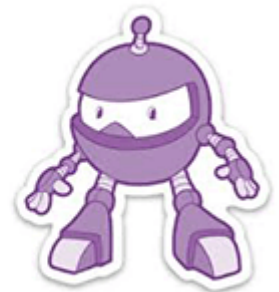


Dot Net Core 3/3.1

New features for programmers



Peter "Shawty" Shaw



Intro

- Who am I?
 - For those who've not met me before, I normally go by the name of "Shawty".
 - I'm a Microsoft Community leader, one of the management team for the Linked .NET (LiDNUG) users group on Linked-In and a published pluralsight and Syncfusion author on various programming topics.

Intro

- Who am I?
 - Please note: I am predominantly deaf and wear hearing aids so if you really have a burning question to ask you'll probably need to do something more than shouting up to attract my attention.
 - You can stalk me at:
 - Twitter @shawty_ds
 - Email shawty.d.ds@googlemail.com

Intro

- What are we talking about tonight?
 - Some of the MANY new features that have appeared in C#8 and Dot Net core that will make your job writing .NET code easier (Hopefully)
 - I'll do some code demos and some slides
 - I won't be able to cover everything as I only have about an hour and a half.

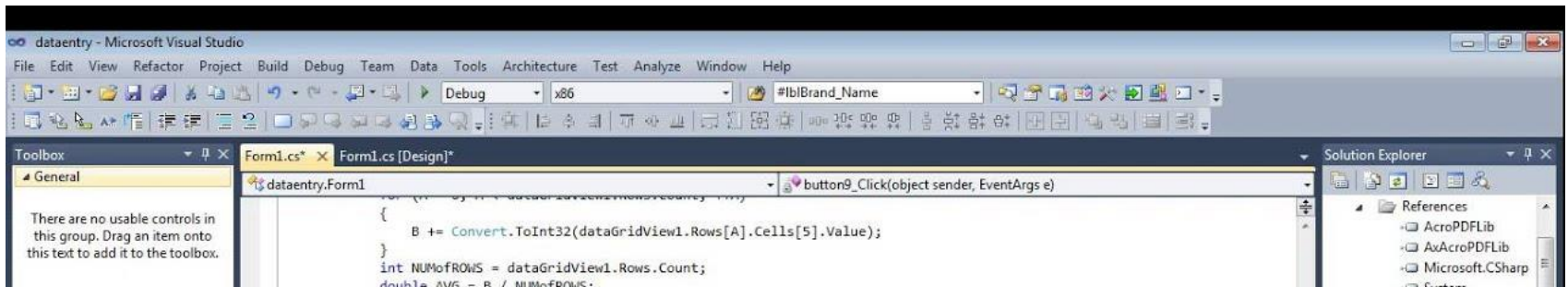
C#8

- Nullable Reference Types
- New Array index and range operators
- IN/OUT/REF new parameter features
- Using Async with a Main Method
- Default Interface Implementations
- Pattern Matching
- Switch Expressions
- Inferred Tuples
- Async Streams

