.NET Intro

.NET Introduction

Free, open-source development platform

App types:

- **Web apps, web APIs, and microservices**
- Serverless functions in the cloud
- Cloud native apps
- Mobile apps
- Desktop apps
  - Windows WPF
  - Windows Forms
  - Universal Windows Platform (UWP)
- Games
- Internet of Things (IoT)
- Machine learning
- Console apps
- Windows services

.NET class libraries

- Shared library concept
- They can be used by multiple applications
- There are three types:

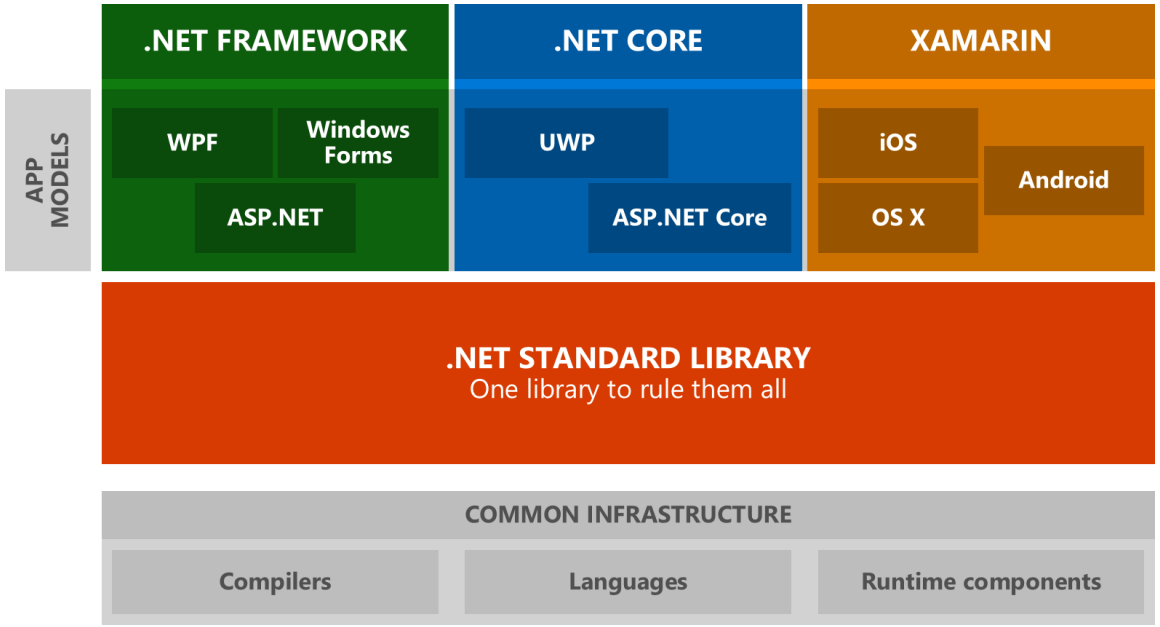
Platform-specific class libraries have access to all the APIs in a given platform (for example, .NET Framework on Windows, Xamarin iOS), but can only be used by apps and libraries that target that platform.

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.

.NET Standard

- formal specification of .NET APIs that are available on multiple .NET implementations.
- establish greater uniformity in the .NET ecosystem
- .NET 5 and later versions adopt a different approach to establishing uniformity
- .NET Standard is versioned
- The recommendation is to target .NET Standard 2.0



.NET

- Supports three programming languages
  - C#
  - F#
  - Visual Basic
- .NET languages support:
  - Type safety
  - Type inference
  - Generic types
  - Delegates
  - Lambdas
  - Events
  - Exceptions
  - Attributes
  - Asynchronous code
  - Parallel programming
  - Code analyzers

.NET SDK

Contains:

- .NET CLI. Command-line tools that you can use for local development and continuous integration scripts.
- dotnet driver. A CLI command that runs framework-dependent apps.
- Roslyn and F# programming language compilers.
- MSBuild build engine.
- .NET runtime. Provides a type system, assembly loading, a garbage collector, native interop, and other basic services.
- Runtime libraries. Provides primitive data types and fundamental utilities.
- ASP.NET Core runtime. Provides basic services for internet-connected apps, such as web apps, IoT apps, and mobile backends.
- Desktop runtime. Provides basic services for Windows desktop apps, including Windows Forms and WPF.

