**FRONT END –VISUAL STUDIO 2019**

**THE .NET FRAMEWORK**

The .NET Framework has two main parts:

1. The Common Language Runtime (CLR).

2. A hierarchical set of class libraries.

The CLR is described as the “execution engine” of .NET. It provides the environment within which programs run.

**Common Type System**

The CLR uses something called the Common Type System (CTS) to strictly enforce type-safety. This ensures that all classes are compatible with each other, by describing types in a common way. CTS define how types work within the runtime, which enables types in one language to interoperate with types in another language, including cross-language exception handling.

**Common Language Specification**

The CLR provides built-in support for language interoperability. To ensure that you can develop managed code that can be fully used by developers using any programming language, a set of language features and rules for using them called the Common Language Specification (CLS) has been defined. Components that follow these rules and expose only CLS features are considered CLS-compliant.

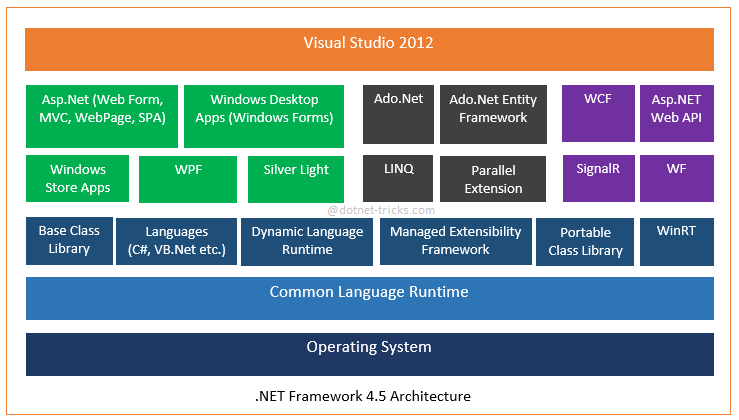
**THE CLASS LIBRARY**

.NET provides a single-rooted hierarchy of classes, containing over 7000 types. The root of the namespace is called System; this contains basic types like Byte, Double, Boolean, and String, as well as Object. All objects derive from System. Object. As well as objects, there are value types. Value types can be allocated on the stack, which can provide useful flexibility. There are also efficient means of converting value types to object types if and when necessary.

[**.NET FRAMEWORK PLATFORM ARCHITECTURE**](javascript:void(0))



**.NET FRAMEWORK 4.5 ARCHITECTURE**

****

.NET Framework 4.5 at its core consists of Common Language Runtime (CLR), Dynamic Language Runtime (DLR), Base Class Library (BCL), Portable Class Library, Parallel Extension and WinRT.

1. **Common Language Runtime**

This acts as the execution engine for the .NET Framework. All .NET programs executes under the supervision of CLR.

1. **Base Class Library**

This is a library of functionalities which are available to all languages using the .NET Framework. It consists of classes, interfaces of reusable types that integrates with CLR

1. **Portable Class Library**

The Portable Class Library project in Visual Studio 2019 allows you to develop and build managed assemblies that work on multiple .NET Framework platforms. Using a Portable Class Library project, you choose the platforms (such as Windows Phone and .NET for Windows Store apps) to target.

### Parallel Extension

This allows you to distribute your work code across multiple processors to take advantage

of the hardware.

1. **Dynamic Language Runtime**

This provides the runtime environment for dynamic languages like python etc. for executing under the full control of CLR.

1. **WinRT**

WinRT or Windows Runtime APIs provides the user interface elements for building Windows Store apps, and provides access to Windows 8 or Windows RT OS features. WinRT supports development in C and other managed languages C# and VB.NET, as well as JavaScript and TypeScript.

