What's New In .NET 8 C# 12

Jim Wilcox, The Granite State Hacker,
Architect II, Modern Applications
2019-2024 Microsoft MVP – Developer Technologies

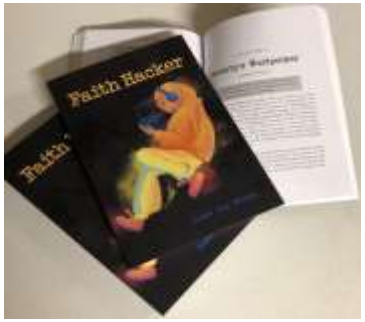
Agenda

- Welcome & Introductions
- Presentation Overview
- What’s new in .NET 8
- What’s new in C# 12

Introduction

<GRANITE STATE />
{CODE CAMP}

Microsoft® MVP Most Valuable Professional



Introductions: Who I am

- Jim Wilcox, “The Granite State Hacker”
  - Dad of two wicked smaht kids
  - Architect, Modern Applications
  - Author, “Faith Hacker” (just released!)
- Co-Founder of
  - Granite State (NH) Microsoft 365 Users Group
  - Granite State (NH) .NET Devs Group
- Co-Organizer of
  - CollabDays New England,
  - Granite State Code Camp
  - NH Cybersecurity Symposium
- Microsoft MVP – Developer Technologies (2019-)

WPDevs

Insight. Digital Innovation

Introductions: Who are you?
Quick check… are you….

- A developer?
  - C# developer?
  - HTML/CSS/JS developer?
- An enterprise architect looking for ideas?
- An app-curious user?

- Familiar with?
  - Mobile App Development?
  - Universal App Development?
  - Web App Development?

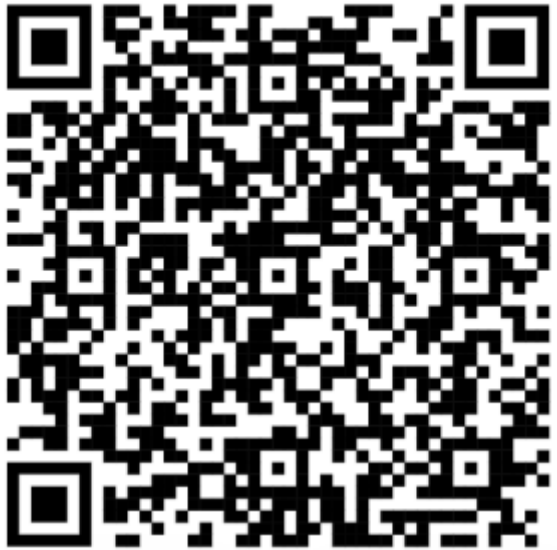
Overview

- Purpose
  - Find out what to prompt ChatGPT for around the exciting new releases of .NET and C#
- Scope
  - Relatively high-level, introduction to new concepts and terms

.NET 8 Overview

- https://learn.microsoft.com/en-us/dotnet/core/whats-new/dotnet-8

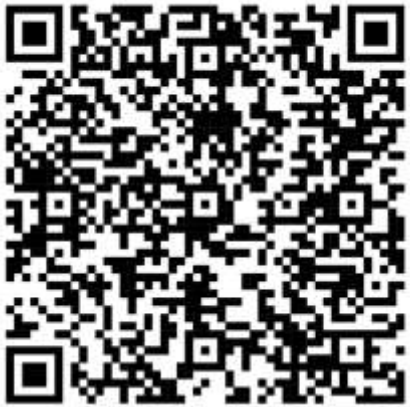
- LTS
- Performance!
- Aspire



.NET 8: Aspire

- https://learn.microsoft.com/en-us/dotnet/aspire

- .NET Aspire is an opinionated, cloud-ready stack for building observable, production ready, distributed applications. .NET Aspire is delivered through a collection of NuGet packages that handle specific cloud-native concerns, and is available in preview for .NET 8.
- Aspects are roughly comparable to
  - Spring Boot (Java)
  - Delphi (Pascal)
  - Electron (Javascript)



.NET 8: Aspire (continued)

- Orchestration
- Components
- Tooling
  - CLI

.NET 8: ASP.NET Core

- Improved startup
- Debugging experience
- Blazor enhancements
- Metrics

.NET 8: Core Libs

- Serialization
  - Time abstraction (“pretend time” for test scenarios)
- UTF8 Improvements (IUtf8SpanFormattable, perf)
- Randomness (GetItem<t>, Shuffle<t>)
- Performance-focused types (FrozenDictionary, FrozenSet)
- System.Numerics, System.Runtime.Intrinsics (perf improvements)
- Data validation (api / config options)
- Metrics (Tag metrics with key-value pairs)
- Cryptography (SHA-3)
- Networking (Https proxy)
- Stream based ZipFile methods (performance improvement)
- Keyed Dependency Injection

.NET 8: Extension Libs

- Windows Forms
- WPF
- Entity Framework Core
- ML.NET
- Azure SDK

.NET 8: Workloads

- Android
- iOS
- macOS
- tvOS
- watchOS

.NET 8: AOT

- Improved compilation to native code results in smaller target file sizes, improves performance