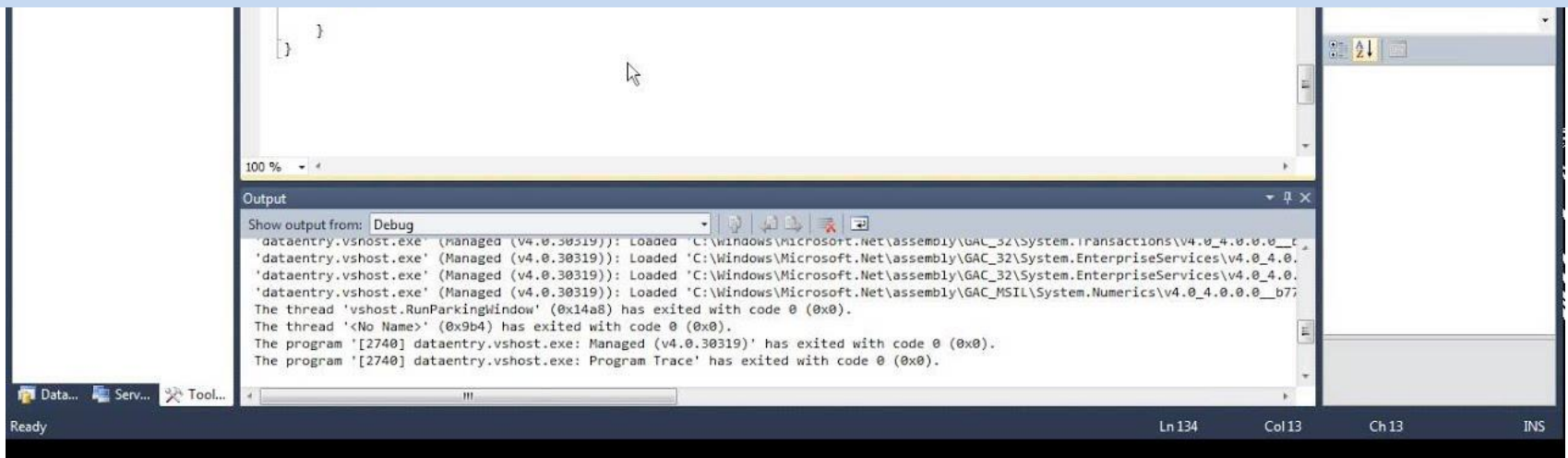
C#8

- Nullable Reference Types
 - Previous versions of C# only allowed value types such as int/bool/float to be nullable.
 - Strings can be set to “null content” but that’s as far as things went.
 - With nullable reference types, strings, structs and many other things can now be set to null.
 - As a result of this new feature however, the ability to cause null reference exceptions grows, so the compiler now has new tools to help you write better code.

Demo Time...



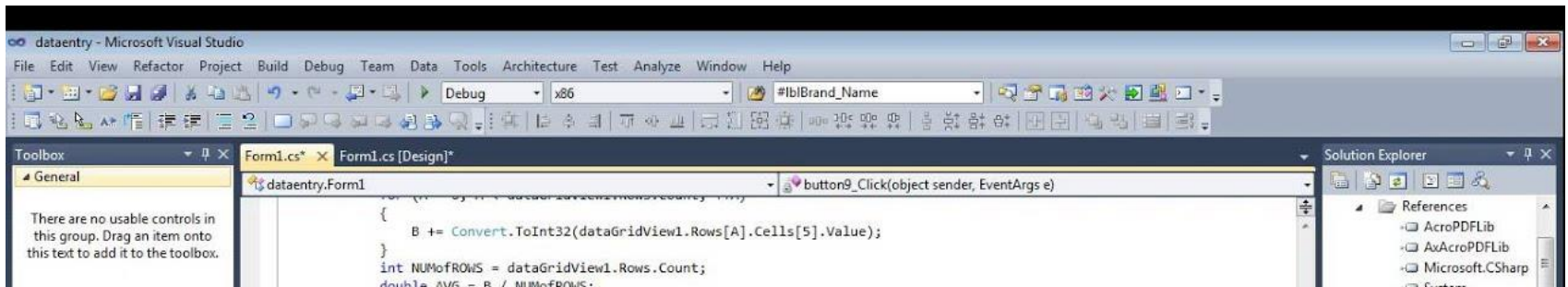
Nullable Reference Types



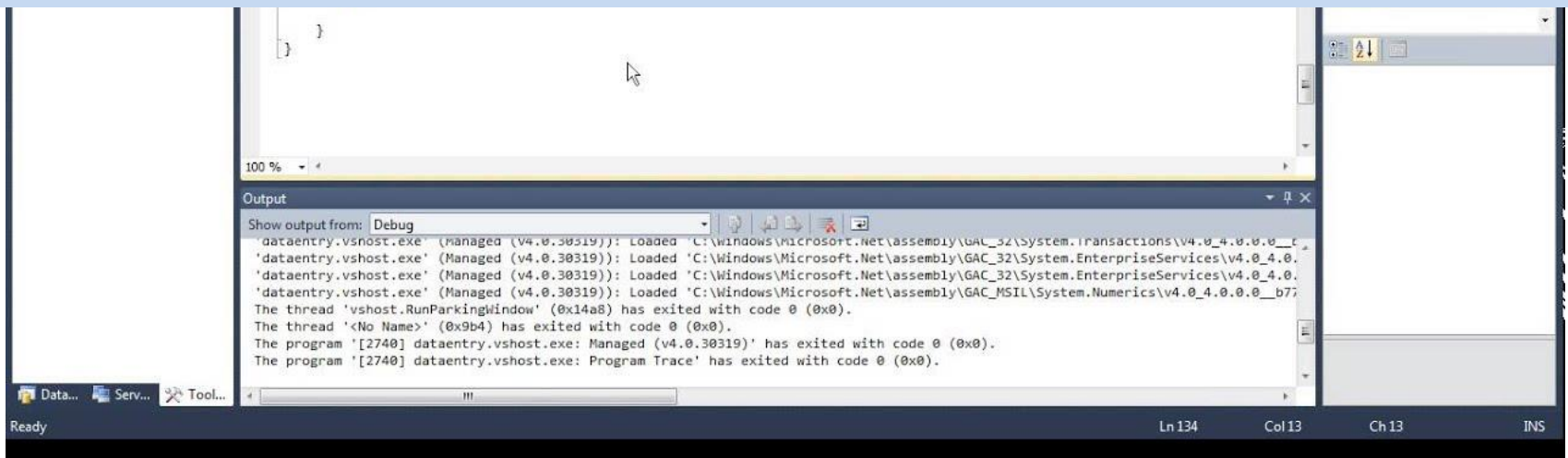
C#8

- New Array index and range operators
 - In previous versions of C# and array was an array and that was that, if you wanted to work with just a small subset of it you had to make a copy of just that part of the array.
 - From C#8 onwards there are 2 new array operators, the array index operator and the array range operator.
 - Range as the name suggests allows you to work with just a small range, index is like the original index in it's operation, but has a small new addition.

