Understanding Document

PHASE - I

PREPARED BY

PINAL PAMBHAR

Table Of Contents

Different types of Windows

WINDOW NAME	DESCRIPTION	PATH TO OPEN WINDOW
Solution Explorer	Solution Explorer window enables us to explore and manage our solutions,projects and files.	View > Solution Explorer
Team Explorer	Team Explorer connects visual studio to control version systems like GitHub , Azure Devops.	View > Solution Explorer
Server Explorer	Server Explorer is a tool window that allows us to examine various server resources including databases.	View > Server Explorer
Cloud Explorer	Cloud Explorer enables the developer to view Azure resources, explore their properties, and perform several actions.	View > Cloud Explorer
SQL Server Object Explorer	SQL Server Object Explorer allows developers to create, edit and browse database objects such as Microsoft SQL server, SQL Azure etc.	View > SQL Server Object Explorer
Test Explorer	Test Explorer is used to running unit tests from visual studio.	View > Test Explorer
Bookmark Window	Bookmark Window displays all bookmarked lines in the code with file location and line number.	View > Bookmark Window

Call Hierarchy	Call Hierarchy displays all lines of code in which child member is called from parent member.	View > Call Hierarchy
Class View	Class View is part of Solution Explorer. It displays the elements of an application.	View > Class View
Code Definition Window	Code Definition Window displays the definition of a selected member in the active project.	View > Code Definition Window
Object Browser	Object Browser displays all available objects in the project and sees their properties, methods, and events.	View > Object Browser
Error List	Error List window displays the errors & warnings with description, project name, file name, and line number.	View > Error List
Output	Output window displays status messages of the compiler/build system as well as output from debugging sessions.	View > Output
Task List	Task List use to track code comments that use tokens such as TODO and HACK.	View > Task List
Notifications	Notifications window notifies developers about licensing, environment, extensions, and updates.	View > Notifications

Toolbox	Toolbox window displays control that we can add to the Visual Studio project	View > Toolbox
Properties Window	Properties Window is used for display and set properties of selected objects.	View > Properties Window
CSS Properties	CSS Properties window enables us to view and modify cascading style sheet(CSS) style properties.	View > CSS Properties
Manage Styles	Manage styles window provides a tree hierarchy view of all CSS selectors available from the current page.	View > Manage Styles
Command Window	Command Window is used to execute commands or aliases directly in the Visual Studio.	View > Other Windows > Command Window
Web Browser	Web Browser window opens a browser window in Visual Studio.	View > Other Windows > Web Browser
Test Results	Test Results shows the status of all tests like Active Runs, Queued Runs & Completed Runs.	View > Other Windows > Test Results
Data Tool Operations	Data Tool Operations window shows the progress of some operations like update database and notifies of any errors.	View > Other Windows > Data Tool Operations

Microsoft Azure Activity Log	This window provides insight into a subscription -level events. It provides information like when the resource is modified or when the virtual machine is started.	View > Other Windows > Microsoft Azure Activity Log
Application Insights Search	This Window is used to find and explore individual telemetry items such as page views, exceptions, log traces, etc.	View > Other Windows > Application Insights Search
Live Share	Live Share enables developers to collaboratively edit and debug with others in real time.	View > Other Windows > Live Share
Task Runner Explorer	Task Runner Explorer shows a list of available tasks is as simple as a click of the mouse.	View > Other Windows > Task Runner Explorer
Containers	Containers window allows you to view running containers, browse available images, view environment variables, etc.	View > Other Windows > Containers
Package Manage Console	Package Manage Console is a Powershell console within visual studio used to interact with NuGet and automate visual studio.	View > Other Windows > Package Manage Console
Browser Link Dashboard	Browser Link Dashboard creates a communication channel between the development environment and one or more web browsers.	View > Other Windows > Browser Link Dashboard

IntelliCode Suggestions	IntelliCode Suggestions window will detect the repeated change you are making, and suggest you make that change in other places.	View > Other Windows > IntelliCode Suggestions
Document Outline	Document Outline window exposes a hierarchical view of elements residing on a window form or a web form.	View > Other Windows > Document Outline
C# interactive	C# interactive is a REPL(Read-Evaluate-Print-Loop) advanced editor in which we can test our code without compiling or running the complete code.	View > Other Windows > C# interactive

Solution and Project

Solutions: Solutions are simply a container for one or more related projects, along with build information and visual studio window settings.