.NET 8: Performance

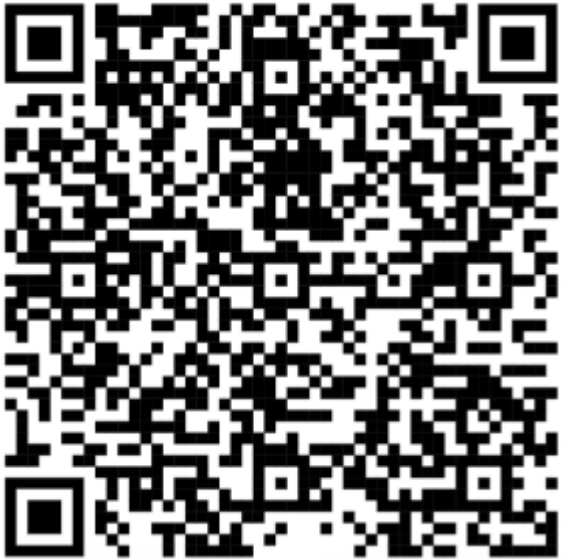
- Updates to
  - JIT compilation
  - Dynamic PGO
  - Garbage Collection (runtime-scalable memory limits)

.NET 8: Tools

- Visual Studio
- Dotnet CLI
- MSBuild
- NuGet
- SDK

C# 12: Overview

- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-12



C# 12: Primary Constructors

```
// A class with a primary constructor
public class Person(string name, int age)
{
    // A property initialized with the primary constructor parameter
    public string Name { get; init; } = name;

    // A method that uses the primary constructor parameter
    public void Greet()
    {
        Console.WriteLine($"Hello, my name is {name} and I am {age} years old.");
    }
}

// A class that inherits from a class with a primary constructor
public class Student(string name, int age, string major) : Person(name, age)
{
    // A property initialized with the primary constructor parameter
    public string Major { get; init; } = major;

    // A method that overrides the base class method
```

C# 12: Collection Expressions

```
// Create an array of integers
int[] array = [1, 2, 3, 4, 5];

// Create a list of strings
List<string> list = ["one", "two", "three"];

// Create a span of characters
Span<char> span = ['a', 'b', 'c', 'd', 'e'];

// Create a jagged array of integers
int[][] jagged = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];

// Create a list of lists of strings using the spread operator
List<string> list1 = ["red", "green", "blue"];
List<string> list2 = ["apple", "banana", "orange"];
List<List<string>> listOfLists = [list1, ...list2, "grape"];
```

C# 12: Inline Arrays

```
// A struct with an inline array of 10 integers
[System.Runtime.CompilerServices.InlineArray(10)]
public struct IntArray
{
    private int _element0; // The first element of the inline array
}

// A struct with an inline array of 5 strings
[System.Runtime.CompilerServices.InlineArray(5)]
public struct StringArray
{
    private string _element0; // The first element of the inline array
}

// A method that creates and returns an inline array of integers
public IntArray CreateIntArray()
{
    // Create a span over the inline array
    Span<int> span = System.Runtime.InteropServices.MemoryMarshal.CreateSpan(ref _element0, 10);

    // Initialize the span with some values
```

C# 12: Optional params in lamdas

```
// A method that takes a delegate with two parameters
public static void DoSomething(Func<int, int, int> func)
{
   // Invoke the delegate with some values
   Console.WriteLine(func(10, 20));
}

// A lambda with an optional parameter
var addWithDefault = (int x, int y = 5) => x + y;

// Call the method with the lambda
DoSomething(addWithDefault); // 15

// Call the method with the lambda and specify the optional parameter
DoSomething((x, y) => addWithDefault(x, 10)); // 20
```

C# 12: ref-only params

```
// A method that takes a ref-only parameter
public static void Print(ref readonly int x)
{
   // You can read the parameter value
   Console.WriteLine(x);

   // But you can't assign a new value to it
   // x = 10; // Compile-time error
}

// A method that calls the Print method
public static void Main()
{
   // Declare and initialize a variable
   int y = 5;

   // Pass the variable by reference to the Print method
   Print(ref y);
}
```

C# 12: alias any type

```
// Create an alias for a tuple type
using Point = (int x, int y);

// Use the alias to declare a variable
Point p = (10, 20);

// Use the alias to define a method
void PrintPoint(Point p)
{
   Console.WriteLine($"({p.x}, {p.y})");
}
```

C# 12: Experimental attribute

```
// Mark a class as experimental
[System.Diagnostics.CodeAnalysis.Experimental("MyFeature")]
public class MyClass
{
    // Mark a method as experimental
    [System.Diagnostics.CodeAnalysis.Experimental("MyMethod")]
    public void MyMethod()
    {
        // Do something
    }
}

// Use the experimental class and method
public class Program
{
    public static void Main()
    {
        // Warning: 'MyClass' is for evaluation purposes only and is subject to change or removal in future updates.
        var myClass = new MyClass();

        // Warning: 'MyClass.MyMethod()' is for evaluation purposes only and is subject to change or removal in future updates.
        myClass.MyMethod();
    }
}
```

C# 12: Interceptors

```
// A class with a method to intercept
public class Example
{
    public string GetText(string text)
    {
        return $"{text}, World!";
    }
}

// A static class with an interceptor method
public static class Interceptor
{
    // The attribute specifies the type and method name to intercept
    [InterceptsLocation(typeof(Example), "GetText")]
    public static string InterceptGetText(string text)
    {
        // The interceptor method can modify the input or output of the original method
        return $"Intercepted: {text.ToUpper()}";
    }
}
```

C# 12: nameof enhancements

```
// A class with an instance property
public class Example
{
    public string Text { get; set; }
}

// A static method that uses the nameof operator to get the name of the instance property
public static void Print()
{
    // You can use the nameof operator with the type name and the dot operator
    Console.WriteLine(nameof(Example.Text)); // Text
}
```

Thank you!