.NET Framework 4.6 and C# 7.0

Lesson 01 : Introduction to .NET Framework 4.6

Capgemini

Lesson Objectives

- In this lesson, you will learn:
  - The .NET Framework
  - Features of Common Language Runtime
  - .NET Framework class Library
  - Managed Execution process
  - Understand the .NET Framework 4.5 stack
  - Know the new productivity enhancements in VS2012

1.1: What is .NET Platform ?

Introduction

- The .NET Platform is used to develop enterprise applications based on industry standards.
- The .NET platform was introduced to offer a much more powerful, more flexible, and simpler programming model than COM
- .NET Framework is a fully managed, protected, simplified, feature rich application execution environment
  - It is OS independent and hardware independent
  - Extensively uses Industry standards like HTTP, SOAP, XML, XSD.
  - Easily maintainable due to simplified deployment and version management
  - A platform to build Web services, Multi Threaded Applications, Windows Services, Rich Internet Applications as well as Mobile Application.
  - Provides seamless integration to a wide variety of languages.

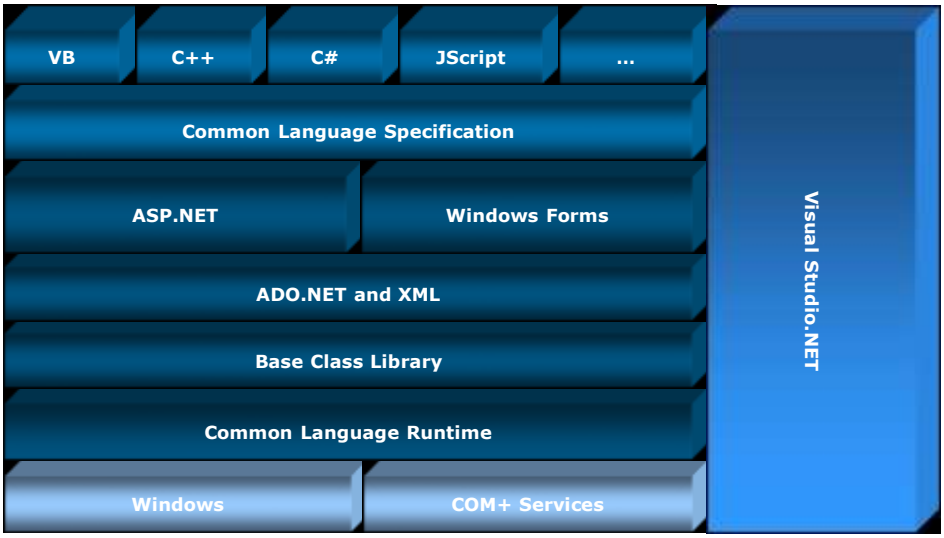
What is .NET Platform?

According to the Microsoft, .NET Framework is:

A platform for building, deploying, and running web services and applications. The .NET Framework provides a highly productive, standards-based, multi-language environment for integrating existing investments with next-generation applications and services as well as the ability to solve the challenges of deployment and operation of Internet-scale applications.

1.4: .NET Framework, Languages, and Tools

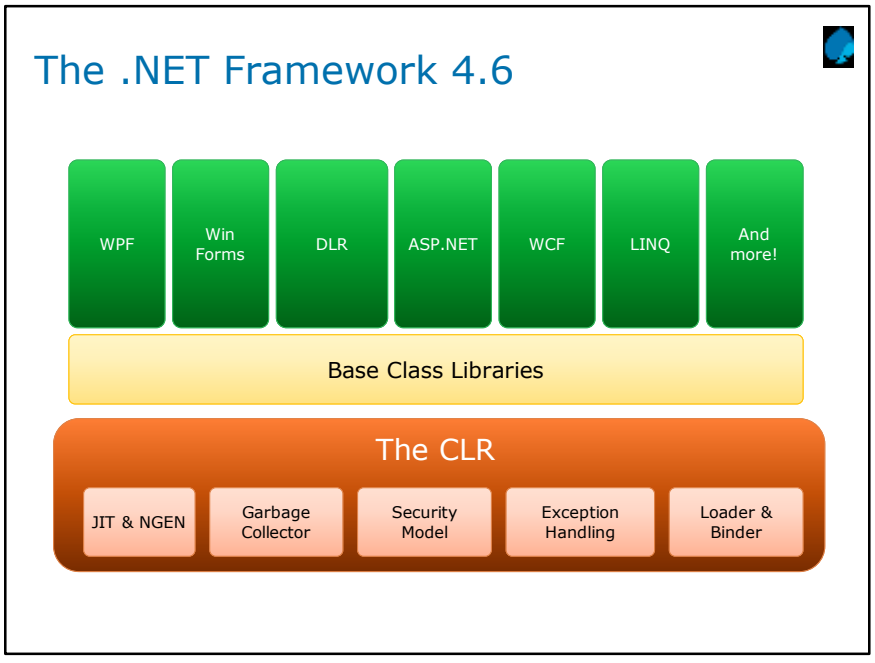
Diagrammatic Representation



.NET Framework, Languages, and Tools:

The common language runtime provides a code-execution environment that manages code targeting the .NET Framework. Code management can take the form of memory management, thread management, security management, code verification and compilation, and other system services.

Managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.


The .NET Framework 4.6
WPF
Win Forms
DLR
ASP.NET
WCF
LINQ
And more!
Base Class Libraries
The CLR
JIT & NGEN
Garbage Collector
Security Model
Exception Handling
Loader & Binder

Concept of .NET

- .NET implementation is language agnostic
- .NET implementation enables multi-language development
- Multi-language development works with
  - Common Type System (CTS)
  - Common Language Specification (CLS)

Common Type System (CTS)

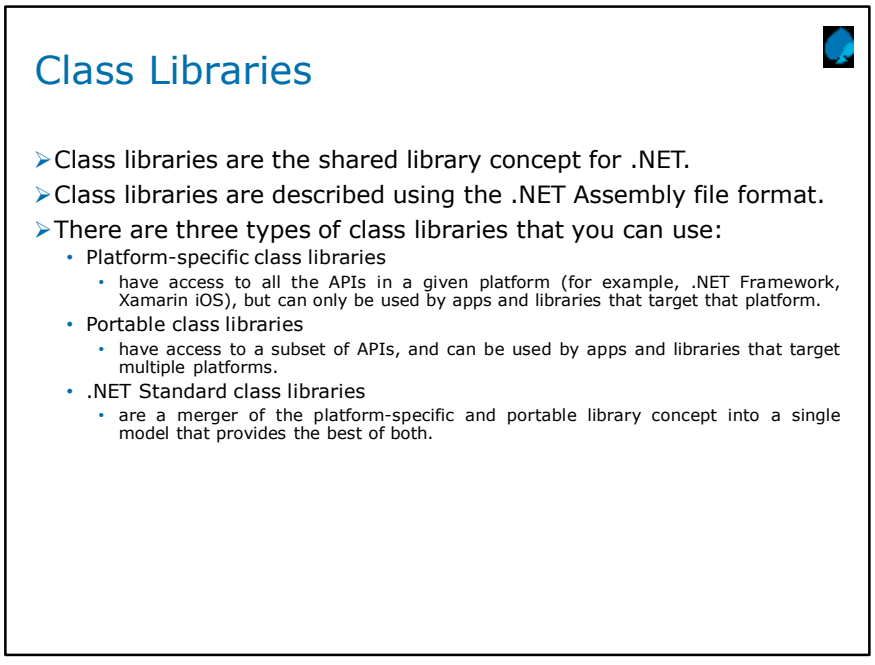
- The common type system defines how types are declared, used, and managed in the common language runtime.
- CTS is an important part of the runtime's support for cross-language integration
- The common type system performs the following functions:
  - Establishes a framework that helps enable cross-language integration, type safety, and high-performance code execution.
  - Provides an object-oriented model that supports the complete implementation of many programming languages.
  - Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.
  - Provides a library that contains the primitive data types (such as Boolean, Byte, Char, Int32, and UInt64) used in application development.