Demo Time...



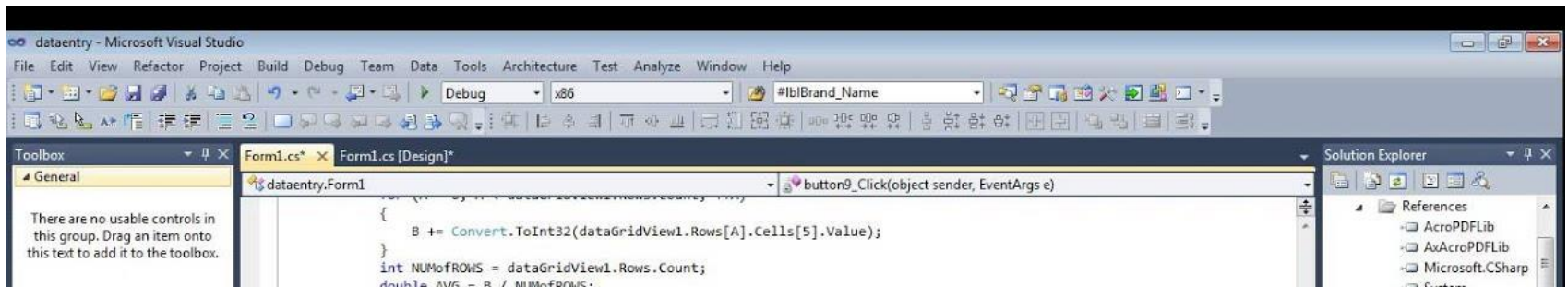
Array Index and Range operators



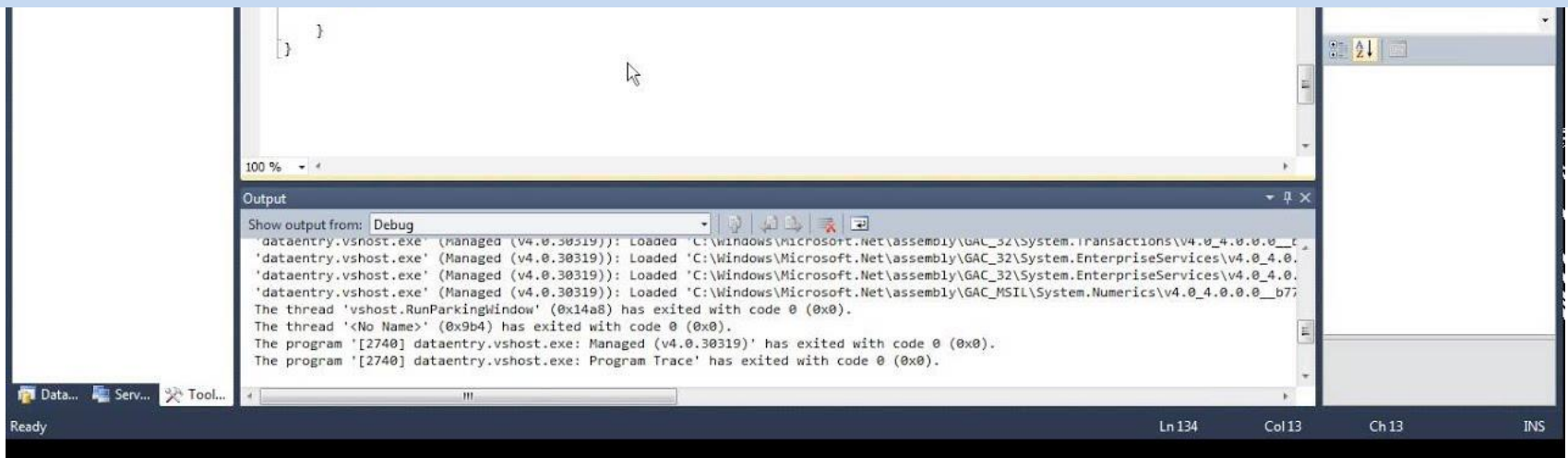
C#8

- Parameter reference operators
 - Passing variables into and out of methods in previous versions of the compiler, generally only existed in 2 forms. “OUT” and “REF”.
 - Out parameters where filled in by the method and passed back, setting the contents of the source.
 - Ref parameters could have a value on input and could be changed on return.
 - C#8 introduces a new operator called “IN”

