METOVA

11/17/2015

Introduction to C# 6

...

.NET Development Meeting

Background

Released July 20, 2015

Shipped with Visual Studio 2015

and .NET Framework 4.6

Language features

- Auto-property initializer
- Get-only properties
- Exception filters
- Nameof expression
- Using static
- Expression bodied members
- Await in catch / finally blocks
- String interpolation
- Null conditional operator

Auto-property Initializers

and

Get-only auto-properties

Gives us the ability to initialize auto-properties inline

Previously forced to use field-backed properties

Auto-property Initializer - Example

Old busted

```
private int count = 0;
public int Count
{
    get
    {
        return count;
    }
    set
    {
        count = value;
    }
}
```

New hotness

```
public int Count { get; set; } = 0;
```

Usage

Saves unnecessary lines of code and clutter.

Avoids tedious property initialization in constructors

Exception filters

Gives us the ability to enter catch blocks based on conditions without jacking the stack

“Exception filters are preferable to catching and rethrowing because they leave the stack unharmed.”[1]

Exception filters - Example

New hotness

```
try
{
    //do some foo
}
catch(ApplicationException ex) when(ex.Message == "Bobbeh!")
{
    WriteToBobbyLog(ex);
}
catch(ApplicationException ex) when(ex.Message == "Propane")
{
    WriteToPropaneLog(ex);
}
```

Usage

Catching special exception cases for logging or other purposes

Avoiding the overhead of rethrowing exceptions

Nameof expression

Gives us the ability to retrieve the name of the given variable

Nameof expression - Example

Old busted

```
public void MyMethod(string input)
{
    if(input == null)
        throw new ArgumentNullException("input");

    doThing();
}
```

Nameof expression - Example

New hotness

```
public void MyMethod(string input)
{
    if(input == null)
        throw new ArgumentNullException(nameof(input));

    doThing();
}
```

Usage

ArgumentNullExceptions and PropertyChanged events which require name of the object

Makes renaming things *safer* (Less reliance on plain literals)

Eliminates need for Reflection in some cases

Using static

Gives us the ability to import members from a static class

Previously we had to fully qualify members on static classes

Using static - Example

Old busted

```
public void DoSomething()
{
   Console.WriteLine("Swiggity swooty!");
}
```

New hotness

```
using static System.Console;
public void DoSomething()
{
   WriteLine("Swiggity swooty!");
}
```

Usage

Shortening code where a large number of static class methods are used

Expression bodied members

Gives us the ability to write method and property bodies as expressions

Expression bodied members - Example

Old busted

```
public IEnumerable<int> ListOfNumbers
{
    get
    {
        return Enumerable.Range(0,100);
    }
}
```

New hotness

```
public IEnumerable<int> ListOfNumbers =>
     Enumerable.Range(0,100);
```

Usage

Making method and property bodies concise

Await in catch / finally blocks

Gives us the ability to us await in catch and finally blocks

Await in catch block - Example

Old busted

```
Exception exception = null;
try
{
    // Do some foo
}
catch (Exception ex)
{
    exception = ex;
}
if (exception != null)
  await AsyncLoggingService.Log(ex);
```

New hotness

```
try
{
    // Do some foo
}
catch (Exception ex)
{
    await AsyncLoggingService.Log(ex);
}
```

Usage

Logging to asynchronous logging service or other async operations in the catch block

String interpolation

Gives us the ability to format strings by variable references

String interpolation - Example

Old busted

```
public string Red = "FF";
public string Green = "99";
public string Blue = "00";

public string Color
{
    get
    {
        return String.Format("{0}{1}{2}", Red, Green, Blue);
    }
}
```

String interpolation - Example

New hotness

```
public string Red = "FF";
public string Green = "99";
public string Blue = "00";

public string Color
{
    get
    {
        return $"{Red}{Green}{Blue}";
    }
}
```

Usage

Increases clarity for templated strings

More concise than string.format

Null conditional operator

Gives us conditional access to members

Null conditional operator - Example

Old busted

```
int count = 0;
if(response != null && response.Results != null)
{
    count = response.Results.Count;
}
```

New hotness

```
int count = response?.Results?.Count ?? 0;
```

Usage

Reducing cluttered nesting

Concerns

Could encourage returning of null values in an incorrect context

References

[1] - New Language Features in C# 6 https://github.com/dotnet/roslyn/wiki/New-Language-Features-in-C%23-6