

Understanding .NET: Concepts, Compilation, and Runtime

1. What is .NET and Why We Use It

- .NET is a development platform by Microsoft for building apps across desktop, web, mobile, and cloud.
- It provides a runtime (CLR), libraries (FCL/BCL), and compilers.
- Benefits: cross-language interoperability, memory management, security, and productivity.

2. Compilation Flow in .NET (C# Example)

1. Source Code -> compiled by Roslyn into MSIL + metadata.
2. Assembly (.exe/.dll) -> contains MSIL and metadata.
3. CLR Execution:
 - OS loader starts CLR when .exe is run.
 - CLR reads MSIL, uses JIT (RyuJIT) to compile into native code.
 - CPU executes the machine code.

3. Files Generated During Compilation

- .exe: Executable assembly containing MSIL.
- .dll: Library assembly for reusable code.
- Metadata: Type info, references.
- MSIL: Platform-independent instructions.

4. CLR, MSIL, and JIT Compiler

- CLR: Manages execution, memory, security, exceptions.
- MSIL: CPU-independent instructions generated by compilers.
- JIT Compiler: Converts MSIL into native code at runtime.
 - Normal JIT: Compiles methods as needed.
 - Pre-JIT (NGen): Compiles entire assembly ahead of time.
 - RyuJIT: Modern JIT for .NET Core and Framework.

5. Roslyn and SDKs

- Roslyn: Open-source C# and VB.NET compiler with APIs for code analysis.

Understanding .NET: Concepts, Compilation, and Runtime

- SDK: Software Development Kit includes compilers, libraries, tools, and runtime.

6. CTS, CLS, and Type Mapping

- CTS: Common Type System ensures all .NET languages share the same type definitions.
- CLS: Common Language Specification defines rules for cross-language compatibility.
- Example: 'int' in C# maps to System.Int32.

7. DLL Files

- DLL: Dynamic Link Library containing compiled MSIL code.
- Enables code reuse and modularity.
- Generated when building class libraries.

8. C# vs VB.NET Compilation Flow

- C#: Source -> Roslyn -> MSIL -> CLR -> JIT -> CPU.
- VB.NET: Source -> VB Compiler -> MSIL -> CLR -> JIT -> CPU.

9. .NET Framework vs .NET Core

Feature	.NET Framework	.NET Core
Platform	Windows only	Cross-platform
Libraries	Monolithic	Modular NuGet
Performance	Moderate	High
Future	Legacy	Actively developed

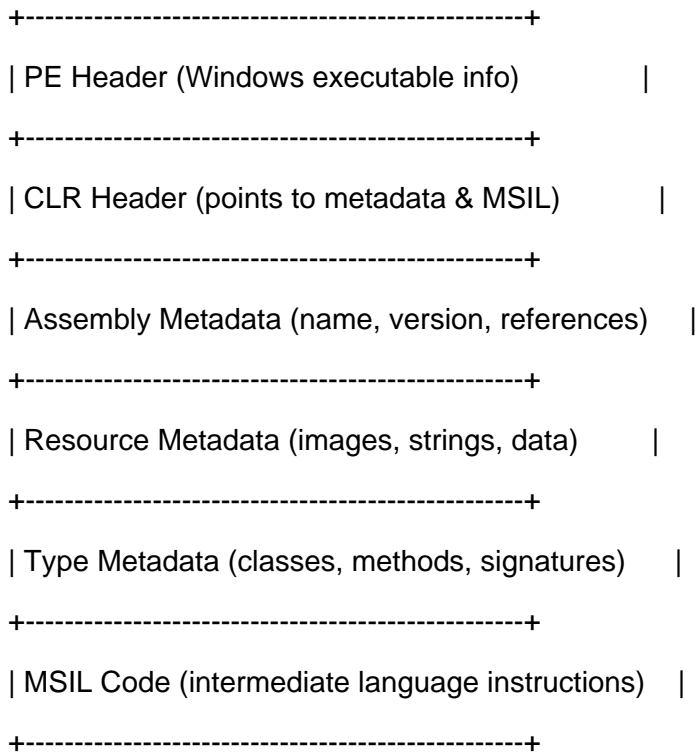
10. Assembly Structure and Components

- PE Header: Windows executable info.
- CLR Header: Points to metadata and MSIL.
- Assembly Metadata: Name, version, references.
- Resource Metadata: Embedded resources.

Understanding .NET: Concepts, Compilation, and Runtime

- Type Metadata: Classes, methods, signatures.
- MSIL Code: Intermediate language instructions.

Diagram:



11. FCL vs BCL

- FCL: Full library collection in .NET Framework (includes ASP.NET, ADO.NET, etc.).
- BCL: Core subset of FCL (System.Object, System.String, collections, etc.).

12. Generics in .NET

- Generic Classes: Define with type placeholders (e.g., List<T>).
- Benefits: Type safety, performance, reusability.
- Also includes: Generic Methods, Interfaces, Delegates, Constraints.

13. Events and Delegates

- Delegate: Type-safe function pointer.
- Event: Wrapper around delegate for pub-sub model.

Understanding .NET: Concepts, Compilation, and Runtime

- Flow: Define delegate -> expose event -> subscribe -> raise event.

14. Global Assembly Cache (GAC)

- GAC: Machine-wide store for shared assemblies in .NET Framework.
- Requires strong naming.
- Replaced by NuGet/local deployment in .NET Core.

15. .NET Core Application Flow

1. Source Code -> Roslyn -> IL.
2. Assembly (.dll/.exe) -> contains IL + metadata.
3. CoreCLR -> loads assemblies.
4. RyuJIT -> compiles IL to native code.
5. CPU executes native instructions.

Deployment:

- Framework-dependent: Uses shared runtime.
- Self-contained: Includes own runtime.

16. In-Proc vs Out-Proc and DLL Types

- In-Proc: DLL runs in same process (faster).
- Out-Proc: DLL runs in separate process (isolated).
- Shared DLLs: In GAC or global cache.
- Private DLLs: In app's local folder.

17. Garbage Collector (GC)

- GC: Automatic memory manager in CLR.
- Responsibilities: Allocate, reclaim, compact memory.
- Uses generational, mark-sweep, compacting algorithm.

Understanding .NET: Concepts, Compilation, and Runtime

18. Do's and Don'ts of GC

Do's:

- Use managed objects.
- Implement IDisposable.
- Use 'using' blocks.

Don'ts:

- Don't call GC.Collect() manually.
- Don't rely on finalizers for cleanup.
- Don't hold unnecessary references.

19. GC Algorithm and Generations

- Generations:

- Gen 0: New objects.
- Gen 1: Survived one collection.
- Gen 2: Long-lived.
- LOH: Large objects.

- Phases:

- Mark: Identify reachable objects.
- Sweep: Remove unreachable.
- Compact: Rearrange memory.

20. Finalize Queue and Generation Queue

- Finalize Queue: Holds objects with finalizers.
- Freachable Queue: For finalization thread.
- Generation Queue: Tracks object age and promotes survivors.

21. Shuffling and Gen Shuffling

- Shuffling: Moving surviving objects between generations.

Understanding .NET: Concepts, Compilation, and Runtime

- Gen Shuffling:
- Gen 0 -> Gen 1 -> Gen 2.
- LOH: Collected during full GC.
- Compaction: Reduces fragmentation.