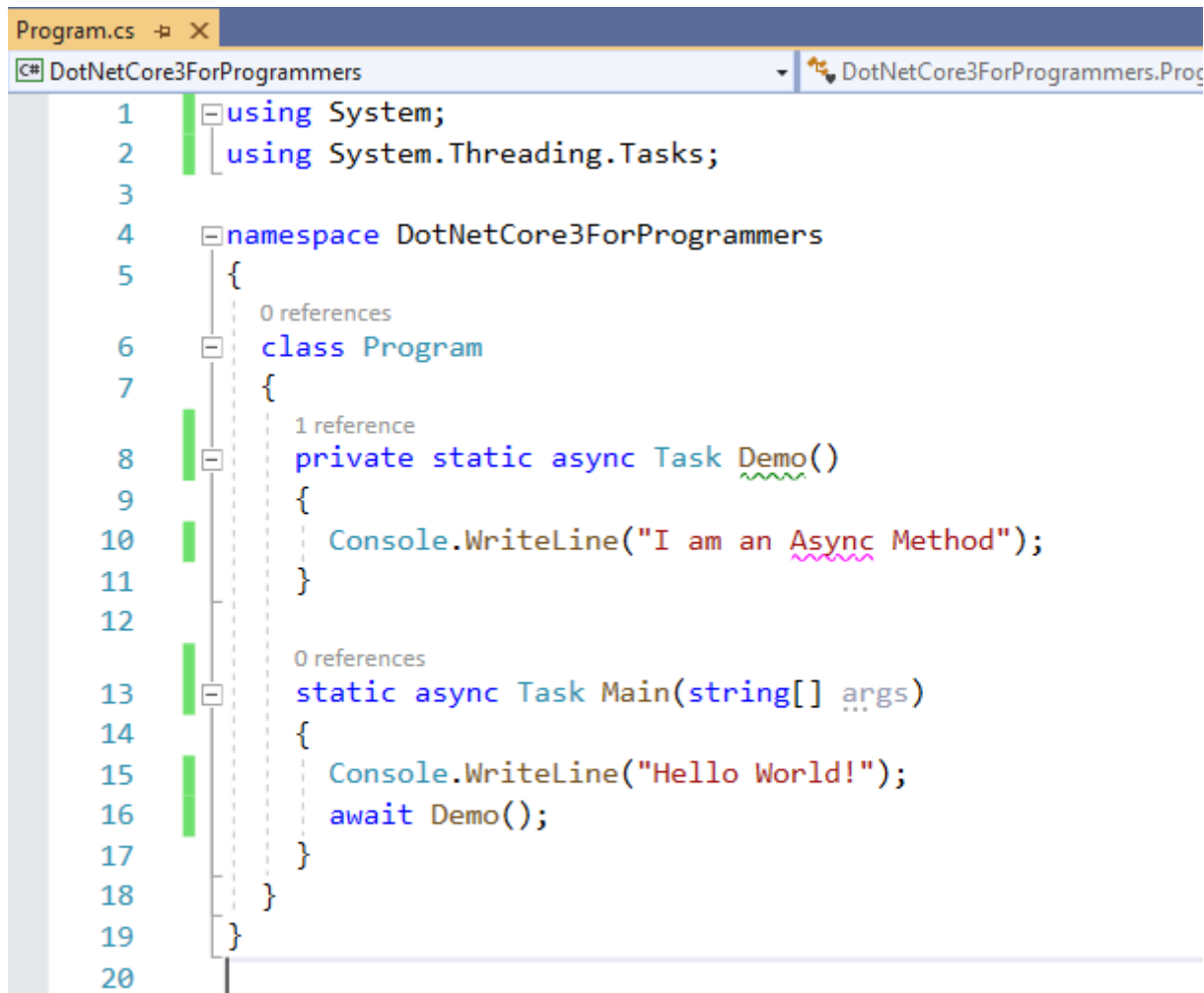
Demo Time...