**Asp.Net**

**Asp.net** is an [open source](http://en.wikipedia.org/wiki/Open_source) [server-side](http://en.wikipedia.org/wiki/Server-side_scripting) [Web application framework](http://en.wikipedia.org/wiki/Web_application_framework) designed for [Web development](http://en.wikipedia.org/wiki/Web_development) to produce [dynamic Web pages](http://en.wikipedia.org/wiki/Dynamic_Web_pages). It was developed by [Microsoft](http://en.wikipedia.org/wiki/Microsoft) to allow [programmers](http://en.wikipedia.org/wiki/Programmer) to build dynamic [web sites](http://en.wikipedia.org/wiki/Web_site), [web applications](http://en.wikipedia.org/wiki/Web_application) and [web services](http://en.wikipedia.org/wiki/Web_service). It was first released in January 2002 with version 1.0 of the [.NET Framework](http://en.wikipedia.org/wiki/.NET_Framework), and is the successor to Microsoft's [Active Server Pages](http://en.wikipedia.org/wiki/Active_Server_Pages) (ASP) technology. ASP.NET is built on the [Common Language Runtime](http://en.wikipedia.org/wiki/Common_Language_Runtime) (CLR), allowing programmers to write ASP.NET code using any supported [.NET language](http://en.wikipedia.org/wiki/List_of_CLI_languages). The ASP.NET [SOAP](http://en.wikipedia.org/wiki/SOAP) extension framework allows ASP.NET components to process SOAP messages.

ASP.NET is in the process of being re-implemented as a modern and modular [web framework](http://en.wikipedia.org/wiki/Web_framework), together with other frameworks like [Entity Framework](http://en.wikipedia.org/wiki/Entity_Framework). The new framework will make use of the new open-source [.NET Compiler Platform](http://en.wikipedia.org/wiki/.NET_Compiler_Platform) (code-name "Roslyn") and be [cross platform](http://en.wikipedia.org/wiki/Cross_platform). [ASP.NET MVC](http://en.wikipedia.org/wiki/ASP.NET_MVC), ASP.NET Web API, and ASP.NET Web Pages (a platform using only [Razor](http://en.wikipedia.org/wiki/ASP.NET_Razor_view_engine) pages) will merge into a unified MVC 6. The project is called "ASP.NET vNext".

**Features of Asp.Net**

ASP.NET Web pages, known officially as Web Forms, are the main building blocks for application development in ASP.NET. Web forms are contained in files with a ".aspx" extension; these files typically contain static ([X](http://en.wikipedia.org/wiki/XHTML))[HTML](http://en.wikipedia.org/wiki/HTML" \o "HTML) mark-up, as well as markup defining server-side Web Controls and User Controls where the developers place all the rc content for the Web page. Additionally, dynamic code, which runs on the server, can be placed in a page within a block <% -- dynamic code -- %>, which is similar to other Web development technologies such as [PHP](http://en.wikipedia.org/wiki/PHP), [JSP](http://en.wikipedia.org/wiki/JavaServer_Pages), and [ASP](http://en.wikipedia.org/wiki/Active_Server_Pages). With [ASP.NET Framework 2.0](http://en.wikipedia.org/wiki/ASP.NET_3.5), Microsoft introduced a new *code-behind* model which allows static text to remain on the .aspx page, while dynamic code remains in an .aspx.vb or .aspx.cs or .aspx.fs file.

**Code-behind model**

[Microsoft](http://en.wikipedia.org/wiki/Microsoft) recommends dealing with dynamic program code by using the code-behind model, which places this code in a separate file or in a specially designated script tag. Code-behind files typically have names like *MyPage.aspx.cs* or *MyPage.aspx.vb* while the page file is *MyPage.aspx* (same filename as the page file (ASPX), but with the final extension denoting the page language). This practice is automatic in [Microsoft Visual Studio](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio) and other [IDEs](http://en.wikipedia.org/wiki/Integrated_development_environment). When using this style of programming, the developer writes code to respond to different events, like the page being loaded, or a control being clicked, rather than a procedural walkthrough of the document.

ASP.NET's code-behind model marks a departure from Classic ASP in that it encourages developers to build applications with [separation of presentation and content](http://en.wikipedia.org/wiki/Separation_of_presentation_and_content) in mind. In theory, this would allow a Web designer, for example, to focus on the design markup with less potential for disturbing the programming code that drives it. This is similar to the separation of the controller from the view in [Model–View–Controller](http://en.wikipedia.org/wiki/Model%E2%80%93View%E2%80%93Controller) (MVC) frameworks.

