. However, you may use a stand-alone development environment where a single computer serves as the client and server. In the stand-alone environment, you may use the *IIS Express web server* that comes with Visual Studio as the web server and *SQL Server Express* as the database server.

ASP.NET Core is a relatively new framework for developing web applications. ASP.NET Core is an open-source cross-platform framework, whereas ASP.NET is Windows-based. ASP.NET Core supports multiple operating systems, including Windows, MacOS and Linux. However, ASP.NET does not.

Another key difference is the web development approaches supported by the two frameworks. ASP.NET supports *ASP.NET Web Forms*. Similar to Windows Forms, ASP.NET Web Forms are designed to work with drag-and-drop server controls, making this approach particularly suitable for rapid application development.

ASP.NET Core supports *Razor Pages* and *Model View Controller (MVC)*, which are more complex than ASP.NET Web Forms, but provide advantages in maintenance and testing. It should be noted that ASP.NET supports an earlier version of MVC, called ASP.NET MVC.

This chapter discusses development of ASP.NET Web Forms using Visual Studio Community.

Review Questions

15.1 What is the domain name in the URL https://www.createspace.com? What is the host name?

15.2 How does a dynamic web page differ from a static web page?

15.3 What is an HTTP request?

15.4 What are two differences between server-side and client-side scripts?

15.5 What is the drawback of the stateless mode of communication between the server and the client?

15.2 Creating a Single-Page ASP.NET Website

In Visual Studio, a website is referred to as a *web project*. The term *web application* also is used interchangeably to refer to websites. To create a web project, you choose a *template* from a set of five different templates, as described shortly: *Empty*, *Web Forms*, *MVC*, *Web API* and *Single Page Application*.

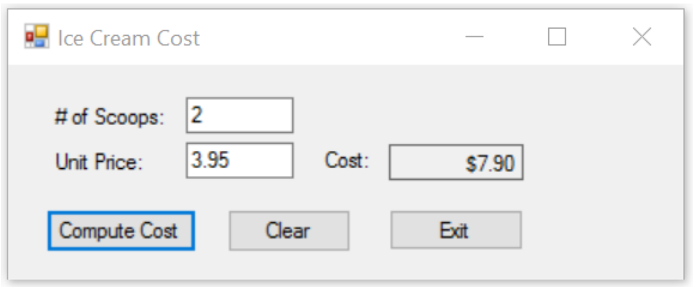
Within the website, you will develop web pages using the Web Forms approach. A web page developed using the Web Forms approach is also called a web form. A web form, which uses web server controls to build user interfaces, also may contain HTML markup, CSS, server-side scripts and client-side scripts, though as you will learn, server-side scripts are typically placed in a separate code-behind page.

Typically, an ASP.NET web project is developed in a network environment involving a server that is configured with IIS as the web server software. However, it is more convenient to develop a website in a stand-alone environment where a single computer serves as both the client and the server. The process of developing websites is essentially the same in both environments. For the sake of convenience, you will use the *stand-alone environment*.

Tutorial 1: Developing a Simple Web Page: Ice Cream Cost

This tutorial develops a web page corresponding to the Ice Cream Cost Windows form, as shown in Figure 15-3, which you created in Tutorials 1 and 2 in Chapter 1. The user interface for the web page is similar, except that there is no Exit button:

Figure 15-3: The Ice Cream Cost Windows form

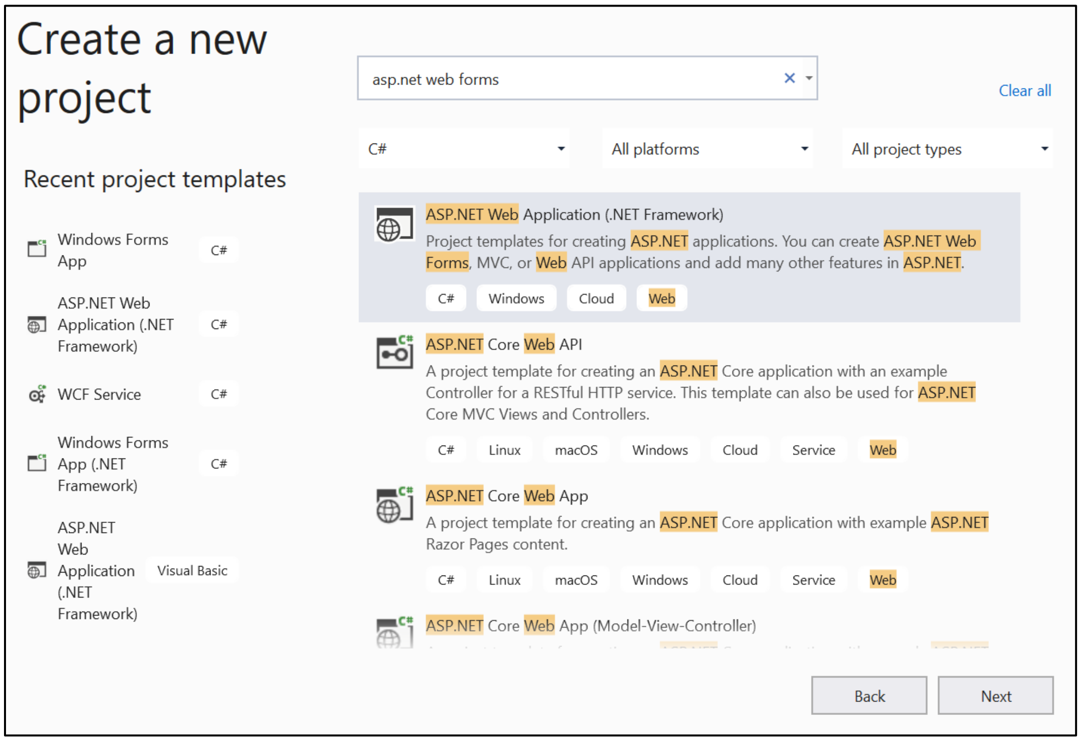


You will use the familiar Visual Studio environment to create a website and then add a web page to it.

Step 1-1: Create the website.

Open Visual Studio and select *Create a new project* from the start page.
You will see the *Create a new project* window, shown in Figure 15-4.

Figure 15-4: New Project window


Create a new project

asp.net web forms

Clear all

C# | All platforms | All project types

Recent project templates

Windows Forms App C#

ASP.NET Web Application (.NET Framework) C#

WCF Service C#

Windows Forms App (.NET Framework) C#

ASP.NET Web Application (.NET Framework) Visual Basic

ASP.NET Web Application (.NET Framework)
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.
C# Windows Cloud Web

ASP.NET Core Web API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web

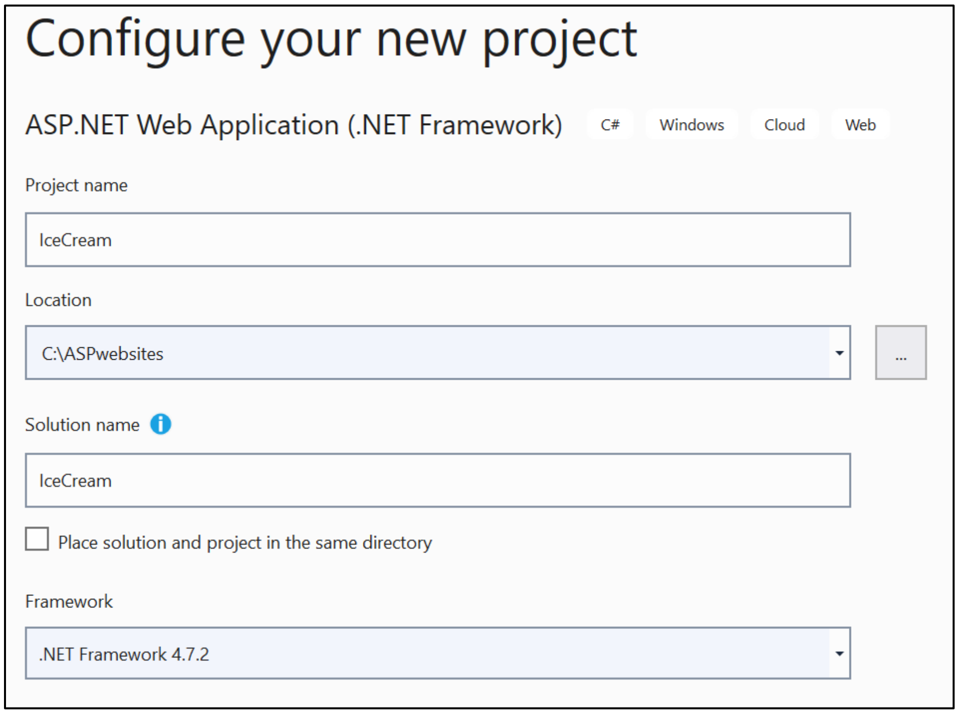
ASP.NET Core Web App
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)