Common Language Specification (CLS)

- CLS defines a set of features that are needed by many common applications.
- It also provides a sort of recipe for any language that is implemented on top of .NET on what it needs to support.
- CLS is a subset of the CTS.
- To fully interact with other objects written in any language, objects must expose to callers only those features that are common to all languages.
- This common set of features is defined by the Common Language Specification (CLS), which is a set of rules that apply to generated assemblies.
- Some rules from CLS
  - Array should be start with zero index
  - Interface can have properties, events and virtual methods. It cannot have static methods, fields, methods with implementation
  - Constructor of derived class must call constructor of base class

To enable full interoperability scenarios, all objects that are created in code must rely on some commonality in the languages that are consuming them (are their callers). Since there are numerous different languages, .NET has specified those commonalities in something called the Common Language Specification (CLS). CLS defines a set of features that are needed by many common applications. It also provides a sort of recipe for any language that is implemented on top of .NET on what it needs to support.

CLS is a subset of the CTS. This means that all of the rules in the CTS also apply to the CLS, unless the CLS rules are more strict. If a component is built using only the rules in the CLS, that is, it exposes only the CLS features in its API, it is said to be CLS-compliant. For instance, the <framework-librares> are CLS-compliant precisely because they need to work across all of the languages that are supported on .NET.



Class libraries are the shared library concept for .NET. They enable you to componentize useful functionality into modules that can be used by multiple applications. They can also be used as a means of loading functionality that is not needed or not known at application startup. Class libraries are described using the .NET Assembly file format.

There are three types of class libraries that you can use:

Platform-specific class libraries have access to all the APIs in a given platform (for example, .NET Framework, Xamarin iOS), but can only be used by apps and libraries that target that platform.
Portable class libraries have access to a subset of APIs, and can be used by apps and libraries that target multiple platforms.
.NET Standard class libraries are a merger of the platform-specific and portable library concept into a single model that provides the best of both.

Concept of .NET

- .NET Framework class library is object-oriented collection of reusable classes.
- You use them to develop applications like:
  - Command-line applications
  - Graphical User Interface (GUI) based Desktop applications
  - Web Applications
  - Web Services
  - Distributed Applications

What is the .NET Framework?

The .NET Framework class library is a comprehensive, object-oriented collection of reusable classes that you can use to develop applications.

The applications can range from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET and Web Services.

1.2: What is the .NET Framework?

Concept of .NET

- It includes the following components:
  - Common Language Runtime (CLR)
  - Rich Class Libraries (FCL/BCL)
    - For e.g.: API for ADO.NET, XML, Threading, Windows Development etc.

What is the .NET Framework?

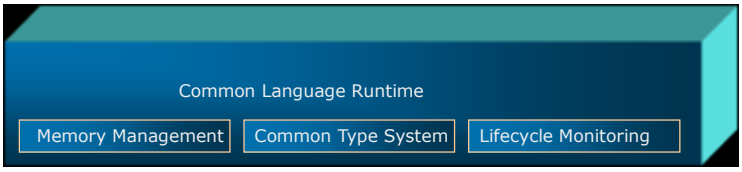
The .NET Framework is a new computing platform designed to simplify application development in the highly distributed environment of the Internet.

The .NET Framework has two main components:

- Common Language Runtime (CLR)
- NET Framework class library

The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict safety and accuracy of the code. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as **managed code**; code that does not target the runtime is known as **unmanaged code**.

Diagrammatic Representation



- Role of CLR:
  - CLR manages the following:
    - Running of code
    - Memory management
    - Compilation of code
    - Provides garbage collection, error handling
    - Code access security for semi-trusted code

.Net Framework Major Components (contd.):

At the bottom is the CLR (Common Language Runtime). It is considered as the heart of .NET Framework.

The CLR drives the key functionality.

The CLR handles the memory management.

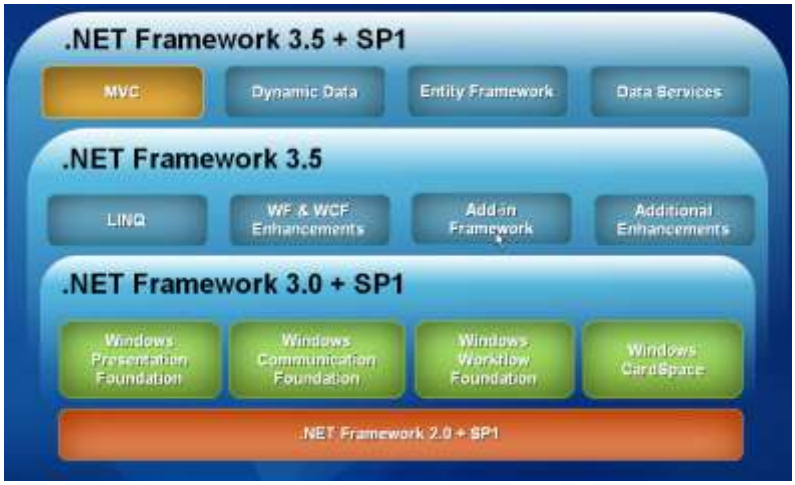
It has a common type subsystem for use across all languages of .NET.

CLR manages lifecycle of objects by way of reference counting and garbage collection.