↳ Solution File

Visual Studio uses two types (.sln and .suo) to store settings for solutions.

EXTENSION	NAME	DESCRIPTION
.sln	Visual Studio Solution	Organizes projects, project item and solution item in the solution.
.suo	Solution User Options	Stores user-level settings and customizations, such as breakpoints.

Projects: Projects contains all files that are compiled into an executable, library, or website.

↳ Project File

Visual Studio uses MsBuid to build each project in a solution, and each project contains a MsBuild project file.

EXTENSION	PROJECT TYPE
.csproj	C# project
.vbproj	Visual Studio project
.dbproj	database project

Code Editor Features

FEATURE	DESCRIPTION
Syntax Coloring	Some syntax elements of code and markup files are colored differently to distinguish them.
Error and Warning marks	When errors or warning occurs you may see different-colored wavy lines or light bulb in the code. ex. the red line denotes syntax error
Brace Matching	When the insertion point is placed on an open brace in a code file, both it and the closing brace are highlighted.
Structure Visualizer	Dotted lines connect matching braces in code files, making it easier to see opening and closing brace pairs.
Line Numbers	Line numbers can be displayed in the left margin of the code window.
Change Tracking	The color of the left margin allows you to keep track of the changes you have made in a file. ex. the left bar turns yellow if changes have been made but not saved.
Zoom	You can zoom in or out in any code window by pressing and holding the Ctrl key and moving the scroll wheel on the mouse.
Virtual Space	By default, lines in Visual Studio editors end after the last character, so that the Right Arrow key at the end of a line moves the cursor to the beginning of the next line.
Printing	You can use the options in the Print dialog box to include line numbers or hide collapsed regions of code when you print a file.

Format Document	Sets the proper indentation of lines of code and moves curly braces to separate lines in the document.
Tabify Selected Lines	Changes leading spaces to tabs where appropriate.
Delete Horizontal White Space	Deletes tabs or spaces at the end of the current line.
View White Space	Displays spaces as raised dots, and tabs as arrows. The end of a file is displayed as a rectangular glyph.
Comment / Uncomment Selection	Adds or Removes comment characters in the selection or the current line.

Shortcuts

Microsoft Visual Studio has many keyboard shortcuts, some of the popular and frequently used shortcuts are shown below.

COMMAND	SHORTCUT
Solution Explorer	Ctrl+Alt+L
Properties Window	F4
Toolbox	Ctrl+Alt+X
Server Explorer	Ctrl+Alt+S
Output Window	Ctrl+Alt+O
Error List	Ctrl+\
Breakpoints	Ctrl+Alt+B
Visual Studio Search	Ctrl+Q
Build Solution	Ctrl+Shift+B
Compile	Ctrl+F7
Comment	Ctrl+K+C
Save All	Ctrl+Shift+S
Uncomment	Ctrl+K+U
Format Code	Ctrl+K+D
Start debugging	F5
Stop debugging	Shift+F5

Windows App, Class Library

Windows App: Windows form is a UI framework for building Windows desktop apps.

- With Windows forms, we develop graphically rich apps that are easy to deploy, update and work while offline or while connected to the internet.

⟶ Create a Windows form application

1. Select Create a new project.
2. In the search for templates box, type WinForms, and press **Enter**.
3. In the code language dropdown, choose C#.
4. In the template list, select Windows Forms app(.NET) and then click **Next.**
5. In the configure your new project window set the project name to names and click **create**.
   (You can also save your project to a different folder by adjusting the location settings.)

Class Library: A class library defines types and methods that are called by an application.

- The extension of the class library file is **".dll (dynamic link library)"**.

⟶ Benefits of class library

- Increases performance of a program.
- Saves the programmer's time to write same code of a function for different parameters.
- Saves memory space.

