.NET 6

What's new

VISMA | DataLøn

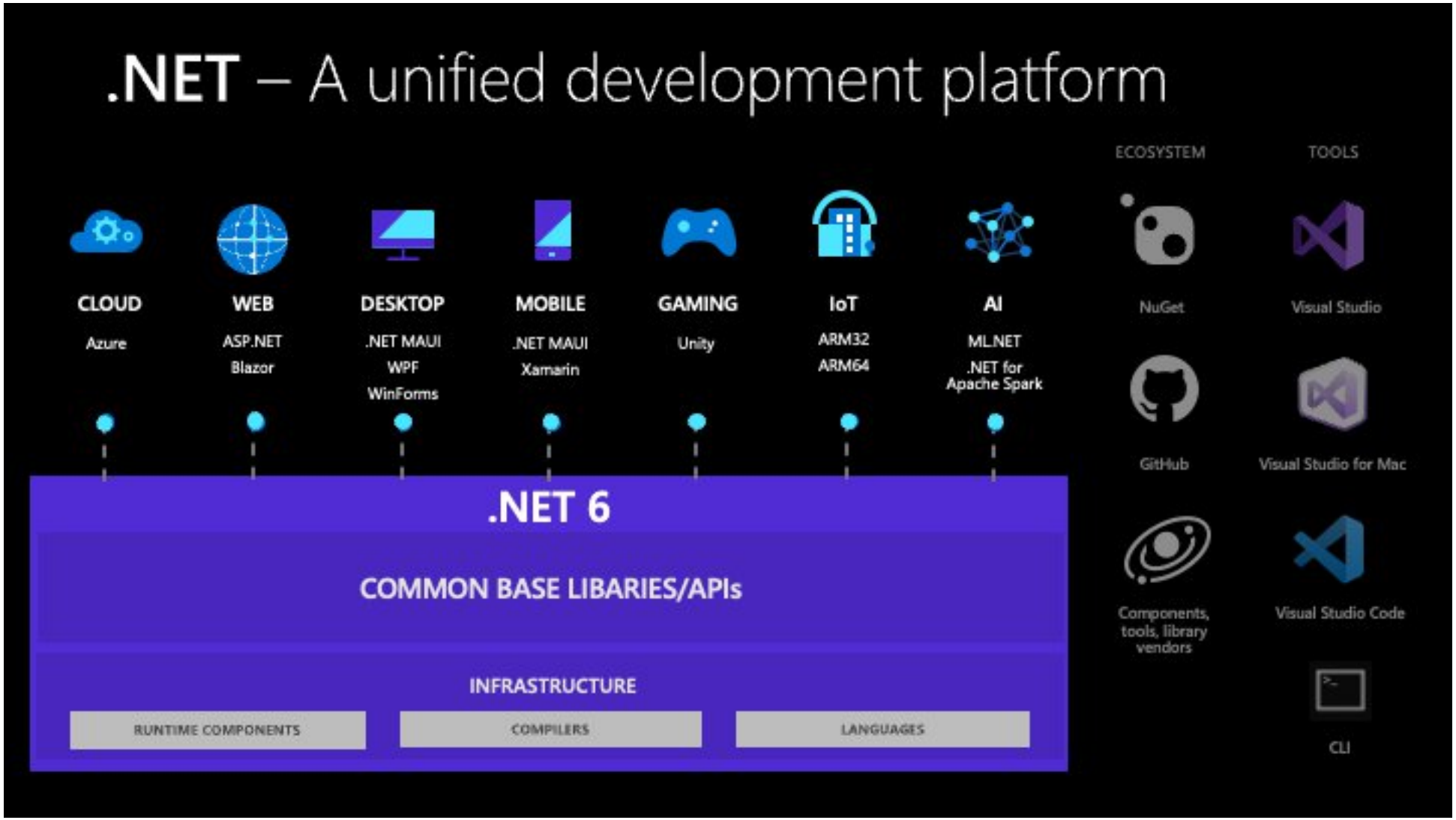
Highlights

- Simplified development
- Better performance
- Visual Studio 2022
- Hot reload
- C# 10
- F# 6

Highlights

- Minimal APIs
- System.Text.Json APIs
- Blazor
- Single-file apps
- Arm64 support
- Unified Platform - MAUI - Preview
- HTTP/3 - Preview
- LTS - 3 years