.NET Evolution

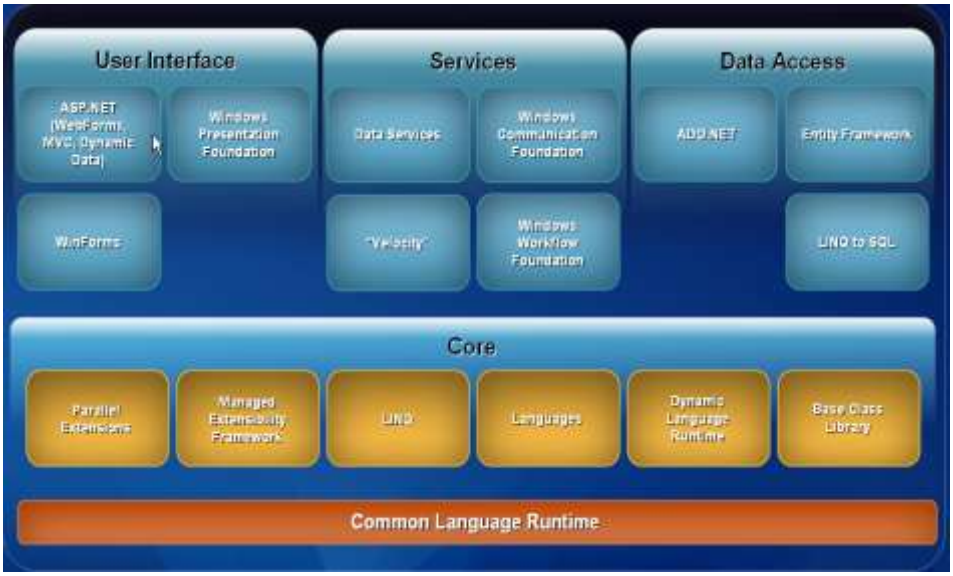
Year	CLR	.NET	Visual Studio	C#	VB
2002	1.0	1.0	2002	1.0	7.0
2003	1.1	1.1	2003	1.0	7.0
2005	2.0	2.0	2005	2.0	8.0
2006	2.0	3.0	.NET 3.0 ext.	2.0	8.0
2007	2.0	3.5	2008	3.0	9.0
2010	4	4	2010	4.0	10.0
2012	4	4.5	2012	5.0	11.0
2015	4	4.5.2	2015	6.0	12.0
2017	4	4.6.2	2017	6.0	12.0
2018	4	4.7.2	2017	7.0	13.0
2019	4	4.8	2019	8.0	14.0

The.NET Framework 3.5 Layer



Technically, the .NET 3.5 release is dominated by four new frameworks—WPF, WCF, WF, and CardSpace—which made their first appearances in .NET 3.0.

.NET Framework 4.0



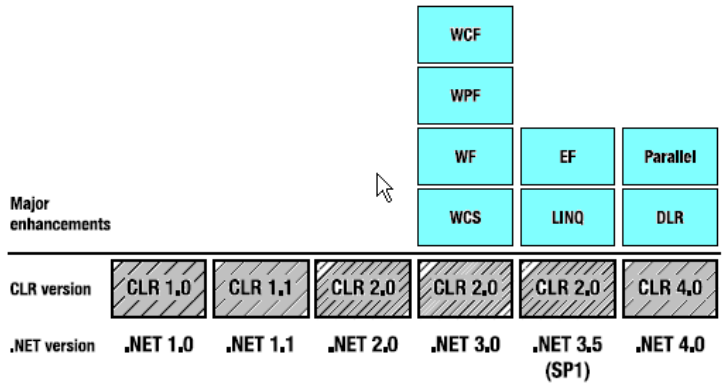
.NET Framework 4.7 and C# 7.0

Introduction to .NET Framework 4.6

CLR and BCL Changes

- The last two releases of.NET (3.0 and 3.5) have been additive releases building on top of the functionality available in CLR version 2.0
- .NET 4.0 however has a new version of the CLR
- Installing .NET 4.0 will not affect existing .NET applications running on previous versions of the framework

New CLR 4.0



The last two releases of .NET (3.0 and 3.5) have been additive releases building on top of the functionality available in CLR version 2.0.

.NET 4.0 however has a new version of the CLR! So you can install .NET 4.0 without fear that it will affect your existing .NET applications running on previous versions of the framework.

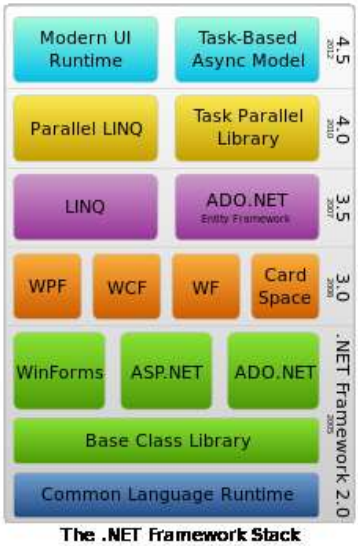
ASP.NET: When using IIS7, the CLR version is determined by the application pool settings. Thus you should be able to run .NET 4.0 ASP.NET applications side by side without fear of affecting existing ASP.NET sites.

Maturation of Existing Technologies - .NET 4

- Windows Workflow Foundation (WF) and Windows Communication Foundation (WCF) are much more closely integrated
- WF has a radical change with a much-improved designer, introduction of new activities, and easier customization
- WCF becomes simpler to use and also introduces new service discovery functionality
- WPF has some great additions, with an improved designer, multi-touch, and Windows 7 task bar support
- Entity Framework architecture is improved
- Enhancements in ASP.NET

.NET Framework 4.5 Core Enhancement

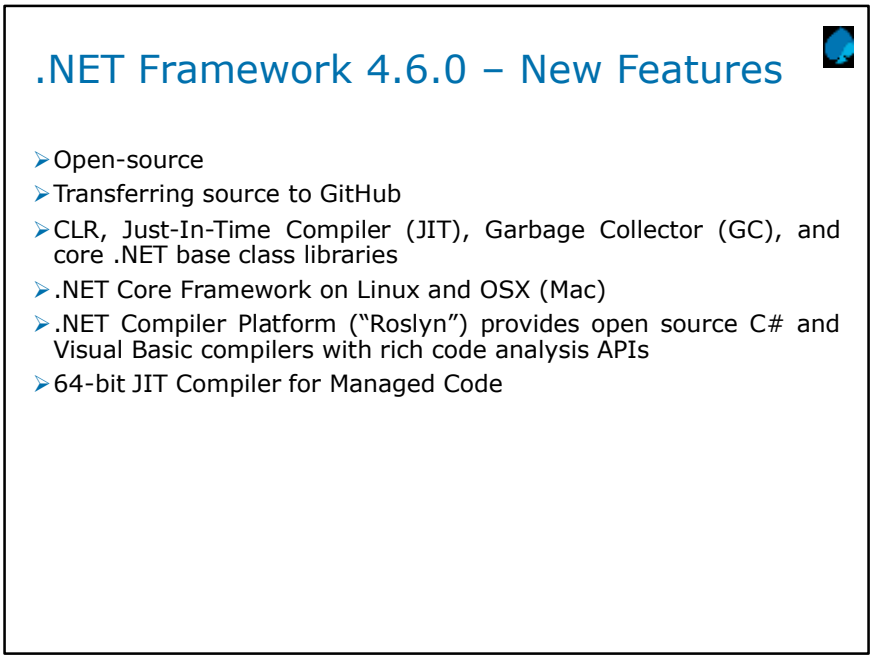
- Windows 8 Support
- Making Asynchronous programming as easy as possible
- Improved Memory Management with parallel background Garbage Collection and better concurrency.
- NuGet – A preferred Distribution and update Mechanism
- Supporting emerging standards and Technologies



The .NET Framework Stack

Why .NET 4.5 and VS2012

- VS2012 and .NET 4.5 Themes:
  - Changes and Improvements in the BCL
  - Maturation of existing technologies
  - Making Async Programming as Easy as Possible
  - Supporting Emerging Standards and Technologies
  - Promoting NuGet as the Preferred Distribution and Update Mechanism
  - Making It Easy to Develop on Different Devices and Other Platforms