⟶ Steps to create class library

1. Select Create a new project.
2. Choose the language as **"C#"** and choose **"Class Library (.NET Framework)"**.
3. Now enter **"Project Name"**, **"Location"**,**"Solution name"** & **"Framework"** and click **Create.**
4. Now Build Library using **(ctrl + Shift + B)** or **Build > Build Solution**.

⟶ Use Class Library in Windows Form application

1. Create a new Windows Form app.
2. Add a reference to the project.
   i. Click on **"References"** available in Solution Explorer.
   ii. Click on **"Add Reference".**
   iii. Click on **"Browse".**
   iv. Choose **.dll (Class Library File)** from your system.
   v. Click the **"OK"** button.
   vi. Add namespace with a **"using"** keyword at the top of the code.
   vii. Now Create an object of the class library in the code file and call methods of the library using an object.

Web Application

A web application is a client-side and server-side software application in which the client runs an app or request in a web browser.

⟶ How a web application works step-by-step

- The user accesses a web application via a web browser.
- The web server forwards the request to the web application server.
- The web application sever performs the requested task- such as querying the database or processing the data.
- Then it generates the results of the requested data.
- The web application server sends the results back to the web server.
- The web server delivers the requested information to the client and the information appears on the user's display.

⟶ Create a Web Application

1. Select Create a new project.
2. In the create a new project window, choose **C#** from the language list. Next, choose **Windows** from the platform list.
3. Then, choose **ASP.NET Core Web App** from the template list.
4. In the configure your project window, type or enter the **Name of web app** in the project name box. Then, choose **Next**.
5. In the additional information window, verify that **.NET Core 3.1** appears in the top drop-down menu.

(You can choose to enable docker support by checking the box. and you can also add authentication support.)

What is Namespace?

Namespaces are used to logically arrange classes, structs, interfaces, enums and delegates.

- .NET framework already contains number of standard namespaces like **System, System.Net, System.Io** etc.
- In addition to these standard namespaces the user can define their own namespaces.

⟶ Declaring a namespace

- The C# language provides a keyword **namespace** to create a user defined namespace.
- The general form of declaring a namespace is as follow.

Syntax:

```
namespace <namespace_name>
{
    //Classes, structs, enum
}
```

⟶ Accessing the members of Namespace

- There are two ways of accessing the members of namespace.

1. The members of a namespace are accessed using **dot(.)** operator.

Syntax:

```
[namespace_name].[member_name]
```

2. C# provides the keyword **"using"** which helps the user to avoid writing fully qualified names again and again.

- The user just has to mention the namespace name at the starting of the program and then can easily avoid the use of fully qualified names.

Syntax:

```
using [namespace_name][.][sub-namespace_name]
```

What is Class?

A **class** is a user-defined blueprint or prototype from which objects are created.

- A class combines the fields and methods into a single unit.
- In C# classes support polymorphism, inheritance, and also provide the concept of derived classes and base classes.

Declaration of class

- A class declaration contains only a keyword class, followed by an identifier(name) of the class.
- There are some optional attributes that can be used with class declaration according to the application requirement.
- Class declarations can include these components, in order:

- **Modifiers:** A class can be public or internal etc. By default modifier of class is internal.
- **Keyword class:** A class keyword is used to declare the type class.
- **Class Identifier:** The variable of type class is provided. The identifier(or name of class) should begin with an initial letter which should be capitalized by convention.
- **Base class or Superclass:** The name of the class's parent (superclass), if any, preceded by the : (colon). This is optional.
- **Interfaces:** A comma-separated list of interfaces implemented by the class, if any, preceded by the : (colon). A class can implement more than one interface. This is optional.
- **Body:** The class body is surrounded by { } (curly braces).

Syntax:

```
[access modifier]  [class]  [identifier]
```

Variable & Method Declaration

Variable: A variable is a name given to a memory location and all the operations done on the variable effects that memory location.

The following are naming conventions for declaring variables in C#.

