



Unit 1 - The .NET Framework, C# Language Basics, ASP.NET

▼ Syllabus

- The .NET Framework: .NET Languages, Common Language Runtime, .NET Class Library
- C# Language Basics: Comments, Variables and Data Types, Variable Operations, Object-Based Manipulation, Conditional Logic, Loops, Methods, Classes, Value Types and Reference Types, Namespaces and Assemblies, Inheritance, Static Members, Casting Objects, Partial Classes
- ASP.NET - Writing Code - Code-Behind Class, Adding Event Handlers, Anatomy of an ASP.NET Application - ASP.NET File Types, ASP.NET Web Folders
- HTML Server Controls - View State, HTML Control Classes, HTML Control Events, HtmlControl Base Class, HtmlContainerControl Class, HtmlInputControl Class, Page Class, global.asax File

ASP.NET

- Microsoft created higher-level development platforms - first ASP and then ASP.NET.
- These technologies allow developers to program dynamic web pages without worrying about the low-level implementation details.
- ASP.NET is stuffed full of sophisticated features, including tools for
 - implementing security
 - managing data
 - storing user-specific information, and much more.
- ASP.NET is designed first and foremost as a server-side programming platform.
- All ASP.NET code runs on the web server.
- When the ASP.NET code finishes running, the web server sends the user the final result - an ordinary HTML page that can be viewed in any browser.

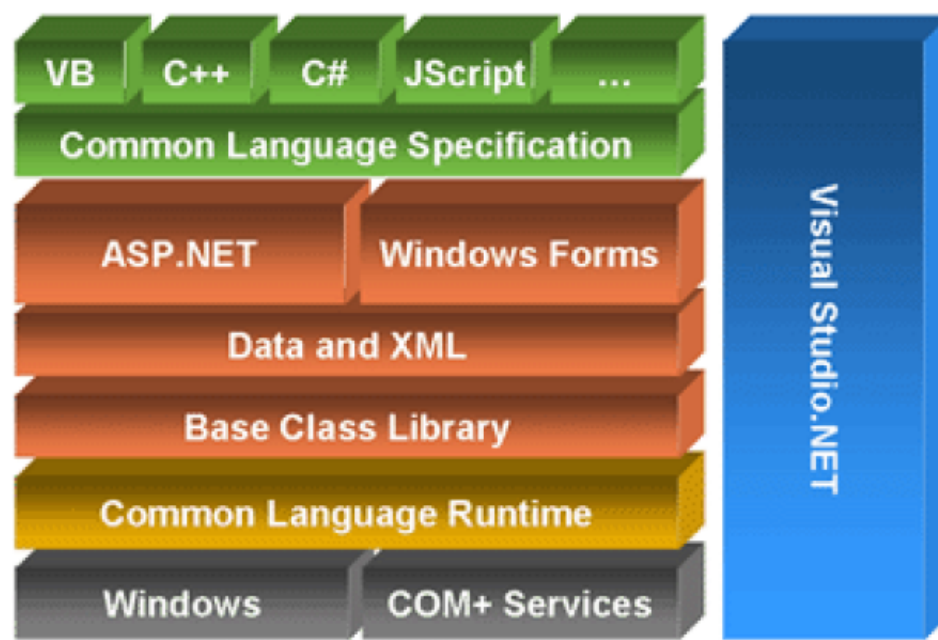
.NET Framework



.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows.

The .NET Framework is really a cluster of several technologies:

- The .NET languages
- The Common Language Runtime (CLR)
- The .NET FCL (Framework Class Library)
- ASP.NET
- Visual Studio



.NET Languages

- These include Visual Basic, C#, F#, and C++, although third-party developers have created hundreds more.
- The .NET Framework ships with two core languages that are commonly used for building ASP.NET applications: **C#** and **VB**.
- Both VB and C# use the .NET class library and are supported by the CLR.
- Almost any block of C# code can be translated, line by line, into an equivalent block of VB code (and vice versa).
- A developer who has learned one .NET language can move quickly and efficiently to another.
- In short, both VB and C# are elegant, modern languages that are ideal for creating the next generation of web applications.