**Custom controls**

Programmers can also build *custom controls* for ASP.NET applications. Unlike user controls, these controls do not have an ASCX markup file, having all their code compiled into a [DLL](http://en.wikipedia.org/wiki/Dynamic-link_library)file. Such custom controls can be used across multiple Web applications and [Visual Studio](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio) projects.

**Rendering technique**

ASP.NET uses a "visited composites" rendering technique. During compilation, the template (.aspx) file is compiled into initialization code which builds a control tree (the composite) representing the original template. Literal text goes into instances of the Literal control class, and server controls are represented by instances of a specific control class. The initialization code is combined with user-written code and results in a class specific for the page. The page doubles as the root of the control tree.

Actual requests for the page are processed through a number of steps. First, during the initialization steps, an instance of the page class is created and the initialization code is executed. This produces the initial control tree which is now typically manipulated by the methods of the page in the following steps. As each node in the tree is a control represented as an instance of a class, the code may change the tree structure as well as manipulate the properties/methods of the individual nodes. Finally, during the rendering step a visitor is used to visit every node in the tree, asking each node to render itself using the methods of the visitor. The resulting HTML output is sent to the client.

After the request has been processed, the instance of the page class is discarded and with it the entire control tree. This is a source of confusion among novice ASP.NET programmers who rely on the class instance members that are lost with every page request/response cycle.

**State management**

ASP.NET applications are hosted by a [Web server](http://en.wikipedia.org/wiki/Web_server) and are accessed using the [stateless](http://en.wikipedia.org/wiki/Stateless_server) [HTTP](http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol) protocol. As such, if an application uses stateful interaction, it has to implement [state management](http://en.wikipedia.org/wiki/State_management) on its own. ASP.NET provides various functions for state management. Conceptually, Microsoft treats "state" as [GUI](http://en.wikipedia.org/wiki/Graphical_user_interface) state. Problems may arise if an application needs to keep track of "data state"; for example, a [finite-state machine](http://en.wikipedia.org/wiki/Finite-state_machine) which may be in a transient state between requests ([lazy evaluation](http://en.wikipedia.org/wiki/Lazy_evaluation)) or which takes a long time to initialize. State management in ASP.NET pages with authentication can make [Web scraping](http://en.wikipedia.org/wiki/Web_scraping) difficult or impossible.

**Template engine**

When first released, ASP.NET lacked a [template engine](http://en.wikipedia.org/wiki/Template_engine_%28web%29). Because the [.NET Framework](http://en.wikipedia.org/wiki/.NET_Framework) is [object-oriented](http://en.wikipedia.org/wiki/Object-oriented) and allows for [inheritance](http://en.wikipedia.org/wiki/Inheritance_%28computer_science%29), many developers would define a new base class that inherits from "System.Web.UI.Page", write [methods](http://en.wikipedia.org/wiki/Method_%28computer_science%29) there that render HTML, and then make the pages in their application inherit from this new class. While this allows for common elements to be reused across a site, it adds complexity and mixes [source code](http://en.wikipedia.org/wiki/Source_code) with [markup](http://en.wikipedia.org/wiki/Markup_language). Furthermore, this method can only be visually tested by running the application – not while designing it. Other developers have used [include files](http://en.wikipedia.org/wiki/Header_file) and other tricks to avoid having to implement the same navigation and other elements in every page.

ASP.NET 2.0 introduced the concept of "master pages", which allow for [template](http://en.wikipedia.org/wiki/Web_template)-based page development. A Web application can have one or more master pages, which, beginning with ASP.NET 2.0, can be nested. Master templates have place-holder controls, called ContentPlaceHolders to denote where the dynamic content goes, as well as [HTML](http://en.wikipedia.org/wiki/HTML) and [JavaScript](http://en.wikipedia.org/wiki/JavaScript) shared across child pages.

Child pages use those ContentPlaceHolder controls, which must be mapped to the place-holder of the master page that the content page is populating. The rest of the page is defined by the shared parts of the master page, much like a [mail merge](http://en.wikipedia.org/wiki/Mail_merge) in a [word processor](http://en.wikipedia.org/wiki/Word_processor). All markup and [server](http://en.wikipedia.org/wiki/Server_%28computing%29) controls in the content page must be placed within the ContentPlaceHolder control.