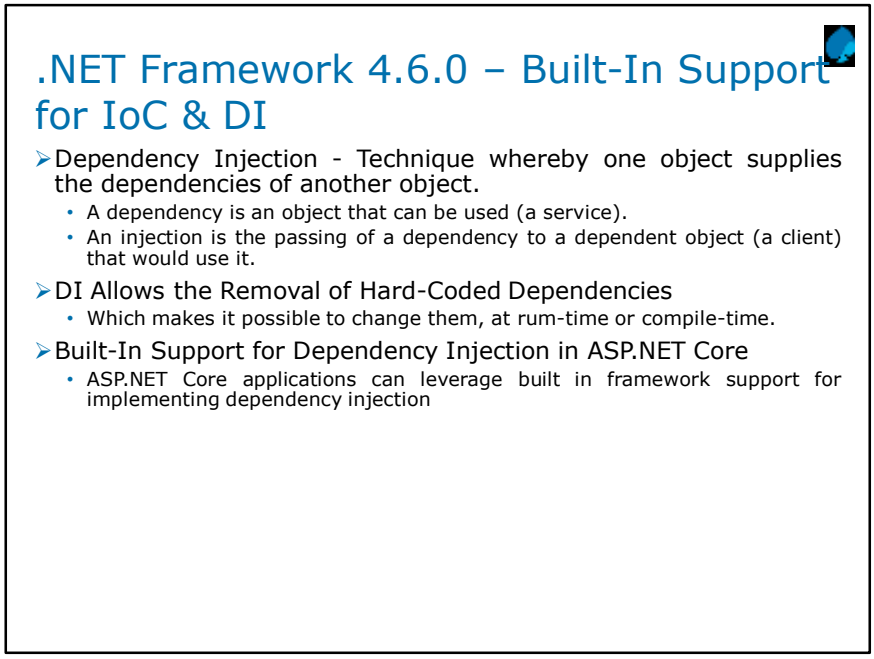
.NET 2015 introduces the .NET Framework 4.6 and .NET Core. Some new features apply to both, and other features are specific to .NET Framework 4.6 or .NET Core.

Now .NET Core has been made open source. Now it has been released with the MIT license. Right now Microsoft has released a few libraries code while the full Core code will be released at the time of final .NET release.

The .NET Compiler Platform ("Roslyn") provides open-source C# and Visual Basic compilers with rich code analysis APIs. You can build code analysis tools with the same APIs that Microsoft is using to implement Visual Studio!

The .NET Framework 4.6 features a new version of the 64-bit JIT compiler (originally code-named RyuJIT).

Please refer to below site for more information
https://docs.microsoft.com/en-us/dotnet/framework/whats-new/#whats-new-in-the-net-framework-462



Dependency Injection is the process of “injecting” the “dependency” whenever the dependency is requested by a client, where the dependency may be another service which the client (requester) may have no means of knowing how to create.

As an analogy, imagine a person (client) going to office carrying his lunch cooked by himself. In this scenario, the person has a “dependency” on food. But he had to know how to cook his food. But honestly, not everyone (client) knows to cook, but people do need food (dependency). This is where restaurants play the role of dependency Injection. They can supply food (“inject dependency”) to the people (client) without the person (client) needing to know how to cook.

ASP.NET core applications can leverage built in framework support for implementing dependency injection.

.NET Framework 4.6.1 – New Features

- Cryptography: Support for X509 certificates containing ECDSA
- ADO.NET Improvement
- WPF Improvement
- Profiling
- (NGEN) PDBs

The .NET Framework 4.6 added RSACng support for X509 certificates. The .NET Framework 4.6.1 adds support for ECDSA (Elliptic Curve Digital Signature Algorithm) X509 certificates.

Always Encrypted support for hardware protected keys ADO.NET now supports storing Always Encrypted column master keys natively in Hardware Security Modules (HSMs)

In WPF, Improved performance The delay in firing touch events has been fixed in the .NET Framework 4.6.1. Also typing in a RichTextBox control no longer ties up the render thread during fast input. Spell checking improvements.

The unmanaged Profiling API has been enhanced.

Cross-machine event tracing allows customers to profile a program on Machine A and look at the profiling data with source line mapping on Machine B

.NET Framework 4.6.2 – New Features

- ASP.NET Area Features
  - Improved support for localized error messages in data annotation validators
  - Async support for session-state store providers
  - Async support for output-cache providers
- Character Encoding - Unicode Standard, Version 8.0.0.
- Cryptography - Support for X509 certificates containing FIPS 186-3 DSA
- ADO.NET SQLClient - Connection pooling and timeouts with Azure SQL databases

Data annotation validators enable you to perform validation by adding one or more attributes to a class property. The attribute's ValidationAttribute.ErrorMessage element defines the text of the error message if validation fails. Starting with the .NET Framework 4.6.2, ASP.NET makes it easy to localize error message.

ASP.NET now allows task-returning methods to be used with session-state store providers, thereby allowing ASP.NET apps to get the scalability benefits of async.

Starting with the .NET Framework 4.6.2, task-returning methods can be used with output-cache providers to provide the scalability benefits of async

Characters in the .NET Framework 4.6.2 are classified based on the Unicode Standard, Version 8.0.0. In .NET Framework 4.6 and .NET Framework 4.6.1, characters were classified based on Unicode 6.3 character categories.

The .NET Framework 4.6.2 adds support for DSA (Digital Signature Algorithm) X509 certificates whose keys exceed the FIPS 186-2 1024-bit limit.

NET Framework Data Provider for SQL Server (System.Data.SqlClient) includes Connection pooling and timeouts with Azure SQL databases.

.NET Framework 4.6.2 – New Features (Cont.…)

- Windows Communication Foundation
  - WCF transport security support for certificates stored using CNG
  - Better support for multiple daylight-saving time adjustment rules by the DataContractJsonSerializer class
  - Support for preserving a UTC time when serializing and deserializing with the XMLSerializer class
  - NetNamedPipeBinding best match
  - SSL 3.0 is not a default protocol

WCF transport security supports certificates stored using the Windows cryptography library (CNG).

Customers can use an application configuration setting to determine whether the DataContractJsonSerializer class supports multiple adjustment rules for a single time zone.

You can use an application configuration setting to determine whether the XmlSerializer preserves UTC time zone information when serializing and deserializing DateTime values.

WCF has a new app setting that can be set on client applications to ensure they always connect to the service listening on the URI that best matches the one that they request.

When using NetTcp with transport security and a credential type of certificate, SSL 3.0 is no longer a default protocol used for negotiating a secure connection.

.NET Framework 4.6.2 – New Features (Cont.…)

- Windows Presentation Framework
  - Soft keyboard support
  - Group sorting
  - Per-monitor DPI
- ClickOnce improvement - Support TLS 1.1 & TLS 1.2
- Converting Windows Forms and WPF apps to UWP apps

Soft Keyboard support enables focus tracking in a WPF applications by automatically invoking and dismissing the new Soft Keyboard in Windows 10 when the touch input is received by a control that can take textual input.