Back | Next

Type in *asp.net web form* in the search box and select *C#* for language.
Select *ASP.NET Web Application (.NET Framework).*
Click *Next*. You will see the *Configure your new project* window similar to the one shown in Figure 15-5.

Figure 15-5: *Configure your new project* window



Configure your new project

ASP.NET Web Application (.NET Framework)

Project name: IceCream

Location: C:\ASPwebsites

Solution name: IceCream

☐ Place solution and project in the same directory

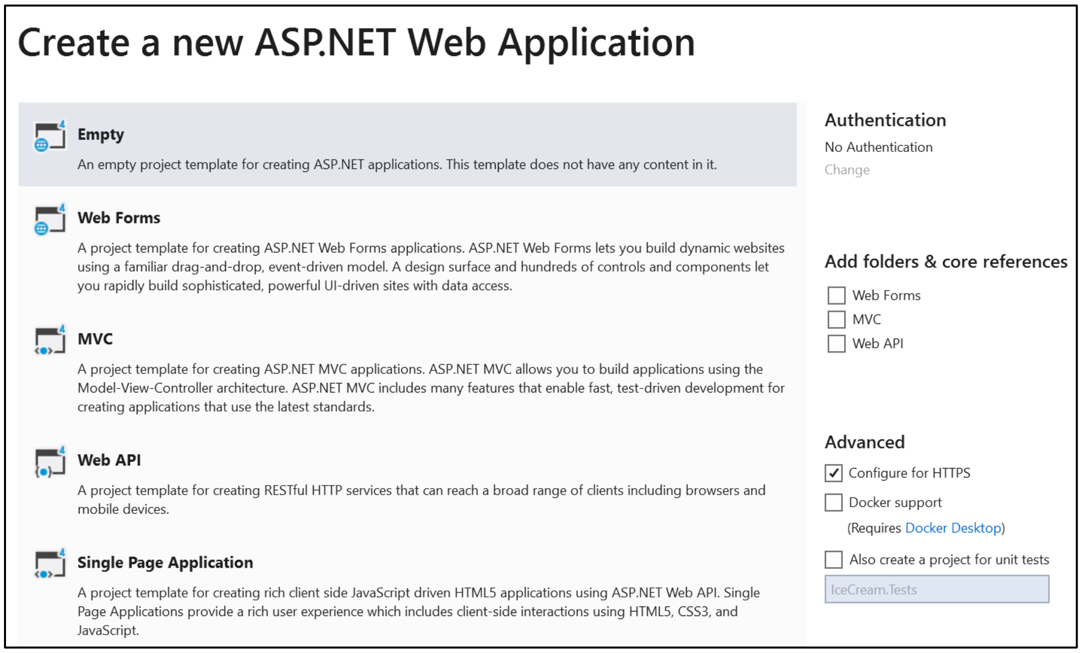
Framework: .NET Framework 4.7.2

Step 1-2: Configure the project.