Parameter Reference Operators



C#8 - Async Main



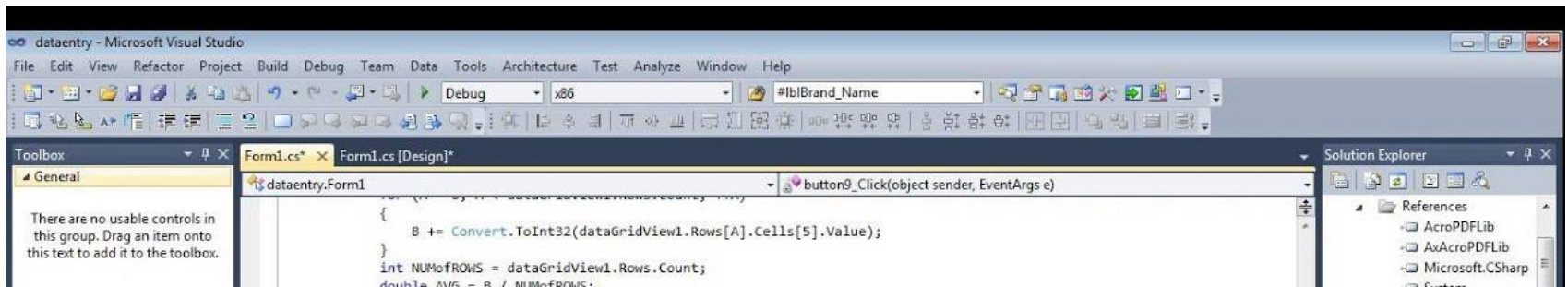
The screenshot shows a code editor window with a file named 'Program.cs'. The code is written in C# and defines a namespace 'DotNetCore3ForProgrammers' containing a class 'Program'. Inside the 'Program' class, there are two static methods: 'Demo()' and 'Main()'. The 'Demo()' method is a private static async Task that writes 'I am an Async Method' to the console. The 'Main()' method is a static async Task that writes 'Hello World!' to the console and then awaits the 'Demo()' method. The code is numbered from 1 to 20 on the left side of the editor.

```
1  using System;
2  using System.Threading.Tasks;
3
4  namespace DotNetCore3ForProgrammers
5  {
6      class Program
7      {
8          private static async Task Demo()
9          {
10             Console.WriteLine("I am an Async Method");
11          }
12
13         static async Task Main(string[] args)
14         {
15             Console.WriteLine("Hello World!");
16             await Demo();
17         }
18     }
19 }
20
```

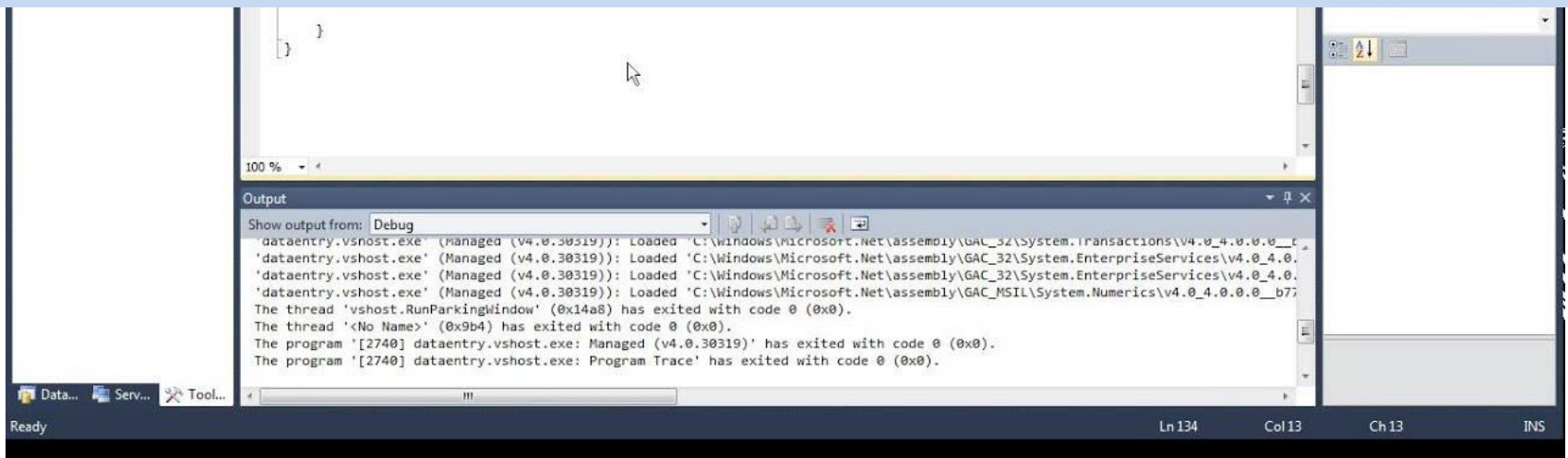
C#8

- Default Interface Implementations
 - Previously you couldn't change an interface contract without forcing everyone else to implement the changes.
 - Adding new functionality to an interface without causing massive breaking changes was very difficult.
 - C#8 allows interfaces to be changed and not force those changes on older code.

Demo Time...