**Directory structure**

In general, the ASP.NET directory structure can be determined by the developer's preferences. Apart from a few reserved directory names, the site can span any number of directories. The structure is typically reflected directly in the URLs. Although ASP.NET provides means for intercepting the request at any point during processing, the developer is not forced to funnel requests through a central application or front controller.

**C#**

C#is a [multi-paradigm programming language](http://en.wikipedia.org/wiki/Multi-paradigm_programming_language) encompassing [strong typing](http://en.wikipedia.org/wiki/Strong_typing), [imperative](http://en.wikipedia.org/wiki/Imperative_programming), [declarative](http://en.wikipedia.org/wiki/Declarative_programming), [functional](http://en.wikipedia.org/wiki/Functional_programming), [generic](http://en.wikipedia.org/wiki/Generic_programming), [object-oriented](http://en.wikipedia.org/wiki/Object-oriented_programming) ([class](http://en.wikipedia.org/wiki/Class_%28computer_science%29)-based), and [component-oriented](http://en.wikipedia.org/wiki/Component-based_software_engineering) programming disciplines. It was developed by [Microsoft](http://en.wikipedia.org/wiki/Microsoft) within its [.NET](http://en.wikipedia.org/wiki/.NET_Framework) initiative and later approved as a standard by [Ecma](http://en.wikipedia.org/wiki/Ecma_International) (ECMA-334) and [ISO](http://en.wikipedia.org/wiki/International_Organization_for_Standardization) (ISO/IEC 23270:2006). C# is one of the programming languages designed for the [Common Language Infrastructure](http://en.wikipedia.org/wiki/Common_Language_Infrastructure).

C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Its development team is led by [Anders Hejlsberg](http://en.wikipedia.org/wiki/Anders_Hejlsberg). The most recent version is C# 5.0, which was released on August 15, 2019.

**Features of C#**

**Portability**

By design, C# is the programming language that most directly reflects the underlying [Common Language Infrastructure](http://en.wikipedia.org/wiki/Common_Language_Infrastructure) (CLI). Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate [Common Intermediate Language](http://en.wikipedia.org/wiki/Common_Intermediate_Language) (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or [Fortran](http://en.wikipedia.org/wiki/Fortran).

**Meta programming**

Meta programming via C# attributes is part of the language. Many of these attributes duplicate the functionality of GCC's and Visual C++'s platform-dependent pre-processor directives.

**Methods and functions**

Like C++, and unlike Java, C# programmers must use the keyword virtual to allow methods to be overridden by subclasses. *Extension methods* in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives. The type dynamic allows for run-time method binding, allowing for JavaScript-like method calls and run-time object composition.

C# has support for strongly-typed function pointers via the keyword delegate. Like the Qt framework's pseudo-C++ *signal* and *slot*, C# has semantics specifically surrounding publish-subscribe style events, though C# uses delegates to do so.

**Property**

C# provides [properties](http://en.wikipedia.org/wiki/Property_%28programming%29) as [syntactic sugar](http://en.wikipedia.org/wiki/Syntactic_sugar) for a common pattern in which a pair of methods, [accessor (getter) and mutator (setter)](http://en.wikipedia.org/wiki/Mutator_method) encapsulate operations on a single [attribute](http://en.wikipedia.org/wiki/Attribute_%28computing%29) of a class. No redundant method signatures for the getter/setter implementations need be written, and the property may be accessed using attribute syntax rather than more verbose method calls.

**Namespace**

A C# namespace provides the same level of code isolation as a Java package or a C++ namespace, with very similar rules and features to a package.

**Memory access**

In C#, memory address pointers can only be used within blocks specifically marked as unsafe, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined [null](http://en.wikipedia.org/wiki/Nullable_type) value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System.IntPtr type, but it cannot dereference them.

Managed memory cannot be explicitly freed; instead, it is automatically garbage collected. Garbage collection addresses the problem of [memory leaks](http://en.wikipedia.org/wiki/Memory_leak) by freeing the programmer of responsibility for releasing memory that is no longer needed.