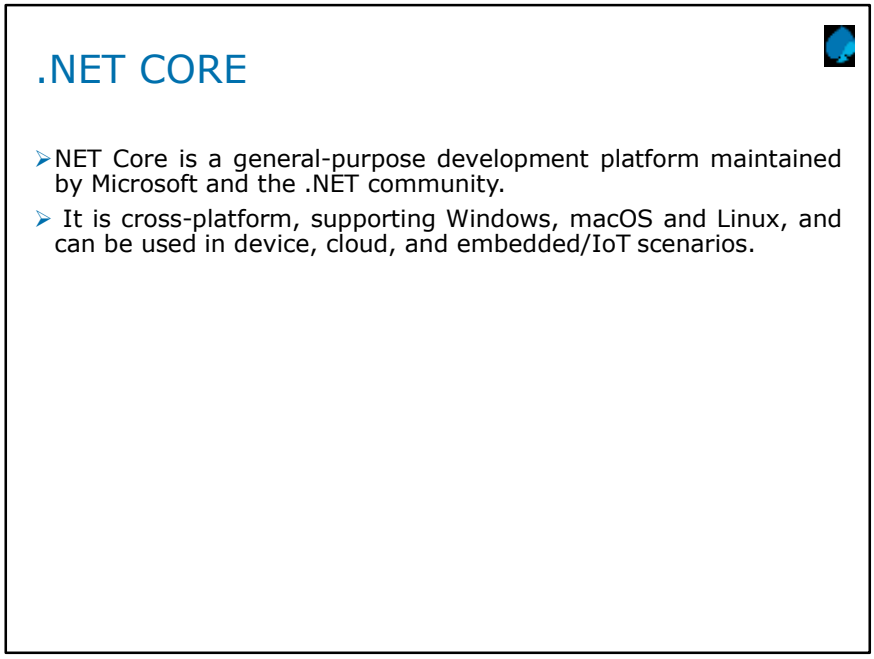
An application that uses a CollectionView object to group data can now explicitly declare how to sort the groups.

To support the recent proliferation of high-DPI and hybrid-DPI environments for WPF apps, WPF in the .NET Framework 4.6.2 enables per-monitor awareness.

ClickOnce has been updated to support TLS 1.1 and TLS 1.2 in addition to the 1.0 protocol, which it already supports. ClickOnce automatically detects which protocol is required; no extra steps within the ClickOnce application are required to enable TLS 1.1 and 1.2 support.

Windows now offers capabilities to bring existing Windows desktop apps, including WPF and Windows Forms apps, to the Universal Windows

Platform (UWP). This technology acts as a bridge by enabling you to gradually migrate your existing code base to UWP, thereby bringing your app to all Windows 10 devices.



Following are the Features :-

Flexible deployment: Can be included in your app or installed side-by-side user- or machine-wide.

Cross-platform: Runs on Windows, macOS and Linux; can be ported to other operating systems.

Command-line tools: All product scenarios can be exercised at the command-line.

Compatible: .NET Core is compatible with .NET Framework, Xamarin and Mono, via the .NET Standard

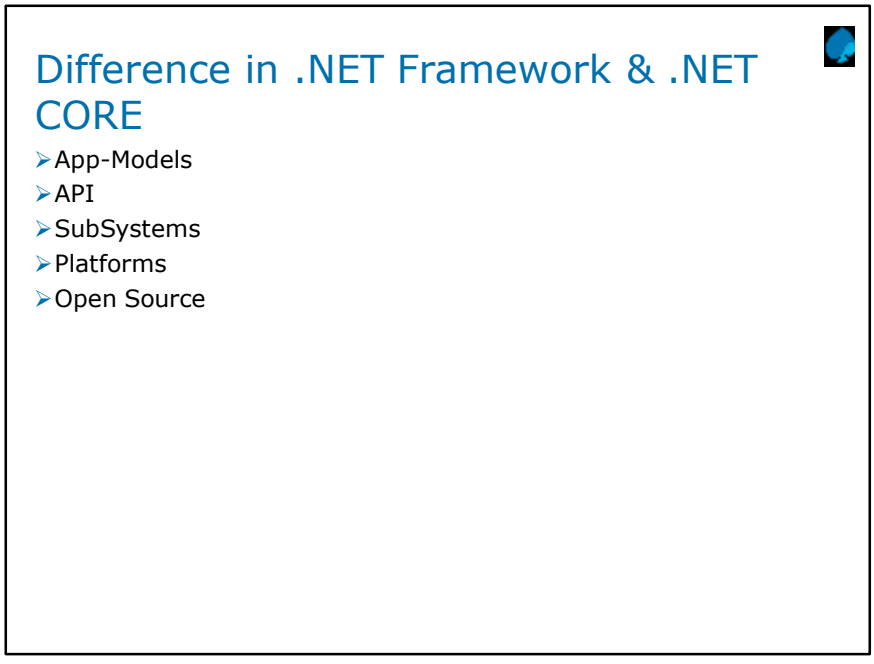
Open source: The .NET Core platform is open source, using MIT and Apache 2 licenses.

.NET Core is composed of

- A .NET Runtime which provides a type system, assembly loading, a garbage collector, native interop and other basic services.
- The 'dotnet' app host, which is used to launch .NET Core apps. It selects the runtime and hosts the runtime, provides an assembly loading policy and launches the app. The same host is also used to launch SDK tools in much the same way.
- A set of framework libraries, which provide primitive data types, app composition types and fundamental utilities

.NET Core Evolution

Year	.NET Core	Visual Studio
2016	.NET Core 1.0	Visual Studio 2015 Update 3
2016	.NET Core 1.1	Visual Studio 2017 Version 15.0
2017	.NET Core 2.0	Visual Studio 2017 Version 15.3
2018	.NET Core 2.1	Visual Studio 2017 Version 15.7
2018	.NET Core 2.2	Visual Studio 2019 Version 16.0
2019	.NET Core 3.0	Visual Studio 2019 Version 16.3
2019	.NET Core 3.1	Visual Studio 2019 Version 16.4



The major differences between .NET Core and the .NET Framework:

App-models -- .NET Core does not support all the .NET Framework app-models, in part because many of them are built on Windows technologies, such as WPF (built on top of DirectX). The console and ASP.NET Core app-models are supported by both .NET Core and .NET Framework.

APIs -- .NET Core contains many of the same, but fewer, APIs as the .NET Framework, and with a different factoring (assembly names are different; type shape differs in key cases). These differences currently typically require changes to port source to .NET Core. .NET Core implements the .NET Standard API, which will grow to include more of the .NET Framework BCL API over time.

Subsystems -- .NET Core implements a subset of the subsystems in the .NET Framework, with the goal of a simpler implementation and programming model. For example, Code Access Security (CAS) is not supported, while reflection is supported.