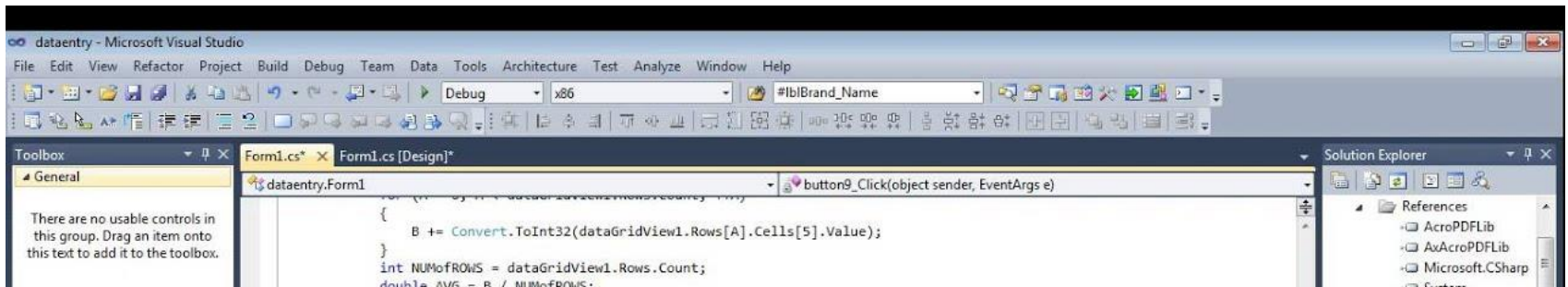
Default Interface Implementations



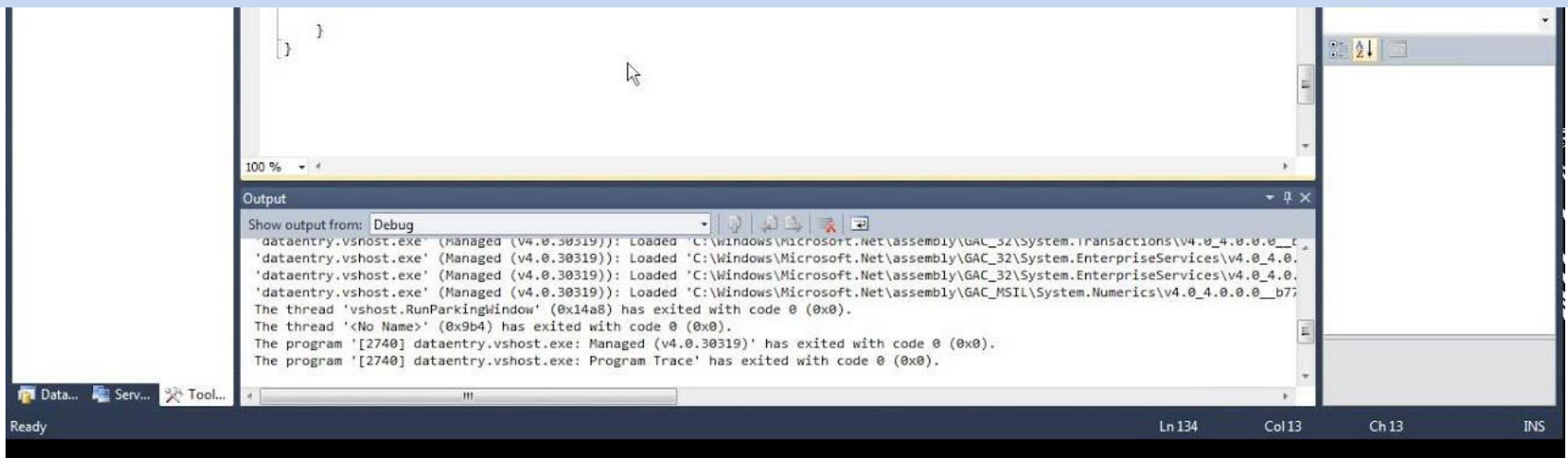
C#8

- Pattern Matching
 - Just not enough time to fully do it justice. Allows a staggering amount of filtering and type matching based on properties, order of properties, anonymous objects and much, much, more.
 - Solves a lot of problems and reduces lots of if/then checks to decide what types of objects and/or structures you're dealing with.
 - Makes Linq and object manipulation 100% better than it already was.

Demo Time...