- The variable name must be unique.
- Variable names can contain letters, digits, and the underscore(_) only.
- Variable name must start with a letter.
- Variable names are case-sensitive, num and Num are considered different names.
- Variable names cannot contain the reserved keywords. Must prefix '@' before keyword if want reserved keyword as identifiers.
- In C#, a variable contains a data value of a specific data type.

Syntax:

```
<data type> <variable_name> = value;
```

Method: A method is a group of statements that together perform a task.

- Every C# program has at least one class with a method named Main.
- The syntax for defining a method in C# is as follows.

Syntax:

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
        //Method Body
}
```

Following are the various elements of a method

- **Access Specifier** – This determines the visibility of a variable or a method from another class.

- **Return type** – A method may return a value. The return type is the data type of the value the method returns. If the method is not returning any values, then the return type is void.
- **Method name** – Method name is a unique identifier and it is case sensitive. It cannot be the same as any other identifier declared in the class.
- **Parameter list** – Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and a number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.
- **Method body** – This contains the set of instructions needed to complete the required activity.

Program Flow

Program Flow: C# code is generally processed sequentially from top to bottom.

- Unless a program flow statement changes the order and position of processing.
- Comments are not considered at the time of compilation.

⟶ **A C# program consist of the following parts**

- Namespace
- A Class
- Class Methods
- Class Attributes
- The Main Method
- Statements and Expressions
- Comments

Understanding Syntax

C# Uses below rules for correct syntax.

- C# is case-sensitive.
- All statements and expressions must end with a semicolon(;).

Here, we used the following code "Hello.cs" to understand syntax.

HelloWorld.cs

```
using System;

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

1. **using System** means that we can use classes from the **System** namespace.
2. A blank line. C# ignores white space. However, multiple lines make the code more readable.
3. A **namespace** is used to organize your code, and it is a container for classes and other namespaces.
4. The curly braces **{}** marks the beginning and the end of a block of code.
5. **Class** is a container for data and methods, which brings functionality to your program. Every line of code that runs in C# must be inside a class. In our example, we named the class **Program**.
6. Another thing that always appears in a C# program, is the **Main** method. Any code inside its curly brackets **{}** will be executed.
7. **Console** is a class of the **System** namespace, which has a **WriteLine()** method that is used to output/print text.

Understanding structure of solution

Solutions: A solution contains a collection of projects, along with information on dependencies between those projects.

- The projects themselves contain files.
- This structure is illustrated in Figure 1-1. You can have as many projects as you like in a solution, but there can be only one solution open at a time in a particular instance.