Project system and MSBuild

- .NET app is built from source code by using MSBuild (Microsoft Build Engine):
  - platform for building applications
  - VS uses MSBuild
  - cmd:
    - `MSBuild.exe MyProj.proj -property:Configuration=Debug`
- Project file (.csproj, .fsproj, or .vbproj) specifies targets and associated tasks that are responsible for compiling, packing, and publishing code

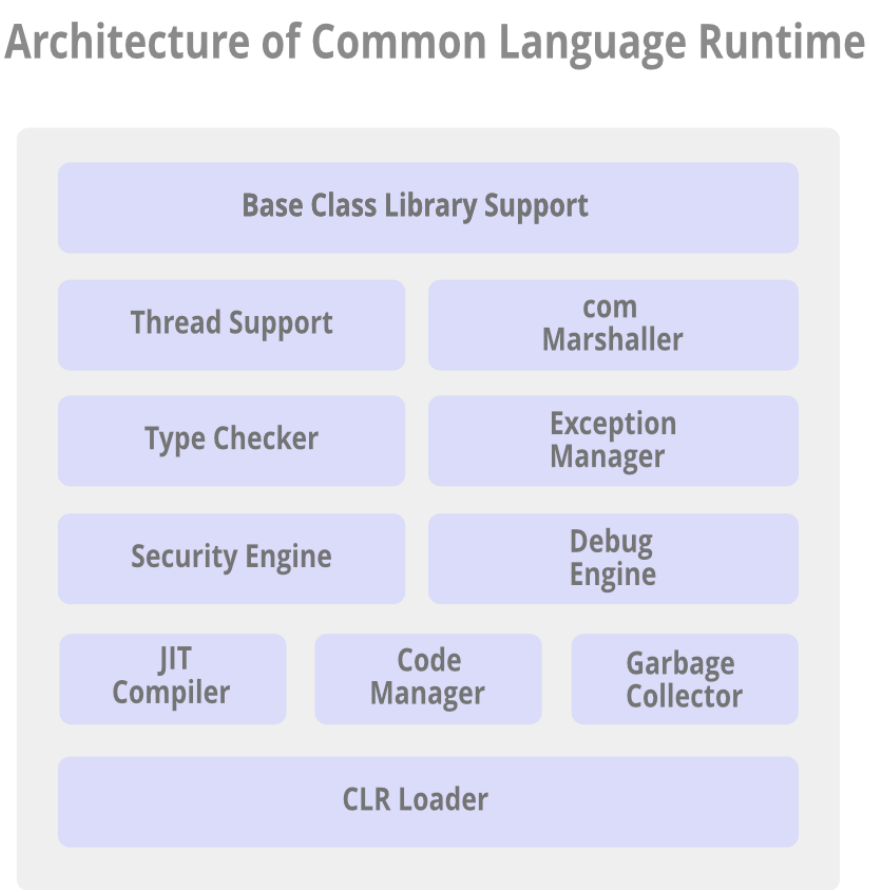
```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net6.0</TargetFramework>
 </PropertyGroup>
</Project>
```

```
<Project Sdk="Microsoft.NET.Sdk.Web">
 <PropertyGroup>
  <TargetFramework>net6.0</TargetFramework>
 </PropertyGroup>
</Project>
```

Execution models

.NET apps run managed code in a runtime environment known as the Common Language Runtime (CLR)

- handles memory allocation and management
- is a virtual machine that not only executes apps but also generates and compiles code using a just-in-time (JIT) compiler
- every loadable common language runtime portable executable (PE) file contains metadata
- The runtime uses metadata to locate and load classes, lay out instances in memory, resolve method invocations, generate native code, enforce security, and set run-time context boundaries.
- CLR makes it easy to design components and applications whose objects interact across languages. This is possible because of usage **common type system**