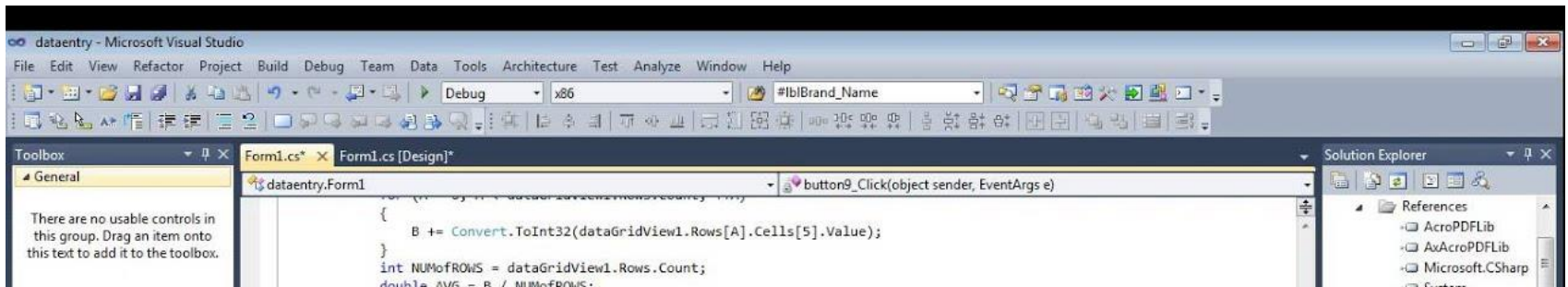
Pattern Matching



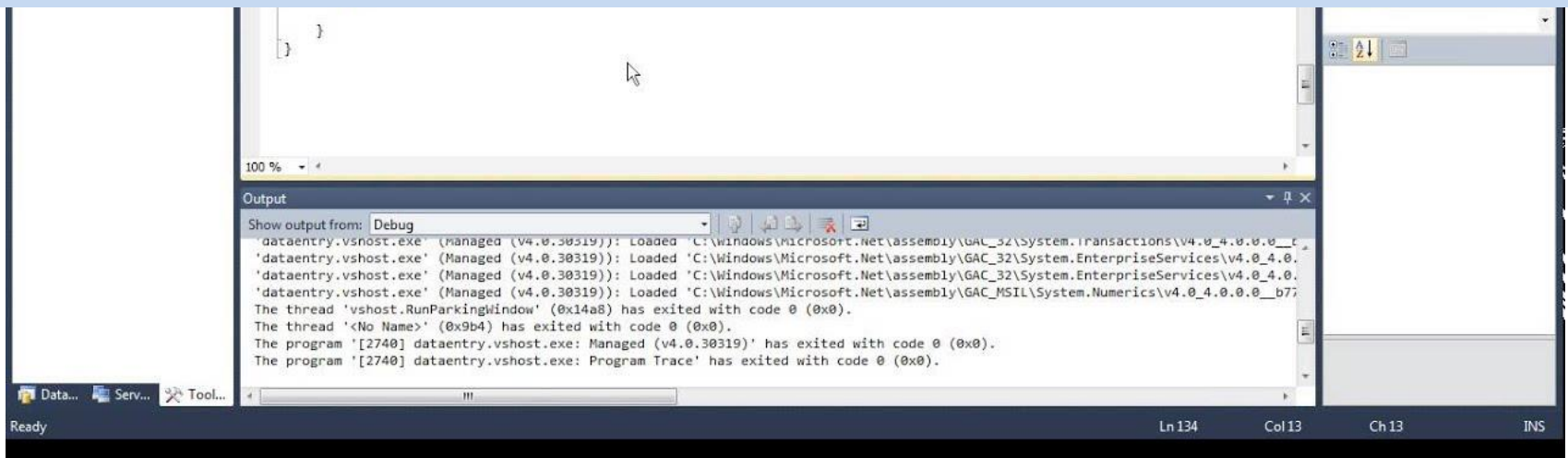
C#8

- Switch Statements
 - Switch like if/then is a staple construct in ALL modern high level languages.
 - C#8 turns the switch statement into an inline result orientated operator.
 - Would NOT be possible without the new pattern matching changes.
 - Makes working with adapter based architecture patterns a breeze.

Demo Time...



Switch Statements



C#8

- Inferred Tuples

- Tuples have been with .NET since at least C#6, I blogged about them a few years ago :
<https://www.codeguru.com/columns/dotnet/a-tipple-with-a-tuple.html>
- In C#8 Tuples have been greatly expanded and now almost have super powers in terms of what they can do.
- Inferred tuples no longer are constrained to the 7 basic “Itemx” properties they used to be.