**Exception**

[Checked exceptions](http://en.wikipedia.org/wiki/Checked_exceptions) are not present in C# (in contrast to Java). This has been a conscious decision based on the issues of scalability and versionability.

**Polymorphism**

Unlike [C++](http://en.wikipedia.org/wiki/Cplusplus), [multiple inheritance](http://en.wikipedia.org/wiki/Multiple_inheritance) is not supported by C#, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI. When implementing multiple interfaces that contain a method with the same signature, C# allows the programmer to implement each method depending on which interface that method is being called through, or, like Java, allows the programmer to implement the method once and have that be the single invocation on a call through any of the class's interfaces.

**Functional programming**

Though primarily an imperative language, C# 2.0 offered limited support for functional programming through [first-class functions](http://en.wikipedia.org/wiki/First-class_functions) and closures in the form of anonymous delegates. C# 3.0 expanded support for functional programming with the introduction of a light weight syntax for lambda expressions, extension methods (an affordance for modules), and a [list comprehension](http://en.wikipedia.org/wiki/List_comprehension) syntax in the form of a "query comprehension" language.

**Microsoft Visual Studio**

**Microsoft Visual Studio** is an [integrated development environment](http://en.wikipedia.org/wiki/Integrated_development_environment) (IDE) from [Microsoft](http://en.wikipedia.org/wiki/Microsoft). It is used to develop [computer programs](http://en.wikipedia.org/wiki/Computer_program) for [Microsoft Windows](http://en.wikipedia.org/wiki/Microsoft_Windows), as well as [web sites](http://en.wikipedia.org/wiki/Web_site), [web applications](http://en.wikipedia.org/wiki/Web_application) and [web services](http://en.wikipedia.org/wiki/Web_service). Visual Studio uses Microsoft software development platforms such as [Windows API](http://en.wikipedia.org/wiki/Windows_API), [Windows Forms](http://en.wikipedia.org/wiki/Windows_Forms), [Windows Presentation Foundation](http://en.wikipedia.org/wiki/Windows_Presentation_Foundation), [Windows Store](http://en.wikipedia.org/wiki/Windows_Store) and [Microsoft Silverlight](http://en.wikipedia.org/wiki/Microsoft_Silverlight). It can produce both [native code](http://en.wikipedia.org/wiki/Native_code) and [managed code](http://en.wikipedia.org/wiki/Managed_code).

Visual Studio includes a [code editor](http://en.wikipedia.org/wiki/Code_editor) supporting [IntelliSense](http://en.wikipedia.org/wiki/IntelliSense) (the [code completion](http://en.wikipedia.org/wiki/Code_completion) component) as well as [code refactoring](http://en.wikipedia.org/wiki/Code_refactoring). The integrated [debugger](http://en.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building [GUI](http://en.wikipedia.org/wiki/GUI) applications, [web designer](http://en.wikipedia.org/wiki/Web_designer), [class](http://en.wikipedia.org/wiki/Class_%28computing%29) designer, and [database schema](http://en.wikipedia.org/wiki/Database_schema) designer. It accepts plug-ins that enhance the functionality at almost every level—including adding support for [source-control](http://en.wikipedia.org/wiki/Source_control) systems (like [Subversion](http://en.wikipedia.org/wiki/Subversion_%28software%29)) and adding new toolsets like editors and visual designers for [domain-specific languages](http://en.wikipedia.org/wiki/Domain-specific_language) or toolsets for other aspects of the [software development lifecycle](http://en.wikipedia.org/wiki/Software_development_lifecycle) (like the [Team Foundation Server](http://en.wikipedia.org/wiki/Team_Foundation_Server) client: Team Explorer).