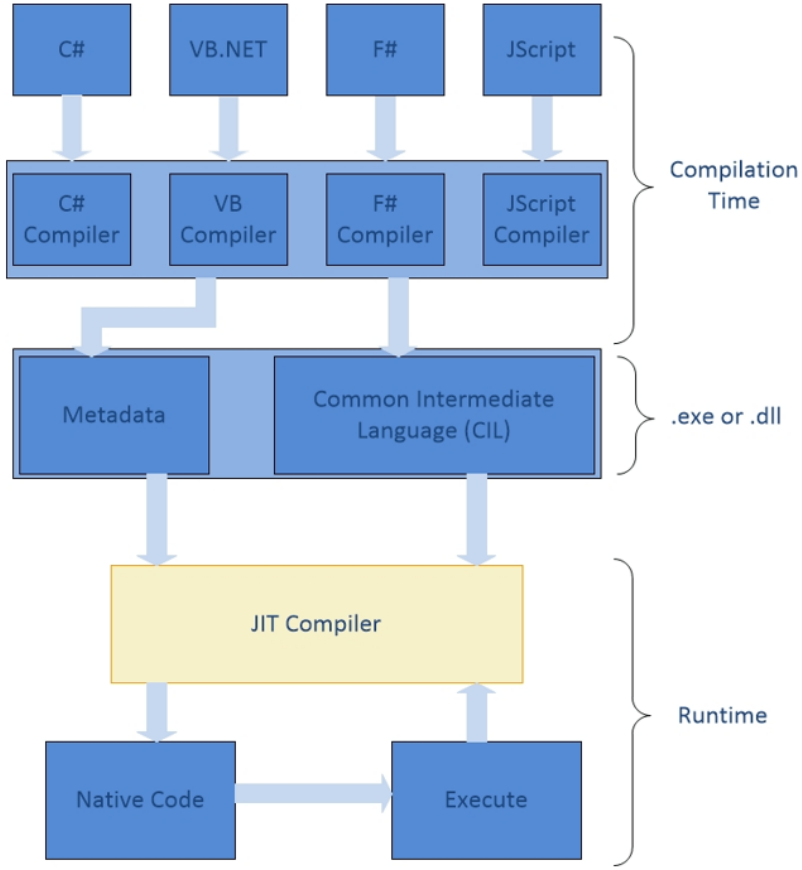
Managed Execution Process

1. Choosing a compiler. To obtain the benefits provided by the common language runtime, you must use one or more language compilers that target the runtime.
2. Compiling your code to MSIL. Compiling translates your source code into Microsoft intermediate language (MSIL) and generates the required metadata.
3. Compiling MSIL to native code. At execution time, a just-in-time (JIT) compiler translates the MSIL into native code. During this compilation, code must pass a verification process that examines the MSIL and metadata to find out whether the code can be determined to be type safe.
4. Running code. The common language runtime provides the infrastructure that enables execution to take place and services that can be used during execution.

JIT compiler and IL

Higher-level .NET languages compile down to a hardware-agnostic instruction set, which is called Intermediate Language (IL)

- JIT (Just-In-Time) compiler translates IL to machine code
- JIT compilation happens on the same machine that the code is going to run on
  - Therefore, JIT compilers have to balance time spent optimizing code against the savings that the resulting code can produce
- The .NET JIT compiler can do *tiered compilation* (recompile individual methods at run time)
- Pre-JIT compilation



Assemblies in .NET

Assemblies in .NET

- forms fundamental units of deployment, version control, reuse, activation scoping, and security permissions for .NET-based applications.
- are the collections of types and resources that are built to work together and form a logical unit of functionality
- take the form of executable (*.exe*) or dynamic link library (*.dll*) files.
- provide the common language runtime with the information it needs to be aware of type implementations.
- have the following properties:
  - are implemented as *.exe* or *.dll* files.
  - One can share assemblies between applications by putting them in the global assembly cache (GAC).
  - Assemblies are only loaded into memory if they are required.
  - You can programmatically obtain information about an assembly by using reflection.

Memory usage

Automatic memory management

- *Garbage collector* (GC)
- Managed Heap – segment of memory to store and manage objects
  - There is a managed heap for each managed process.
  - All threads in the process allocate memory for objects on the same heap.

Working with unmanaged resources

- Unmanaged resources are resources that aren't automatically maintained by the .NET runtime
  - IDisposable interface
    - Dispose()
    - using()

Deployment models

Self-contained application

- executable file includes
  - .NET runtime and libraries
  - application
  - its dependencies

Framework-dependent application

- produces
  - executable file and binary files (.dll)
- Users of the application have to separately install the .NET runtime.
- The executable file is platform-specific, but the *.dll* files of framework-dependent applications are cross-platform.

https://docs.microsoft.com/en-us/dotnet/core/deploying/

Data access

Object/Relational Mapper (ORM)

- Entity Framework (EF) Core
  - open source and cross-platform

Language-integrated query (LINQ)

- declarative code for operating on data

Resources and useful links

https://docs.microsoft.com/en-us/dotnet/core/

https://www.c-sharpcorner.com/

https://www.codeproject.com/?cat=24

https://dotnetfiddle.net/

https://sharplab.io/