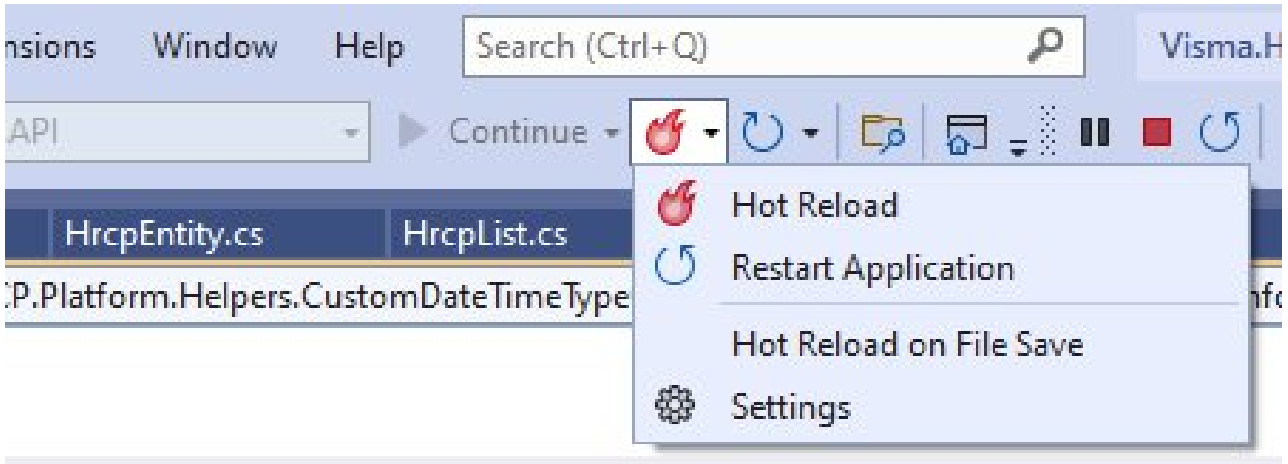
Unified and extended platform



Hot Reload

- Enabled to make a wide variety of code edits to a **running** application.
- Available through both the **dotnet watch** CLI tool and **Visual Studio 2022**.

```
PS C:\Users\rich\mvcapp> dotnet watch
watch : Hot reload enabled. For a list of supported edits, see https://aka.ms/dotnet/hot-reload. Press "Ctrl + R" to res
tart.
watch : Building...
  Determining projects to restore...
  All projects are up-to-date for restore.
  mvcapp -> C:\Users\rich\mvcapp\bin\Debug\net6.0\mvcapp.dll
watch : Started
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7072
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5036
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\Users\rich\mvcapp\
watch : File changed: C:\Users\rich\mvcapp\Controllers\HomeController.cs.
watch : Hot reload of changes succeeded.
watch : File changed: C:\Users\rich\mvcapp\Views\Home\Index.cshtml.
watch : Hot reload of changes succeeded.
```



Performance - FileStream

- **System.IO.FileStream** type has been almost entirely rewritten for **.NET 6**
- Async file IO few times **faster** for some cases
- Async file IO on Windows is not using **blocking API** anymore
- New stateless and offset-based APIs for **thread-safe** file IO.
- New APIs for specifying file preallocation size.
- **Concurrent** reads and writes.
- **Scatter/Gather IO** - Allows reducing the number of expensive sys-calls by passing multiple buffers in a single sys-call.
- **FileStream.Position** is not synchronized with the OS anymore.
- **FileStream.Position** is updated after the async operation has completed, not before it was started.

Blazor

- **Razor** components can be dynamically-rendered from **JavaScript** (JS) for existing JS apps.
- **Custom elements** usable with any web framework (React, Angular) - Experimental.
- **WebAssembly AOT** compilation for **Blazor WebAssembly (Wasm)** apps, as well as support for runtime relinking and native dependencies.

```
<my-counter increment-amount={incrementAmount}></my-counter>
```

Security

- Added support for **OpenSSL 3**.
- .NET 6 requires **OpenSSL 1.1** or higher.
- Preferred the highest installed version of **OpenSSL**.
- Initial implementations of **W^X** ("write xor execute").
- Initial implementations of **Intel Control-flow enforcement technology** (CET)

Single-file Apps

- In-memory **single file apps** have been enabled for **Windows** and **macOS**. In **.NET 5**, this deployment type was limited to **Linux**.
- Enabled to publish a **single-file binary** that is both deployed and launched as a **single file**.
- **Single files apps** no longer extract any **core runtime assemblies** to temporary directories.
- This expanded capability is based on a building block called **“superhost”**.
- **“apphost”** is the executable that launches your application in the non-single-file case, like **myapp.exe** or **./myapp**.
- Compression support.