Visual Studio supports different [programming languages](http://en.wikipedia.org/wiki/Programming_language) and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include [C](http://en.wikipedia.org/wiki/C_%28programming_language%29),[C++](http://en.wikipedia.org/wiki/C%2B%2B) and [C++/CLI](http://en.wikipedia.org/wiki/C%2B%2B/CLI) (via [Visual C++](http://en.wikipedia.org/wiki/Visual_C%2B%2B)), [VB.NET](http://en.wikipedia.org/wiki/VB.NET) (via [Visual Basic .NET](http://en.wikipedia.org/wiki/Visual_Basic_.NET)), [C#](http://en.wikipedia.org/wiki/C_Sharp_%28programming_language%29) (via [Visual C#](http://en.wikipedia.org/wiki/Visual_C_Sharp)), and [F#](http://en.wikipedia.org/wiki/F_Sharp_%28programming_language%29) (as of Visual Studio 2010). Support for other languages such as [M](http://en.wikipedia.org/wiki/MUMPS), [Python](http://en.wikipedia.org/wiki/IronPython), and [Ruby](http://en.wikipedia.org/wiki/IronRuby) among others is available via language services installed separately. It also supports [XML](http://en.wikipedia.org/wiki/XML)/[XSLT](http://en.wikipedia.org/wiki/XSLT), [HTML](http://en.wikipedia.org/wiki/HTML)/[XHTML](http://en.wikipedia.org/wiki/XHTML), [JavaScript](http://en.wikipedia.org/wiki/JavaScript) and [CSS](http://en.wikipedia.org/wiki/Cascading_Style_Sheets). Java (and J#) were supported in the past.

**Visual Studio 2019**

Visual Studio 2019 was announced on 1 August 2019 and the official launch event was held on 12 September 2019.Unlike prior versions, Visual Studio 2019 cannot record and play [macros](https://en.wikipedia.org/wiki/DTE80) and the macro editor has been removed. New features include support for [WinRT](https://en.wikipedia.org/wiki/WinRT) and [C++/CX](https://en.wikipedia.org/wiki/C%2B%2B/CX) (Component Extensions) and [C++ AMP](https://en.wikipedia.org/wiki/C%2B%2B_AMP) ([GPGPU](https://en.wikipedia.org/wiki/GPGPU) programming) Semantic Colorization. The source code of Visual Studio 2019 consists of approximately 50 million lines of code.

**Features of Visual Studio 2019**

* **Semantic colorization**: Improved syntax coloring, various user-defined or default colors for C++ syntax such as macros, enumerations, type names and functions.
* **Reference highlighting**: Selection of a symbol highlights all of the references to that symbol within scope.
* **New Solution Explorer**: The new Solution Explorer allows for visualization of class and file hierarchies within a solution/project. It can search for calls to functions and uses of classes.
* **Automatic display of IntelliSense list**: IntelliSense is automatically displayed whilst typing code, as opposed to previous versions where it had to be explicitly invoked through use of certain operators (i.e. the scope operator (::)) or shortcut keys (*Ctrl-Space* or *Ctrl-J*).
* **Member list filtering**: IntelliSense use fuzzy logic to determine which functions/variables/types to display in the list.
* **Code snippets**: Code snippets are included in IntelliSense to automatically generate relevant code based on the user's parameters, custom code snippets can be created.

**Interface backlash**

In Visual Studio 2019 RC, a major change to the interface is the use of all-caps menu bar, as part of the campaign to keep Visual Studio consistent with the direction of other Microsoft user interfaces, and to provide added structure to the top menu bar area. ]The redesign was criticized for being hard to read, and going against the trends started by developers to use [CamelCase](https://en.wikipedia.org/wiki/CamelCase" \o "CamelCase) to make words stand out better. Some speculated that the root cause of the redesign was to incorporate the simplistic look and feel of Metro apps. However, there exists a [Windows Registry](https://en.wikipedia.org/wiki/Windows_Registry) option to allow users to disable the all-caps interface.

### Revamped UI

### The monochrome scheme in earlier versions of VS 2019 was met with a bitter backlash as developers took an instant dislike to it. In fairness, the idea was to deemphasize the UI and make the code stand out more. However, that was a bad idea. VS 2019 RC reintroduces color, and once again you can tell the difference between the icons on the toolbars.

### Ability to Build Metro-style Applications

### VS 2019 build Metro apps using HTML, JavaScript or XAML and VB, C# or C++, and it includes a new set of Metro-style app templates. It also supports building Windows Phone applications.