Intermediate Language



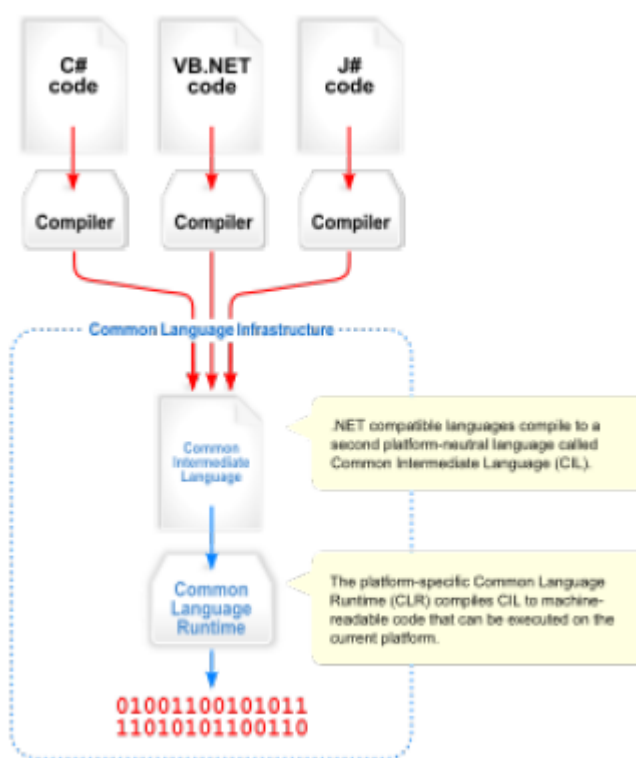
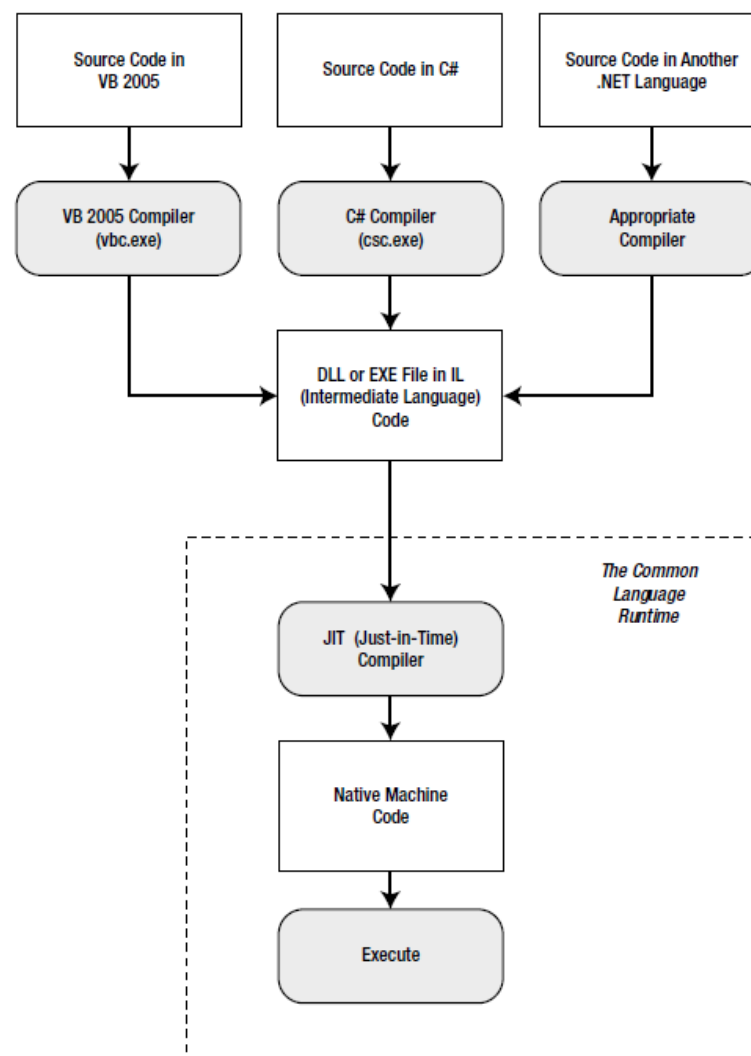
All the .NET languages are compiled into another lower-level language before the code is executed. This lower-level language is the **Common Intermediate Language** (CIL or IL).

- The CLR, the engine of .NET, uses only IL code. Because all .NET languages are based on IL, they all have profound similarities
- This is the reason that the VB and C# languages provide essentially the same features and performance.


Common Language Specification

- The .NET Framework formalizes the language compatibility with something called the **Common Language Specification (CLS)**.
- It is a document that says how computer programs can be turned into CIL code.
- When several languages use the same bytecode, different parts of a program can be written in different languages.
- Essentially, the CLS is a contract that, if respected, guarantees that a component written in one .NET language can be used in all the others.
- Microsoft has defined CLS which are nothing but guidelines for languages to follow so that it can communicate with other .NET languages in a seamless manner.
- **Common Type System**
 - One part of the CLS is the **common type system (CTS)**, which defines the rules for data types such as strings, numbers, and arrays that are shared in all .NET languages.