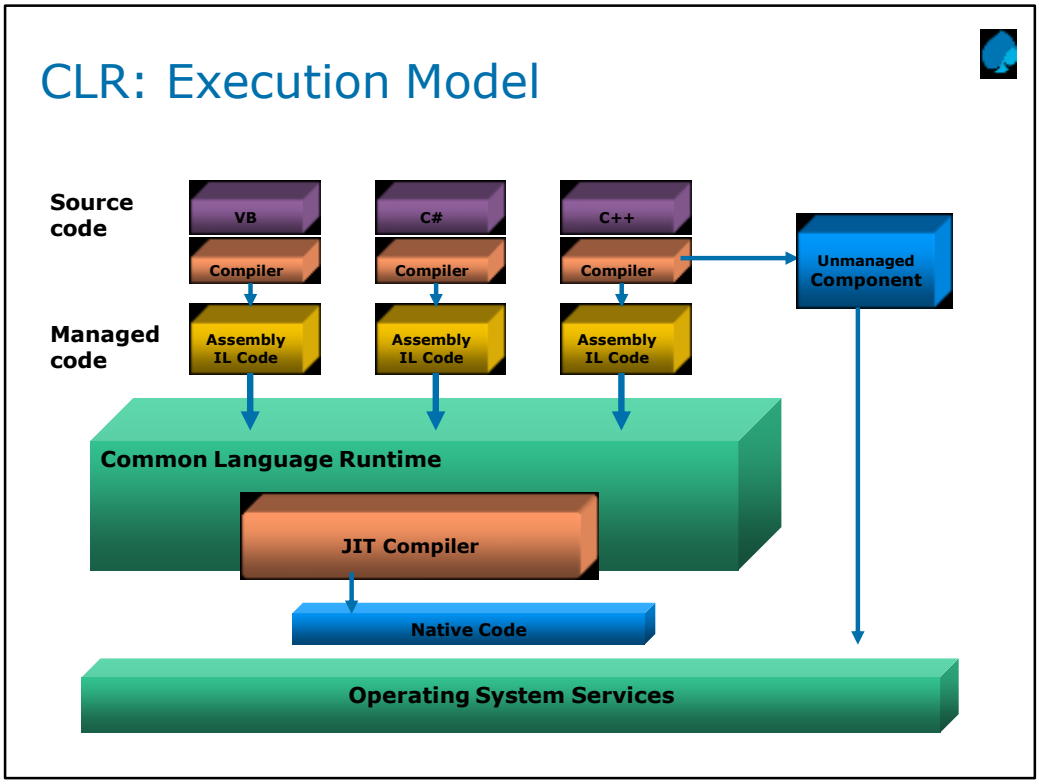
Platforms -- The .NET Framework supports Windows and Windows Server while .NET Core also supports macOS and Linux.

Open Source -- .NET Core is open source, while a read only subset of

.NET Framework is open source.

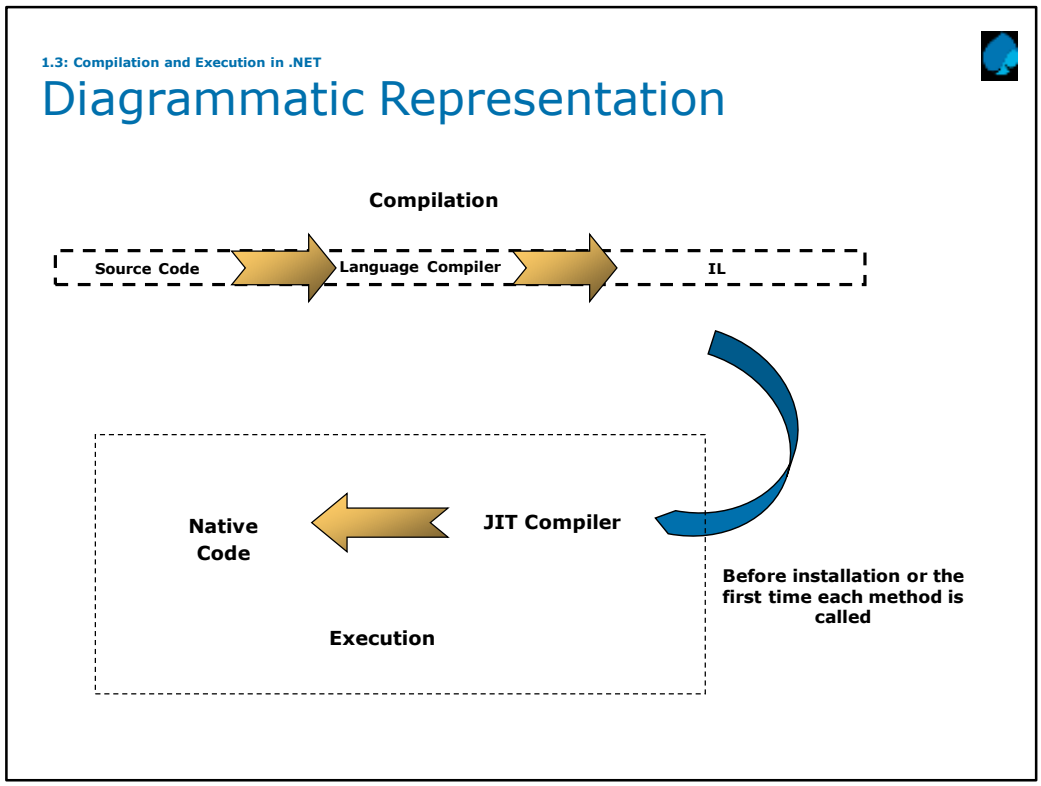
Use .NET Core for your server application when:
You have cross-platform needs.
You are targeting microservices.
You are using Docker containers.
You need high-performance and scalable systems.
You need side-by-side .NET versions per application.

Use .NET Framework for your server application when:
Your app currently uses .NET Framework
Your app uses third-party .NET libraries or NuGet packages not available for .NET Core.
Your app uses .NET technologies that aren't available for .NET Core.
Your app uses a platform that doesn’t support .NET Core.



When you compile code that uses the .NET Framework library, you don't immediately create operating-system-specific native code. Instead, you compile your code into *Common Intermediate Language (CIL)* code. This code isn't specific to any operating system (OS) and isn't specific to C#. Other .NET languages — Visual Basic .NET, for example — also compile to this language as a first stage. This compilation step is carried out by VS or VCE when you develop C# applications.

Obviously, more work is necessary to execute an application. That is the job of a *just-in-time* (JIT) compiler, which compiles CIL into native code that is specific to the OS and machine architecture being targeted. Only at this point can the OS execute the application. The *just-in-time* part of the name reflects the fact that CIL code is compiled only when it is needed.