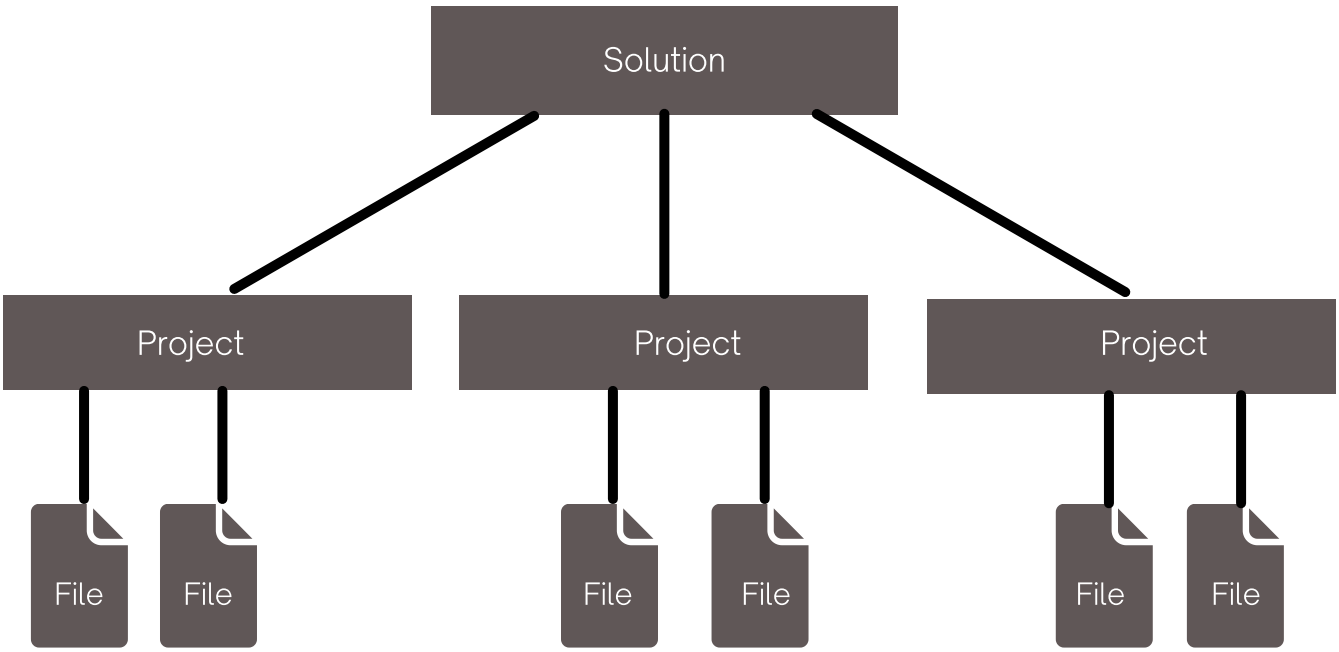

Figure 1-1. A solution, its projects, and their files

Understanding structure of Project

Windows App

All Universal Windows apps following MVVM(Model-View-ViewModel) pattern should have a similar directory structure.

Below are the all things that go into each folder.

- App.xaml (Extensible Application Markup Language file)
- **Controls:** Reusable UI controls (application independent views) without view models. Platform-specific Controls are added directly to the specific project and Shared controls are added in the Shared Project.
- **Strings:** Classes and resources for application localization
    1. **en-US:** Separate directory for every supported language.
        -> Resources.resw
- **Models:** Model and domain classes
- **ViewModels:** View models classes
    MainWindowModel.cs
    MyViewModel.cs
    Dialogs
        SelectItemDialogModel.cs
- **Converters:** This folder includes the Value Converters.
- **Themes:** It contains Theme Resources that is Resource Dictionary. Platform-specific resources are added directly to the specific project and Shared resources are added in the Shared Project.
- **Services:** This could include classes for web service calls, navigation service, etc.
- **Utils:** It includes all utility functions that would be used across the app. This may include AppCache, FileUtils, Constants, NetworkAvailability, GeoLocation, DataTemplateSelector, etc.
- **Views:** Contains the views. Platform-specific Views are added directly to the specific project and Shared Views are added in the Shared Project.

The name of a view should end with its type:

- **Window:** A non-modal window
- **Dialog:** A (modal) dialog window
- **Page:** A page view
- **View:** A view that is used as a subview in another view, page, window, or dialog

Web App

Web Application structure it defines two folders in solution as follows:

1. **Solution Items** – it contains a global.json file.
2. **Src** – it contains the web application.

It also creates some initial folders for your projects such as

- Controllers
- Migrations
- Models
- ViewModels
- Views, etc.

The wwwroot Folder

- It contains all the static files in your project; go into this folder.
- Code files are placed outside of this folder.
- This folder contains the clean and clear separation between the static files and code files.

The global.json File: This file contains the configuration of the project, created by the visual studio.

The appsettings.json File: This file contains information about the project, like connection string, etc.

The project.json File: It contains the information about the project – like which framework is used, server-side dependencies, commands, and scripts, etc.

Controllers: The Controllers folder contains class files for the controllers.

- A Controller handles users' requests and returns a response.

Models: The Models folder contains model class files.

- Typically model class includes public properties, which will be used by the application to hold and manipulate application data.

Views: The Views folder contains HTML files for the application.

- Typically view file is a .cshtml file where you write HTML and C# code.

Web API

Below are the files and folders that will be created when we create a new web API

Dependencies: The Dependencies contain all the packages and SDKs that are installed in this project.