- CTS is a Microsoft-designed ECMA standard that sets the ground rules that all .NET languages must follow when dealing with data.
- The CLS also defines object-oriented ingredients such as classes, methods, events, and quite a bit more.



Common Language Runtime (CLR)

 The CLR is the engine that supports all the .NET languages.

- All .NET code runs inside the CLR. This is true whether you're running a Windows application or a web service.
- For example, when a client requests an ASP.NET web page, the ASP.NET service runs inside the CLR environment, executes your code, and creates a final HTML page to send to the client.

- Not only does the CLR execute code, it also provides a whole set of related services such as code verification, optimization, and object management.
- The CLR runs only IL code, which means it has no idea which .NET language you originally used.
- The CLR takes the IL code and transforms it to native machine language code that's appropriate for the current platform.
- **The implications of the CLR are wide-ranging:**
 - **Deep language integration:** VB and C#, like all .NET languages, compile to IL. The CLR makes no distinction between different languages. This is far more than mere language compatibility; it's language integration.
 - **Side-by-side execution:** The CLR also has the ability to load more than one version of a component at a time. In other words, you can update a component many times, and the correct version will be loaded and used for each application. As a side effect, multiple versions of the .NET Framework can be installed.
 - **Fewer errors:** Whole categories of errors are impossible with the CLR. For example, the CLR prevents many memory mistakes that are possible with lower-level languages such as C++.

Drawbacks of CLR:

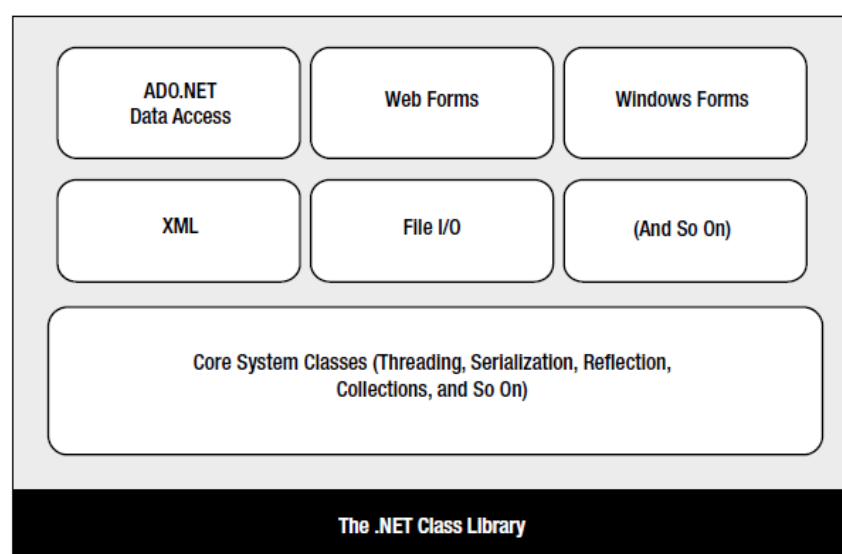
- **Performance:** A typical ASP.NET application is extremely fast, because ASP.NET code is compiled to machine code before it's executed. However, processor-crunching algorithms still can't match the blinding speed of well-written C++ code, because the CLR imposes some additional overhead (e.g. real-time games).
- **Code transparency:** IL is much easier to disassemble, meaning that if you distribute a compiled application, others may have an easier time determining how your code works. This isn't much of an issue for applications which aren't distributed but are hosted on a secure web server.
- **Questionable cross-platform support:** No one is entirely sure whether .NET will ever be adopted for use on other operating systems and platforms. However, .NET will probably never have the wide reach of a language such as Java because it incorporates too many different platform and OS-specific technologies and features.

.NET Class Library



The .NET class library is a giant repository of classes that provide prefabricated functionality for everything from reading an XML file to sending an e-mail message.

- Any .NET language can use the .NET class library's features by interacting with the right objects.
- Some parts of the class library are meant for desktop applications with the Windows interface and some are targeted directly at web development.
- There are some classes that include the base set of classes that define common variable types and the classes for data access.
- Microsoft's philosophy is that it will provide the tedious infrastructure so that application developers need only to write business-specific code.