Libraries APIs

- **WebSocket** Compression.
- **Socks** proxy support.
- **Microsoft.Extensions.Hosting** - ConfigureHostOptions API.
- **Microsoft.Extensions.DependencyInjection** - CreateAsyncScope APIs.
- **Microsoft.Extensions.Logging** - compile-time source generator.
- **System.Linq**
  - Enumerable support for Index and Range parameters.
  - TryGetNonEnumeratedCount.
  - DistinctBy/UnionBy/IntersectBy/ExceptBy.
  - MaxBy/MinBy.
  - Chunk.
  - FirstOrDefault/LastOrDefault/SingleOrDefault overloads taking default parameters.
  - Zip overload accepting three enumerables.

Libraries APIs

- **PriorityQueue**.
- Faster handling of **structs** as **Dictionary** values.
- New **DateOnly** and **TimeOnly** structs.
- Perf improvements to **DateTime.UtcNow**.
- Support for both **Windows** and **IANA** time zones on all platforms.
- Improved time zone display names.
- Improved support for **Windows ACLs**.
- **HMAC** one-shot methods.
- **DependentHandle** is now **public**.
- Portable thread pool.

C# 10 - What's new

- Record **structs.**
- Improvements of **structure** types.
- **Interpolated** string handler.
- Global **using** directives.
- File-scoped **namespace** declaration.
- Extended property **patterns.**
- **Lambda** expression improvements.
- Constant **interpolated** strings.

C# 10 - What's new

- Record types can **seal ToString.**
- Assignment and declaration in same deconstruction.
- Improved definite assignment.
- Allow **AsyncMethodBuilder** attribute on methods.
- **CallerArgumentExpression** attribute diagnostics.
- Enhanced **#line** pragma.
- Generic **attributes.**

C# 10 - Record structs

- **Record struct** and **readonly records struct** declarations.
- Reference type record using **record class.**

```
public readonly record struct Point(double X, double Y, double Z);

public record struct Point
{
    public double X {  get; init; }
    public double Y {  get; init; }
    public double Z {  get; init; }
}

public record class Point
{
    public double X {  get; init; }
    public double Y {  get; init; }
    public double Z {  get; init; }
}
```

C# 10 - Improvements of structure types

- Instance **parameterless constructor**.
- Field or property **initializer**.
- A left-hand operand of the **with** expression can be of any **structure** type or an **anonymous** (reference) type.

C# 10 - Improvements of structure types

```
public readonly struct Measurement
{
	public double Value { get; init; }
		public string Description { get; init; } = "Ordinary measurement";

	public Measurement()
	{
		Value = double.NaN;
		Description = "Undefined";
	}

	public Measurement(double value, string description)
	{
		Value = value;
		Description = description;
	}
}
```

```
Measurement m1 = new Measurement();
Measurement m2 = m1 with { Value = "Code and coffee" }
```

C# 10 - Interpolated string handler

- An **interpolated string handler** is a custom type that converts the **interpolated string** into a **string**
- The compiler checks if the **interpolated string** is assigned to a type that satisfies the **interpolated string handler pattern**
- System.Runtime.CompilerServices.InterpolatedStringHandlerAttribute

C# 10 - Interpolated string handler

```
[InterpolatedStringHandler]
public ref struct LogInterpolatedStringHandler
{
    StringBuilder builder; // Storage for the built-up string

    public LogInterpolatedStringHandler(int literalLength, int formattedCount)
    {
        builder = new StringBuilder(literalLength);
    }

    public void AppendLiteral(string s)
    {
        builder.Append(s);
    }

    public void AppendFormatted<T>(T t)
    {
        builder.Append(t?.ToString());
    }

    internal string GetFormattedText() => builder.ToString();
}
...
public void LogMessage(LogLevel level, LogInterpolatedStringHandler builder)
{
    Console.WriteLine(builder.GetFormattedText());
}
...
logger.LogMessage(LogLevel.Error, $"Error Level. CurrentTime: {time}. This is an error. It will be printed.");
```

C# 10 - Global using directives

- **Global** modifier on **using** directives instructs the compiler that the directive applies to all source files in the compilation.
- Used typically for all source files in a project.
- Can appear at the beginning of any source code file.
- Must appear before:
  - All **using** directives without the **global** modifier.
  - All **namespace** and **type** declarations in the file.

```
global using static System.Math;
```

C# 10 - File-scoped namespace declaration

- Enabled to declare that all types in a file are in a single **namespace**.
- Saves both horizontal and vertical space.
- Can't include additional **namespace** declarations.

```
using System;

namespace SampleFileScopedNamespace;

class SampleClass { }

interface ISampleInterface { }

struct SampleStruct { }

enum SampleEnum { a, b }

delegate void SampleDelegate(int i);

namespace AnotherNamespace; // Not allowed!

namespace ANestedNamespace // Not allowed!
{
  // declarations...
}
```

