.NET Platform Guide

.NET is a general purpose development platform. It can be used for any kind of app type or workload where general purpose solutions are used. It has several key features that are attractive to many developers, including automatic memory management and modern programming languages, that make it easier to efficiently build high-quality applications. .NET enables a high-level programming environment with many convenience features, while providing low-level access to native memory and APIs.

C#, F# and Visual Basic are popular languages that target and rely on the .NET platform. The .NET languages are known for key features such as their asynchronous programming model, language-integrated query, generic types and type system reflection. The languages also provide great options for both object-oriented and functional programming paradigms.

There is great diversity across these languages, in philosophy and syntax, but also symmetry provided by a shared type system. This type system is provided by the underlying runtime environment. .NET was designed around the idea of a "common language runtime" that could support the requirements of diverse languages -- for example, dynamic and statically typed languages -- and enable interoperability between them. For example, it's possible to pass a collection of `People` objects between languages with no loss in semantics or capability.

Multiple .NET implementations and products are available, based on open .NET Standards that specify the fundamentals of the platform. They are separately optimized for different application types (for example, desktop, mobile, gaming, cloud) and support many chips (for example, x86/x64, ARM) and operating systems (for example, Windows, Linux, iOS, Android, macOS). Open source is also an important part of the .NET ecosystem, with multiple .NET implementations and many libraries available under OSI-approved licenses.

- Learn about C#
- Learn about F#
- Browse the .NET API Library
- Introduction to the Common Language Runtime

Fundamentals

Multi-Language -- .NET provides a well-defined type system, file formats, runtime, framework and tools that can be used by multiple languages, both for their own execution and also to interoperate with other languages using those same components of .NET as their shared currency.

Managed Memory -- .NET automatically manages memory for you via a garbage collector. It ensures that you always reference live objects, guaranteeing that you avoid nasty problems like buffer overruns and access violations. This includes array bounds checking.

Type Safety -- The primary .NET model for functionality and memory representation is "types". Types define shape and optionally behavior. The runtime ensures that calling code can only operate on types according to their definition and specified visibility of members, providing consistent, reliable and secure results.

Features

User-defined Value Types -- Value types are a useful category of types since they offer the semantic of "pass by value" instead of "pass by reference", as is the case for classes. Value types are most obviously usefully for numeric data. .NET enables value types for both primitive types, like integers, and user-defined types.

Generic types -- Generic types are types with one or more type parameters that can be specified on a per-instantiation basis. This is useful for many types, which otherwise would expose contents as the Object type or require multiple type definitions. For example, a given instantiation of a collection type can be made specific to People, GPS locations or strings.

Reflection -- .NET defines a metadata format that describes the types within a binary. The reflection subsystem uses this data, exposing APIs for both reading and instantiating types at runtime. This facility is very useful for dynamic scenarios where it is not convenient to know the exact implementation of a program ahead of time.

Flexible code generation -- .NET does not prescribe a specific approach to transforming .NET binaries into machine code. Many approaches have been used successfully, including interpretation, just-in-time (JIT) compilation, ahead-of-time (AOT) compilation with JIT fallback and AOT compilation with no JIT fallback. Each of these strategies can be valuable and there are opportunities for using them together.

Cross-platform -- .NET was intended to be cross platform from its inception. The binary format and the instruction set are operating system, CPU and pointer-size agnostic. A given .NET binary built in 2000 to run on a 32-bit Windows machine can run on the ARM64 iOS device on 2016 without modification.

Open source

The .NET Core and Mono implementations of .NET are open source, using the MIT license. Documentation uses the Creative Commons CC-BY license. .NET Core and Mono are sponsored by Microsoft and have many contributors from the community.

These general purpose runtimes can be used as the basis of academic research or teaching/learning or commercial products. Their open nature also means that anyone can contribute back to the upstream product code given a bug or the desire for a new feature.

Projects

- CoreCLR - .NET runtime, used by .NET Core.
- Mono - .NET runtime, used by Xamarin and others.
- CoreFX - .NET class libraries, used by .NET Core and to a degree by Mono via source sharing.
- Roslyn - C# and Visual Basic compilers, used by most .NET platforms and tools. Exposes APIs for reading, writing and analyzing source code.
- F# - F# compiler.
- Xamarin SDK - Tools and libraries needed to write Android, iOS and macOS in C# and F#.

Standardized

.NET is specified via open ECMA standards that outline its capabilities and that can be used to make a new implementation. There are other .NET implementations, with Mono and Unity being the most popular beyond the Microsoft ones.