C# Language Basics

- Case-sensitive language
- **Comments**
 - Comments are lines of descriptive text that are ignored by the compiler. C# provides two basic types of comments: single-line and multiple-line.
 - C# also includes an XML-based commenting syntax that you can use to describe your code in a standardized way.

```
// A single-line C# comment.

/* A multiple-line
C# comment. */

/// <summary>
/// This application provides web pages
/// for my e-commerce site.
/// </summary>
```

- **Statement Termination**
 - C# uses a semicolon `;` as a statement-termination character.
 - Every statement in C# code must end with this semicolon, except when you’re defining a block structure.
- **Blocks**
 - The C# language relies heavily on curly braces.
 - Curly braces group multiple code statements together (for block structures like loops, functions, etc.).
- **Variables and Data Types**
 - You keep track of data in C# by using variables.
 - Variables can store numbers, text, dates, and times, and they can even point to full-fledged objects.
 - When you declare a variable, you give it a name and specify the type of data it will store.
 - Every .NET language uses the same variable data types, because they all adhere to the CTS.

💡 Different languages may provide slightly different names, but the CLR makes no distinction - in fact, they are just two different names for the same base data type.
e.g. VB Integer = C# int → System.Int32

C# Name	VB Name	.NET Type Name	Contains
byte	Byte	Byte	An integer from 0 to 255.
short	Short	Int16	An integer from −32,768 to 32,767.
int	Integer	Int32	An integer from −2,147,483,648 to 2,147,483,647.
long	Long	Int64	An integer from about −9.2e18 to 9.2e18.
float	Single	Single	A single-precision floating-point number from approximately −3.4e38 to 3.4e38 (for big numbers) or −1.5e-45 to 1.5e-45 (for small fractional numbers).
double	Double	Double	A double-precision floating-point number from approximately −1.8e308 to 1.8e308 (for big numbers) or −5.0e-324 to 5.0e-324 (for small fractional numbers).
decimal	Decimal	Decimal	A 128-bit fixed-point fractional number that supports up to 28 significant digits.
char	Char	Char	A single Unicode character.
string	String	String	A variable-length series of Unicode characters.
bool	Boolean	Boolean	A true or false value.

C# Name	VB Name	.NET Type Name	Contains
-	Date	DateTime	Represents any date and time from 12:00:00 AM, January 1 of the year 1 in the Gregorian calendar, to 11:59:59 PM, December 31 of the year 9999.
-	-	TimeSpan	Represents a period of time, as in ten seconds or three days. The smallest possible interval is 1 tick (100 nanoseconds).
object	Object	Object	The ultimate base class of all .NET types. Can contain any data type or object.

- **Assignment and Initializers**

- After you've declared your variables, you can freely assign values to them (as long as these values have the correct data type).
- You can also assign a value to a variable in the same line that you declare it.
- C# safeguards you from errors by restricting you from using uninitialized variables.
- C# defines a few special characters that you can append to literal values to indicate their data type so that no conversion will be required: M (decimal), D (double), F (float), L (long).

- **Strings and Escaped Characters**

- C# interprets any embedded backslash `\` as the start of a special character sequence.
- `\"` (double quote)
- `\n` (new line)
- `\t` (horizontal tab)
- `\\` (backward slash)
- You can turn off C# escaping by preceding a string with an `@` symbol.

- **Arrays**

- Arrays allow you to store a series of values that have the same data type.
- Each individual value in the array is accessed by using one or more index numbers.
- Typically, arrays are laid out contiguously in memory.
- All arrays start at a fixed lower bound of 0. This rule has no exceptions.
- When you create an array in C#, you specify the number of elements.

Member	Description
<code>Length</code>	Returns an integer that represents the total number of elements in all dimensions of an array.
<code>GetLowerBound()</code> <code>GetUpperBound()</code>	Determines the dimensions of an array.
<code>Clear()</code>	Empties part or all of an array's contents, depending on the index values that you supply.
<code>IndexOf()</code> <code>LastIndexOf()</code>	Searches a one-dimensional array for a specified value and returns the index number
<code>Sort()</code>	Sorts a one-dimensional array made up of comparable data such as strings or numbers.
<code>Reverse()</code>	Reverses a one-dimensional array so that its elements are backward, from last to first.

- **ArrayList**

- C# arrays do not support redimensioning (after you create an array, you can't change its size).
- One of the simplest collection classes that .NET offers is the ArrayList, which supports any type of object and always allows dynamic resizing.