Enter **IceCream** for the *Name* field.
Create a folder named **ASPwebsites** on your drive. You may do it in Windows Explorer.
Click *Browse* and select the ASPwebsites folder for *Location*.

Click *Create*. You will see the *Create a new ASP.NET Web Application* window, as shown in Figure 15-6, which lets you select a template for your project.

Figure 15-6: *Create a new ASP.NET Web Application* window



Create a new ASP.NET Web Application

Empty
An empty project template for creating ASP.NET applications. This template does not have any content in it.

Web Forms
A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

MVC
A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Web API
A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

Single Page Application
A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

Authentication
No Authentication
Change

Add folders & core references
☐ Web Forms
☐ MVC
☐ Web API

Advanced
☑ Configure for HTTPS
☐ Docker support
(Requires Docker Desktop)
☐ Also create a project for unit tests
IceCream.Tests

Templates for Web Projects

Figure 15-6 shows the different templates available to create a web project. The **Empty** template and the **Web Forms** template essentially use the same approach called the *Web Forms* approach to develop web pages. The Web Forms approach, similar to Windows Forms development, uses a design surface and drag-and-drop controls (rather than writing HTML code) to rapidly build rich user interfaces.

A web form may contain HTML markup, CSS, server-side scripts and client-side scripts. Ease of development is the key advantage of the Web Forms approach. However, other approaches like MVC provide advantages in certain areas, as described below.

You will use the **Empty** template in this chapter because of its simplicity. The Empty template, as the name indicates, creates a web project with no web pages, whereas a project created using the Web Forms template includes a default page and folders/files needed to provide additional functionality.

You may select the Empty template and still add a subset of the folders and references that are included in the Web Forms template by checking the *Web Forms* CheckBox in the section entitled *Add folders and core references for*.

The **MVC** template supports the MVC (Model View Controller) architecture, which makes it easier to do *separation of concerns* (i.e., separation of different aspects of software functionality like business logic, user interface and data access) to enhance maintenance and reuse of code and shared development. Thus it is particularly suitable for large projects involving multiple developers. Further, this approach provides some performance advantages and gives better control over generated HTML markup.

The **Web API** template lets you create and consume Web API (Application Program Interface) Services. The **Single Page Applications** (SPAs) help create highly interactive applications that include a single page. Rather than sending new pages to the client, the server sends data that dynamically updates the single page based on user interactions.

Step 1-3: Select a template for the project:

Select **Empty** template. Click *Create*.

Visual Studio creates the IceCream project. If you open the ASPwebsites folder, you will see a new folder named IceCream that holds the Solution file (IceCream.sln) and the project folder, also named IceCream, which holds configuration files and files that store information on web pages.

Adding a Web Form to a Website

A web form typically includes two types of code: (1) the C# (or other compatible language) code that deals with the application logic and (2) the HTML and CSS that deals with the user interface. The C# code is generally stored in a separate file called a **code-behind file**, which has the extension .aspx.cs (or .aspx.vb for Visual Basic, and so on). C# code also may be embedded in the HTML as **in-line code**.

The two-file system, which uses a separate code-behind file, provides some separation of the application logic from the user interface, making it easier to understand and maintain the code.

Step 1-4: Add a web form to the website:

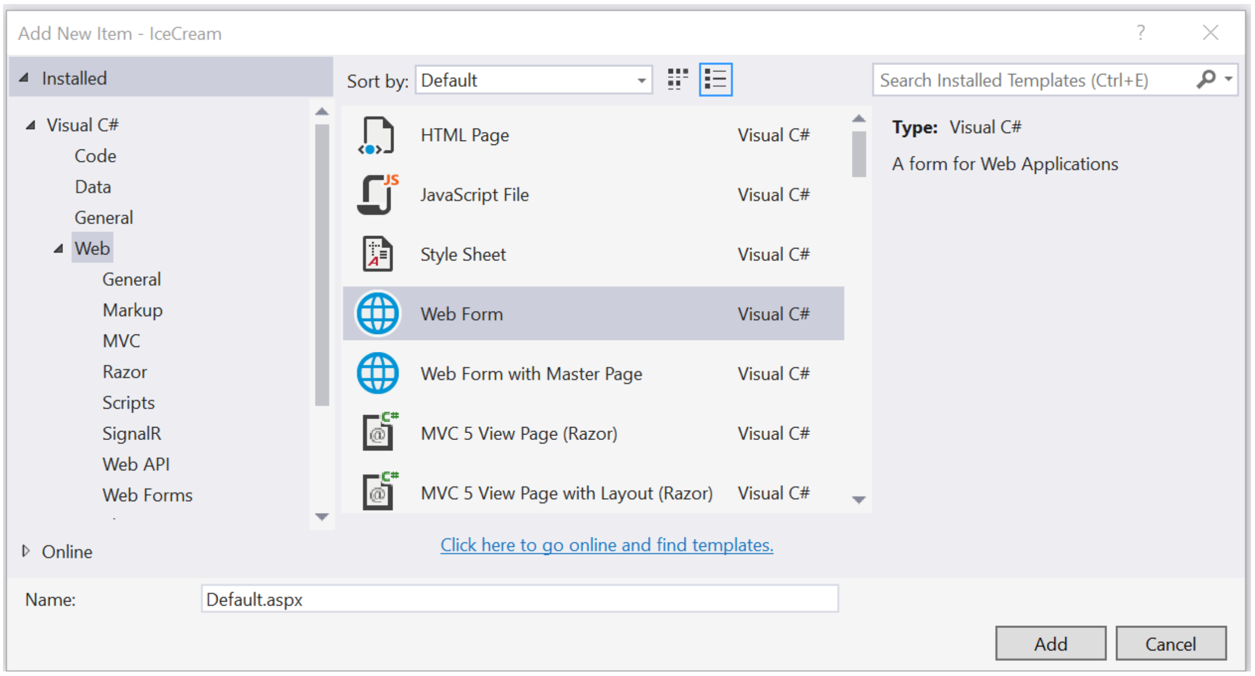
Right click the project name IceCream (not the Solution, IceCream) in the Solutions Explorer, and select *Add > New Item***.**
You will see the *Add New Item* window, similar to the one shown in Figure 15-7.

Select *Web Form.*
Change the name of the web form to **Default.aspx**.

The .aspx file stores the ASP.NET and HTML codes that specify the form. In addition to the Default.aspx file, there are two other files: the code-behind file, **Default.aspx.cs**, which contains the C# code, and **Default.aspx.designer.cs**, which stores certain code generated by Visual Studio.

An alternative is to select **Add > Add Webform**, which doesn't give you the option to add items other than web forms.

Figure 15-7: *Add New Item* window


Add New Item - IceCream
Installed
Visual C#
Code
Data
General
Web
General
Markup
MVC
Razor
Scripts
SignalR
Web API
Web Forms
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
HTML Page — Visual C#
JavaScript File — Visual C#
Style Sheet — Visual C#
Web Form — Visual C#
Web Form with Master Page — Visual C#
MVC 5 View Page (Razor) — Visual C#
MVC 5 View Page with Layout (Razor) — Visual C#
Type: Visual C#
A form for Web Applications
Click here to go online and find templates.
Add
Cancel