Compilation and Execution in .NET:

The managed execution process includes the following steps:

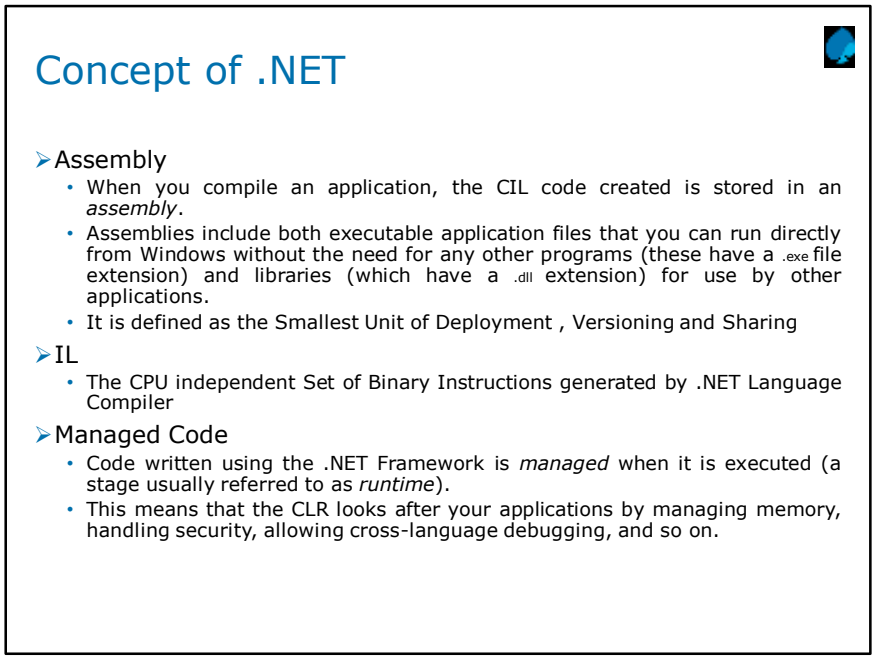
Designing and writing your source code

To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.

Compiling your code to Intermediate language (IL)

Compiling translates your source code into IL and generates the required metadata.

When compiling to managed code, the compiler translates your source code into intermediate language (IL), which is a CPU-independent set of instructions that can be efficiently converted to native code. IL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. Before code can be executed, MSIL must be converted to CPU-specific code by a just in time (JIT) compiler. Since the runtime supplies one or more JIT compilers for each computer architecture it supports, the same set of MSIL can be JIT-compiled and executed on any supported architecture.



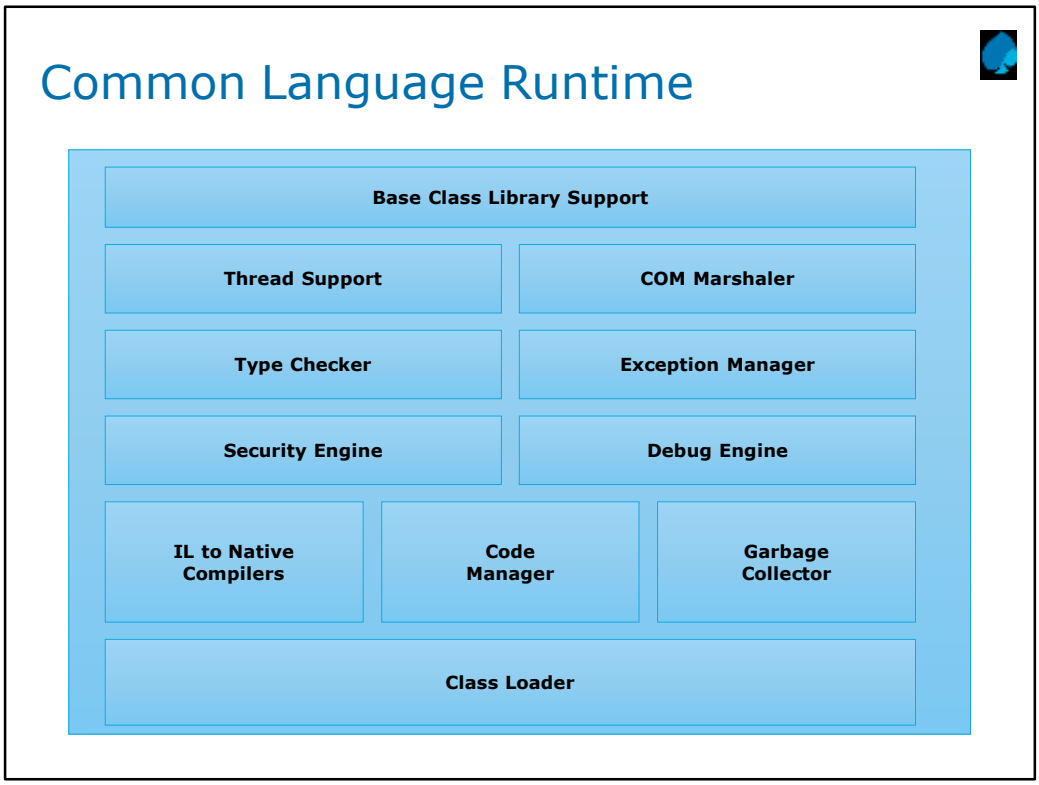
Assembly

In the .NET framework, an assembly is a compiled code library used for deployment, versioning, and security. There are two types: process assemblies (EXE) and library assemblies (DLL). A process assembly represents a process which will use classes defined in library assemblies. .NET assemblies contain code in CIL, which is usually generated from a CLI language, and then compiled into machine language at runtime by the CLR just-in-time compiler.

In addition to containing CIL, assemblies also include meta information (that is, information about the information contained in the assembly, also known as metadata) and optional resources (additional data used by the CIL, such as sound files and pictures). The meta information enables assemblies to be fully self-descriptive. You need no other information to use an assembly, meaning you avoid situations such as failing to add required data to the system registry and so on, which was often a problem when developing with other platforms.

UnManaged Code: By contrast, applications that do not run under the

control of the CLR are said to be unmanaged, and certain languages such as C++ can be used to write such applications, which, for example, access low-level functions of the operating system. However, in C# you can write only code that runs in a managed environment. You will make use of the managed features of the CLR and allow .NET itself to handle any interaction with the operating system.



Compilation and Execution in .NET:

Class Loader: This component provides metadata management and loads classes. This is part of the execution process and involves the metadata stored within the executable file. Class Loader uses Code Manager to assign memory for the objects and data. It computes the layout of classes in memory and each method receives an entry in the methods table. Later, when the **Just-In-Time** compiler converts the MSIL code (see next point) to native code, the entry in the methods table is replaced with the pointer to the native code.

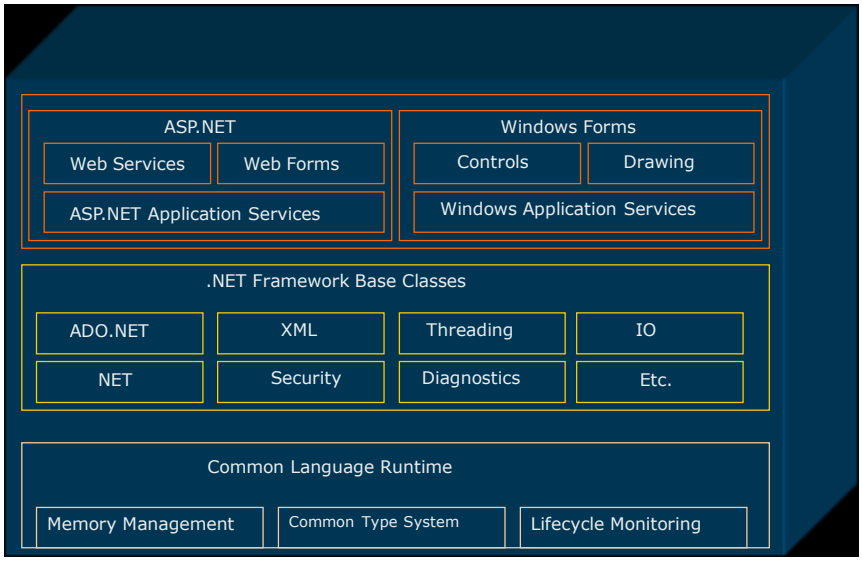
Intermediate Language (IL) to native compilers: This component converts code in Microsoft intermediate language to native code. This can be done through either Just-in-Time (JIT) compiling or native generation using the NGEN tool that comes as part of the Microsoft .NET Framework SDK. (More information is provided later in this lesson).