- **Enumerations**

- An enumeration is a group of related constants, each of which is given a descriptive name.
- Each value in an enumeration corresponds to a preset integer.

- **Type Conversions**

- Converting information from one data type to another is called type conversion.
- Two types: Widening and Narrowing
- Widening conversions always succeed.
- Narrowing conversions may or may not succeed, depending on the data.
- To convert a variable, you simply need to specify the type in parentheses before the expression you're converting (called casting).

```
count16 = (short)count32
```

• **Strings**

Member	Description
Length	Returns the number of characters in the string (as an integer).
ToUpper() ToLower()	Returns a copy of the string with all the characters changed to uppercase or lowercase characters.
Trim() TrimEnd() TrimStart()	Removes spaces (or the characters you specify) from either end (or both ends) of a string.
PadLeft() PadRight()	Adds the specified character to the appropriate side of a string as many times as necessary to make the total length of the string equal to the number you specify.
Insert()	Puts another string inside a string at a specified (zero-based) index position.
Remove()	Removes a specified number of characters from a specified position.
Replace()	Replaces a specified substring with another string.
Substring()	Extracts a portion of a string of the specified length at the specified location (as a new string).
StartsWith() EndsWith()	Determines whether a string starts or ends with a specified substring.
IndexOf() LastIndexOf()	Finds the zero-based position of a substring in a string. This returns only the first match and can start at the end or beginning.
Split()	Divides a string into an array of substrings delimited by a specific substring
Join()	Fuses an array of strings into a new string. You must also specify the separator that will be inserted between each element.

• **DateTime**

Member	Description
Now	Gets the current date and time.
Today	Gets the current date and leaves time set to 00:00:00.
Year Date Month Hour Minute Second Millisecond	Returns one part of the DateTime object as an integer.
DayOfWeek	Returns an enumerated value that indicates the day of the week for this DateTime, using the DayOfWeek enumeration.
Add() Subtract()	Adds or subtracts a TimeSpan from the DateTime. For convenience, these operations are mapped to the + and - operators.
AddYears() AddMonths() AddDays() AddHours() AddMinutes() AddSeconds() AddMilliseconds()	Adds an integer that represents a number of years, months, and so on, and returns a new DateTime.
DaysInMonth()	Returns the number of days in the specified month in the specified year.
IsLeapYear()	Returns true or false depending on whether the specified year is a leap year.
ToString()	Returns a string representation of the current DateTime object.

• **TimeSpan**

Member	Description
<code>Days</code> <code>Hours</code> <code>Minutes</code> <code>Seconds</code> <code>Milliseconds</code>	Returns one component of the current TimeSpan.
<code>TotalDays</code> <code>TotalHours</code> <code>TotalMinutes</code> <code>TotalSeconds</code> <code>TotalMilliseconds</code>	Returns the total value of the current TimeSpan as a number of days, hours, minutes, and so on.
<code>Add()</code> <code>Subtract()</code>	Combines TimeSpan objects together. For convenience, these operations are mapped to the <code>+</code> and <code>-</code> operators.
<code>FromDays()</code> <code>FromHours()</code> <code>FromMinutes()</code> <code>FromSeconds()</code> <code>FromMilliseconds()</code>	Allows you to quickly create a new TimeSpan.
<code>ToString()</code>	Returns a string representation of the current TimeSpan object.

- **Not including conditional statements, loops, methods**

Classes



Classes are the code definitions for objects.

- Classes interact with each other with the help of three key ingredients:
- **Properties:** Properties allow you to access an object’s data. Some properties are read-only, so they cannot be modified, while others can be changed.
- **Methods:** Methods allow you to perform an action on an object. Unlike properties, methods are used for actions that perform a distinct task or may change the object’s state significantly.
- **Events:** Events provide notification that something has happened.
- In ASP.NET, you’ll create your own custom classes to represent individual web pages.
- You can use some class members without creating an object first. These are called **static members**, and they’re accessed by class name.
- Property accessors, like any other public piece of a class, should start with an initial capital.

```
public class Product
{
    private string name;
    private decimal price;
    private string imageUrl;
    public string Name
    {
        get{return name;}
        set{name = value;}
    }
}
```

Keyword	Accessibility
public	Can be accessed by any class
private	Can be accessed only by members inside the current class
internal	Can be accessed by members in any of the classes in the current assembly (the file with the compiled code)
protected	Can be accessed by members in the current class or in any class that inherits from this class
protected internal	Can be accessed by members in the current application (as with internal) and by the members in any class that inherits from this class.