Name: Default.aspx

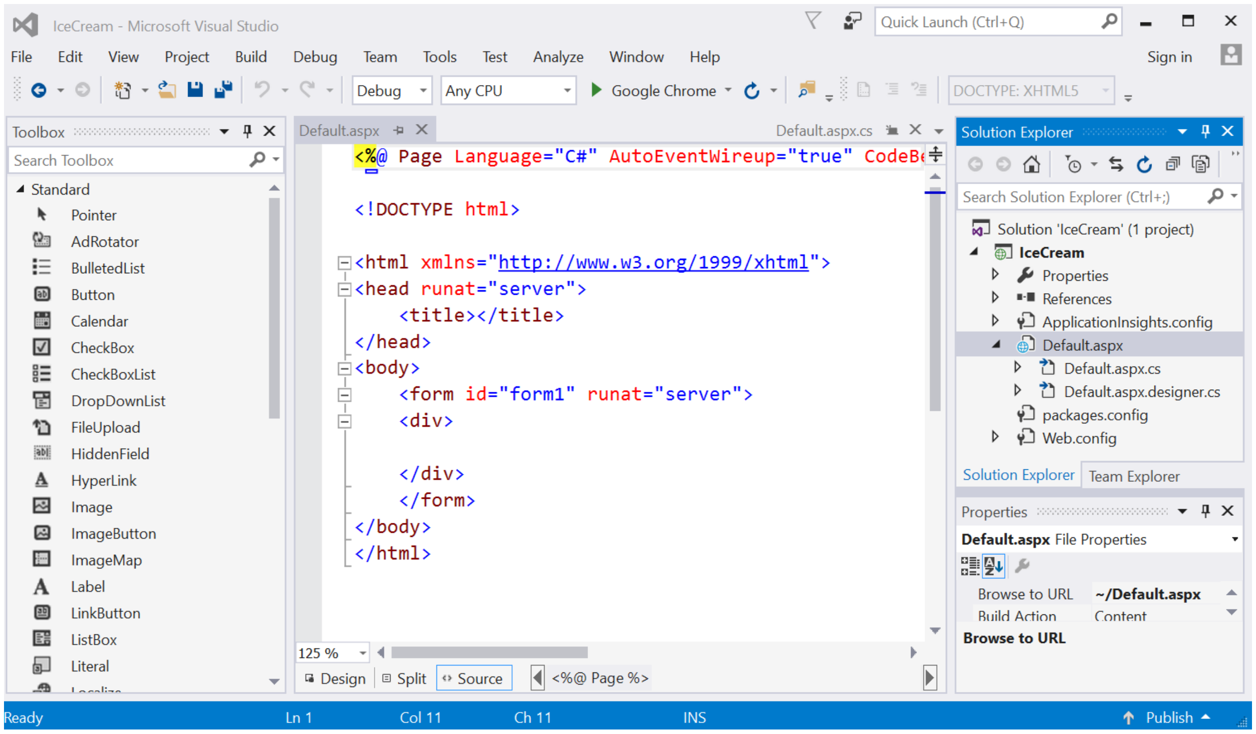
When you add a new web form, Visual Studio displays the **Web Forms Designer** window in **Source** view, which shows the source code for the page, This source code is generated by Visual Studio and stored in the **Default.aspx** file, as shown in Figure 15-8. For convenience, we will refer to the source code within the Default.aspx file as "**aspx code**."

You may change the view to **Design** or **Split** by clicking the corresponding buttons at the bottom of the window.

You may display the C# code window by clicking the Default.aspx.cs tab at the top of the window.

Note that the ASPwebsite\IceCream\IceCream folder now includes the additional files Default.aspx, Default.aspx.cs and Default.aspx.designer.cs.

Figure 15-8: The Designer window in Source view



```
<%@ Page Language="C#" AutoEventWireup="true" CodeBe

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

To help familiarize you with the code in the .aspx file, we will take a look at certain basic aspects of HTML in the next section.

Closing and Opening a Website

To close a website, select **File** > **Close Solution**.

This is how you open a website from Visual Studio:

From the start page, select *Open a Project or Solution*.
Select the Solution file *IceCream.sln* and click *Open*.
You also may select the Solution file from the list of recent projects on the start page.

When Visual Studio is closed, you may double click the Solution file *IceCream.sln* to open the project.

Please proceed to Section 15.3 in the textbook.