Code Manager: This component manages code execution and is used by the Class Loader to assign memory for the objects and data.

Garbage collector: This component manages the allocation and release of memory for the application, and automatically reclaims unused memory. Its optimizing engine determines the best time to perform a collection in order to free some memory based on the allocations being made. When the garbage collector performs a collection, it checks for objects in the managed heap that are no longer being used by the application and performs the necessary operations to reclaim their memory. The System.GC class can be used to control the system garbage collector.

1.5: .NET Framework Major Components

Diagrammatic Representation



.NET Framework Major Components:

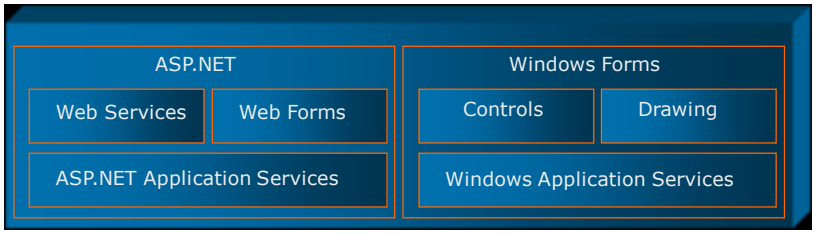
The runtime enforces security in a way that enables users to trust that although an executable attached to an e-mail can play an animation on screen or sing a song, it cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature-rich.

The runtime also enforces code robustness by implementing a strict **type-** and **code-verification infrastructure** called the **common type system** (CTS). The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS. This means that managed code can consume other managed classes, types, and objects, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime ensures that the most common types of software issues are solved or eradicated completely. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management eliminates the two most common application errors, memory leaks, and invalid memory references.

Explanation

- The top layer consists of user and program interfaces.
- Windows Forms, WPF can be used for Desktop Development
- Asp. Net Web Forms for the Web Development.



.NET Framework Major Components (contd.):

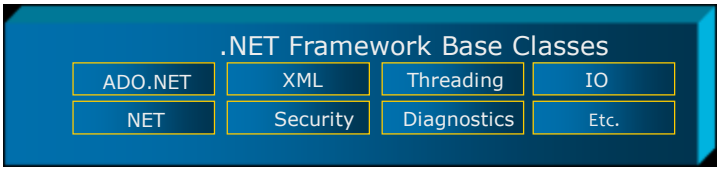
Client applications are the closest to a traditional style of application in Windows-based programming. These are the types of applications that bring up Windows or Forms on the desktop and which you use to perform a task. Client applications include applications such as word-processors and spreadsheets, as well as custom business applications such as data-entry tools, reporting tools, and so on. Client applications usually employ windows, menus, buttons, and other GUI elements, and they likely access local resources such as the file system and peripherals such as printers.

Another kind of client application is the traditional ActiveX control (now replaced by the managed Windows Forms control) deployed over the Internet as a Web page. These types of applications are much like other client applications, in that they are executed natively, have access to local resources, and include graphical elements.

In the past, developers created such applications using C/C++ in conjunction with the Microsoft Foundation Classes (MFC) or with a rapid application development (RAD) environment such as Microsoft® Visual Basic®. The .NET Framework incorporates aspects of existing products into a single, consistent development environment that drastically simplifies the development of client applications.

Explanation

- The middle layer consists of classes for next generation of system services.
- These are ADO.NET and XML as well as many others.
- Under the control of Framework, all components are universally available.
- Web Services, Remoting could be used.



.NET Framework Major Components (contd.):

The .NET Framework class library is a collection of reusable classes, or types, that tightly integrate with the common language runtime. The class library builds on the object-oriented nature of the runtime, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the learning curve associated with using a new piece of code. In addition, third-party components can integrate seamlessly with the classes in the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces, which you can use to develop your own collection classes. Your collection classes will then blend seamlessly with the classes in the .NET Framework.

As you would expect from an object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios.

1.6: Simplified Deployment

Explanation

- .NET Framework provides simplicity during deployment:
  - No registration required
    - Code is completely self-describing
    - Simply copy components to app dir
  - Zero-impact install
    - Installing one app will not affect another
  - Side-by-side execution
    - Multiple component versions can co-exist

1.7: .NET Framework Namespace

Explanation

- Namespaces are the way by which .NET avoids name clashes between classes.
- .NET requires all types to be defined in a namespace.
- The System namespace is the root namespace that contains the fundamental types of the .NET Framework.
  - This namespace contains the Object class that is the root of the inheritance hierarchy, primitive and extended types, and many other classes.

.NET Framework Namespace:

Namespaces are the way by which .NET avoids name clashes between classes. They are designed to avoid the situation in which you define a class to represent a customer, name your class Customer, and then someone else does the same thing (a likely scenario - the proportion of businesses that have customers seems to be quite high).

A namespace is no more than a grouping of data types, but it has the effect that the names of all data types within a namespace are automatically prefixed with the name of the namespace. It is also possible to nest namespaces within each other.

For example: Most of the general-purpose .NET base classes are in a namespace called System. The base class Array is in this namespace, so its full name is System.Array.

.NET requires all types to be defined in a namespace;

For example: You can place your Customer class in a namespace called YourCompanyName. This class will have the full name YourCompanyName.Customer.

The System namespace is the root namespace that contains the fundamental types of the .NET Framework. This namespace contains the Object class that is the root of the inheritance hierarchy, primitive and extended types, and many other classes. Indeed, there are almost 100 classes to handle exceptions, support runtime execution, application domains, garbage collection, and so on.

Demo

- Demo on how the various .NET framework versions are installed on a machine.

Summary

- In this lesson, you have learnt about:
  - Microsoft .NET and its main components
    - Development tools and .NET languages
    - .NET Enterprise servers
    - The Microsoft .NET Framework
    - The .NET Framework class library, Common Language Runtime, and web services
  - The Common Language Runtime
    - The components of the Common Language Runtime
    - The concept of managed code, which includes compiler-generated code in Microsoft Intermediate Language, metadata, as well as Just-in-Time compiling into the native, platform-dependent code.

Review Questions

- Question 1: The ___ is the foundation of the .NET Framework.
- Question 2: Code that targets the runtime is known as ___; code that does not target the runtime is known as ___.
- Question 3: In .NET, the applications are Compiled to a common language called ___.
- Question 4: ___ component manages the allocation and release of memory for the application, and automatically reclaims unused memory.

Answers for the Review Questions:

Answer 1: CLR
Answer 2: Managed code, unmanaged code
Answer 3: Intermediate Language (IL)
Answer 4: Garbage Collector