Leveraging the Power of Prebuilt Attributes in Your Code

Jason Roberts
.NET MVP

@robertsjason dontcodetired.com

Overview

- Understanding attribute constructors and properties
- Controlling the debugging experience
- Marking code as deprecated
- Conditionally compiling code
- Setting assembly level metadata
- Exposing internal code to other assemblies
- Applying attributes to return values
- Specifying data validation
- Indicating that a class can be serialized

[ ]

[Display]

[Display(...)]

Understanding Attribute Constructors and Properties

Required constructor parameters

Optional constructor parameters

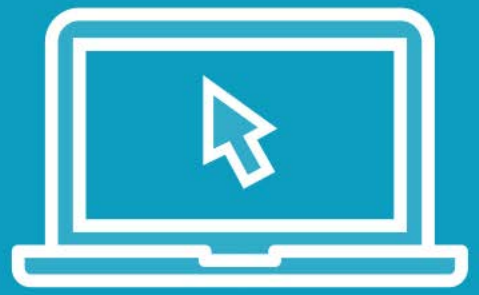
Additional attribute properties

Understanding Attribute Constructors and Properties

```
[Display(42.5)]

[Display(42.5, 0.5)]

[Display(42.5, precision: 0.5)]

[Display(value: 42.5, precision: 0.5)]

[Display(42.5, 0.5, Prefix = "X")]

[Display(42.5, 0.5, Prefix = "X", Postfix = "Y")]

[Display(42.5, Postfix = "Y")]
```

Attributes can also define multiple constructor overloads as with regular class definitions

Demo



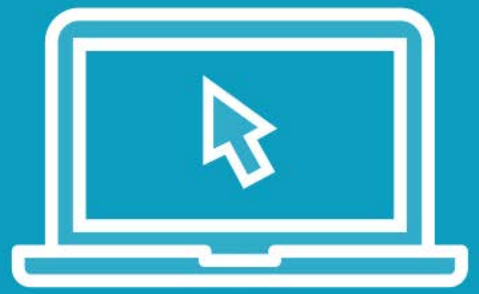
Controlling the debugging experience

[DebuggerDisplay]

[DebuggerTypeProxy]

[DebuggerBrowsable]

Demo

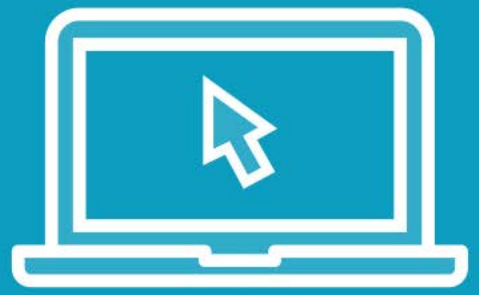


Marking code as deprecated

[Obsolete]

Compilation warnings/errors

Demo



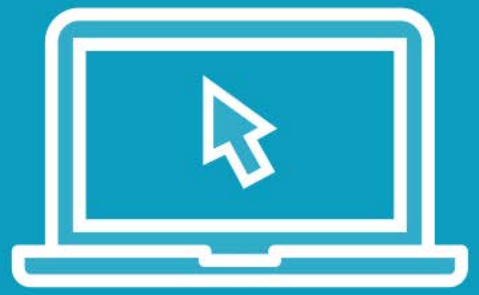
Conditionally compiling code

[Conditional]

Conditional compilation symbols

DEBUG

Demo



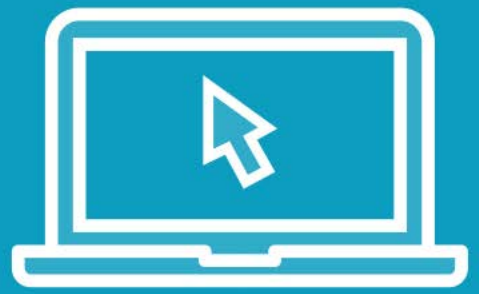
Applying attributes at assembly level

E.g. assembly level metadata

- Title
- Version
- Copyright
- Etc.

assembly: keyword

Demo

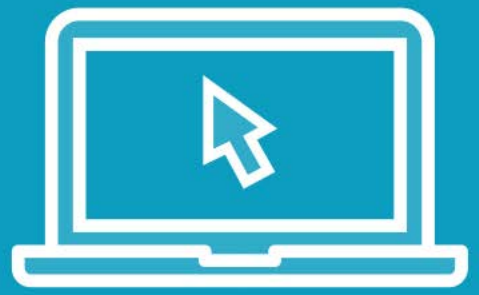


Exposing internal code

[InternalsVisibleTo]

Use in testing scenarios

Demo



Applying attributes to return values

E.g. Azure Functions bindings

return: keyword

Specifying Data Validation

```
public class Person
{

    public string Name { get; set; }



    public string EmailAddress { get; set; }

    public int AgeInYears { get; set; }
}
```

Specifying Data Validation

```
public class Person
{
    [Required]

    public string Name { get; set; }



    public string EmailAddress { get; set; }


    public int AgeInYears { get; set; }
}
```

Specifying Data Validation

```
public class Person
{
    [Required]
    [StringLength(100)]
    public string Name { get; set; }



    public string EmailAddress { get; set; }


    public int AgeInYears { get; set; }

}
```

Specifying Data Validation

```
public class Person
{
    [Required]
    [StringLength(100)]
    public string Name { get; set; }

    [Required]
    [EmailAddress]
    public string EmailAddress { get; set; }


    public int AgeInYears { get; set; }

}
```

Specifying Data Validation

```
public class Person
{
    [Required]
    [StringLength(100)]
    public string Name { get; set; }

    [Required]
    [EmailAddress]
    public string EmailAddress { get; set; }

    [Range(0, 120)]
    public int AgeInYears { get; set; }
}
```

System.ComponentModel.DataAnnotations;

http://bit.ly/aspvalidation

Indicating That a Class Can Be Serialized

```
public class Person
{
    public string FirstName { get; set; }

    public int AgeInYears { get; set; }



    public int Id;
}
```

Indicating That a Class Can Be Serialized

```
[Serializable]
public class Person
{
    public string FirstName { get; set; }

    public int AgeInYears { get; set; }



    public int Id;
}
```

Indicating That a Class Can Be Serialized

```
[Serializable]
public class Person
{
    public string FirstName { get; set; }

    public int AgeInYears { get; set; }


    [NonSerialized]
    public int Id;
}
```

Summary

[Display(42.5, precision: 0.5, Prefix = "X")]

[DebuggerDisplay]

[Obsolete]

[Conditional]

[assembly: AssemblyCopyright(...)]

[InternalsVisibleTo]

[return: ...]

[Required] [StringLength(100)]

[Serializable] [NonSerialized]

Next:

Gaining Flexibility and Expanding Your Solutions with Custom Attributes