C# 10 - Extended property patterns

- Allowed to reference nested properties or fields within a **property pattern.**

```
// C# 8.0 and later
if (e is MethodCallExpression { Method: { Name: "MethodName" } })

// C# 10.0 and later
if (e is MethodCallExpression { Method.Name: "MethodName" })
```

C# 10 - Lambda expression improvements

- Natural typed **lambda expression**, where the compiler can infer a delegate type from the **lambda expression** or method group.
- May declare a **return type** when the compiler can't infer it.
- **Attributes** can be applied.
- More similar to **methods** and **local functions**.
- Easier to use without declaring a variable of a **delegate type**.

C# 10 - Lambda expression improvements

Natural type

```
Func<string, int> parse = (string s) => int.Parse(s);

var parse = (string s) => int.Parse(s); // Func<string, int>
Delegate parse = (string s) => int.Parse(s); // Func<string, int>
var parse = s => int.Parse(s); // ERROR: Not enough type info in the lambda
```

Declared return type

```
var choose = (bool b) => b ? 1 : "two"; // ERROR: Can't infer return type

var choose = object (bool b) => b ? 1 : "two"; // Func<bool, object>
```

Attributes

```
Func<string, int> parse = [Example(1)] (s) => int.Parse(s);

var choose = [Example(2)][Example(3)] object (bool b) => b ? 1 : "two";
```

C# 10 - Constant interpolated strings

- **Const strings** may be initialized using **string interpolation** if all the placeholders are themselves **constant strings**.
- Placeholder expressions can't be **numeric constants** because those constants are converted to **strings** at run time.

```
const string language = "C# 10";
const string greeting = $"Hello from {language}.";
```

C# 10 - Record types can seal ToString

- **Sealed** modifier for overridden **ToString** in a **record type**.
- Prevents the compiler from synthesizing a **ToString** method for any derived record types.

```
record Point (double X, double Y)
{
        public override sealed string ToString() => $"({X}, {Y})";
}
```

C# 10 - Assignment and declaration in same deconstruction

- Removes a restriction, where deconstruction could:
  - **assign** all values to existing variables
  - or **initialize** newly declared variables

```
// Initialization:
(int x, int y) = point;

// assignment:
int x1 = 0;
int y1 = 0;
(x1, y1) = point;

// C# 10:
int x = 0;
(x, int y) = point;
```

C# 10 - Improved definite assignment

- Improvements in scenarios where **definite assignment** and **null-state analysis** produced **warnings** that were **false positives**.

```
string representation = "N/A";

if ((c != null && c.GetDependentValue(out object obj)) == true)
{
   representation = obj.ToString(); // undesired error
}

// Or, using ?.
if (c?.GetDependentValue(out object obj) == true)
{
   representation = obj.ToString(); // undesired error
}

// Or, using ??
if (c?.GetDependentValue(out object obj) ?? false)
{
   representation = obj.ToString(); // undesired error
}
```

C# 10 - CallerArgumentExpression attribute diagnostics

- System.Runtime.CompilerServices.CallerArgumentExpressionAttribute to specify a parameter that the compiler replaces with the text representation of another argument.
- Enables libraries to create more specific diagnostics.

```
public static void Validate(bool condition, [CallerArgumentExpression("condition")] string? message=null)
{
    if (!condition)
    {
        throw new InvalidOperationException($"Argument failed validation: <{message}>");
    }
}

Validate(1 != /* test */ 1);

// Exception: Argument failed validation: <1 != /* test */ 1>
```

C# 10 - Generic attributes

- Preview feature.
- Allowed declaration of a **generic class** whose base class is **System.Attribute**.
- Previously needed to create an **attribute** that takes a **Type** as its constructor parameter.

C# 10 - Generic attributes

```
public class TypeAttribute : Attribute
{
   public TypeAttribute(Type t) => ParamType = t;

   public Type ParamType { get; }
}

[TypeAttribute(typeof(string))]
public string Method() => default;

// C# 10:

public class GenericAttribute<T> : Attribute { }

[GenericAttribute<string>()]
public string Method() => default;

public class GenericType<T>
{
   [GenericAttribute<T>()] // Not allowed! generic attributes must be fully closed types.
   public string Method() => default;
}
```

Demo

Q&A

References

- What's new in .NET 6
- Announcing .NET 6
- Performance Improvements in .NET 6
- What's new in C# 10

Thank you

You are great ;)