Properties: The Properties Folder in ASP.NET Core Web Application by default contains one JSON file called launchsettings.json

- The launchsettings.json file contains some settings that are going to be used by .NET Core Framework when we run the application.
- launchSettings.json file is only used within the local development machine.

appsettings.json file: The appsettings.json file is used to store the configuration settings such as database connections strings, any application scope global variables, etc.

appsettings.Development.json: If you want to configure some settings based on the environments then you can do such settings in the appsettings.Development.json file.

Class Library

The class library contains a Dependencies folder and a code file.

Dependencies: The Dependencies contain all the packages and SDKs that are installed in this project.

Code file's extension is dependant on coding language. ex. for C# file's extension will be .cs

Familiar with different types of file extensions

FILE EXTENSION	CONTENTS
.htm, .html, .xsp, .asp, .htc, .hta, .xml	Common Web files.
.exe, .dll	Executable or dynamic-link library files.
.sln	The solution file.
.suo	The solution options file.
.txt	A text file, usually the "readme" file.
.csproj	The C# project file.
.cs	Main source code files for your application.
.asp	Active Server Page file.

Understanding data types & variables with conversion

Base Datatype

C# mainly categorized data types into two types

- Value types
- Reference types

1. **Value types** include simple types (such as int, float, bool, and char), enum types, struct types
2. **Reference types** include class types, interface types, and array types.

Datatype Conversion

Type conversion happens when we assign the value of one data type to another.

There are two types of type conversion.

- **Implicit / Automatic Type Conversion:** If the data types are compatible, then C# does Automatic Type Conversion.

  It happens When:
  1. The two data types are compatible.
  2. When we assign the value of a smaller data type to a bigger data type.

Following table shows the implicit types of conversion that is supported by C# :

CONVERT FROM DATA TYPE	CONVERT TO DATA TYPE
byte	short, int, long, float, double
short	int, long, float, double

int	long, float, double
long	float, double
float	double

- **Explicit Type conversion:** If the data types are not compatible, then they need to be converted explicitly which is known as Explicit Type conversion.

Syntax:

Base_datatype variable_name = (Target_datatype) variable_name;

- It is also possible to convert data types explicitly by using built-in methods, such as Convert.ToDouble(), Convert.ToString()

Boxing/Unboxing

- **Boxing and unboxing** in C# allow developers to convert .NET data types from value type to reference type and vice versa.
- Converting a value type to a reference type is called **Boxing.**
- Converting a reference type to a value type is called **Unboxing.**

Boxing Syntax:
object object_name = variable_name;

Unboxing Syntax:
datatype variable_name = (Target_datatype) object_name;

if-else, switch

- The if statement contains a boolean condition followed by a single or multi-line code block to be executed.
- At runtime, if a boolean condition evaluates to true, then the code block will be executed, otherwise not.

C# includes the following types of if statements:

- if statement
- else statement
- else-if statement
- Nested if Statements

if statement: Use the if statement to specify a block of C# code to be executed if a condition is True.

Syntax:

```
if (condition)
{
    // block of code to be executed if the condition is True
}
```

else statement: Use the else statement to specify a block of code to be executed if the condition is False.

Syntax:

```
if (condition)
{
    // block of code to be executed if the condition is True
}
else
{
    // block of code to be executed if the condition is False
}
```

else-if statement: Use the else if statement to specify a new condition if the first condition is False.

Syntax:

```
if (condition1)
{
   // block of code to be executed if the condition is True
}
else if (condition1)
{
    // block of code to be executed if the condition1 is false and condition2 is True
}
else
{
   // block of code to be executed if the condition1 is false and condition2 is False
}
```

Nested if Statements: if...else statement can exist within another if...else statement. Such statements are called nested if...else statements.

syntax:

```
if (condition)
{
     if (condition)
    {
          // code to be executed
    }
    else
    {
          // code to be executed
    }
}
else
{
      // code to be executed
}
```

Switch

Use the **switch** statement to select one of many code blocks to be executed.

This is how it works:

- The switch expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The default keyword is optional and specifies some code to run if there is no case match.
- When C# reaches a break keyword, it breaks out of the switch block.

Syntax:

```
switch (expression)
{
    case x:
       // code block
       break;
    case y:
       // code block
       break;
    default:
       // code block
       break;
}
```