C#8 – Inferred Tuples

Program.cs* [Icon] [X]

[C#] DotNetCore3ForProgrammers [DotNetCore3ForProgrammers.Program] [Main(string[] args)]

```
1 using System;
2 using System.Threading.Tasks;
3
4 namespace DotNetCore3ForProgrammers
5 {
6     0 references
7     class Program
8     {
9         0 references
10        static void Main(string[] args)
11        {
12            string Name = "Peter";
13            int Age = 21;
14
15            var inferredTuple = (Name, Age);
16
17            var theName = inferredTuple.
18
19        }
20    }
```

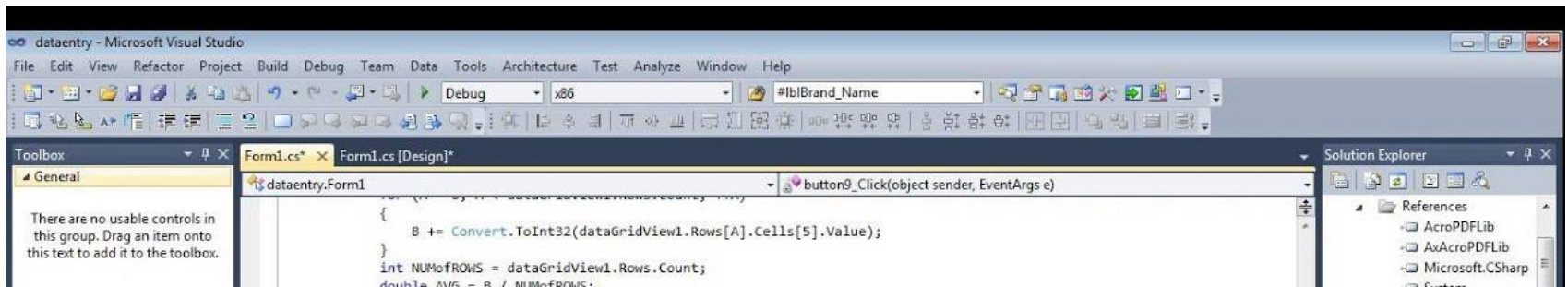
Age
CompareTo
Equals
GetHashCode
GetType
Name
ToString
ToTuple<>

(field) string (string Name, int Age).Name
Gets the value of the current ValueTuple<T1, T2> instance's first element.

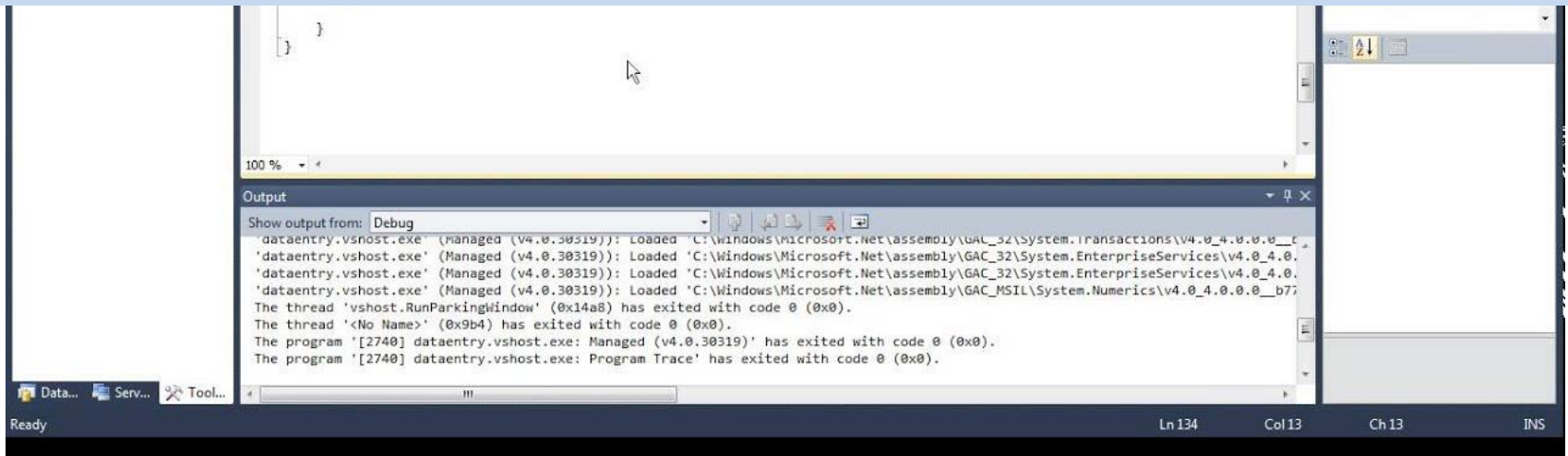
C#8

- Async Streams
 - If you've done ANY kind of reactive programming then you'll be right at home with async streams.
 - Those of you who are used to “Generators” in JavaScript will understand the concept straight away.
 - C# async prior to C#8 could not await on common constructs such as for each loops, so waiting on things like message queues was a bit of a hassle
 - Async streams solve this problem by enabling async on many of these loop operations, and providing new async based enumerable types to work with.

Demo Time...



Async Streams



Dot Net Core

- Dependency Injector
- SPAN<T> and MEMORY<T>
- JSON Parsing
- Crypto Functionality
- Brotli Compression
- Desktop Apps
- Custom Code Injection

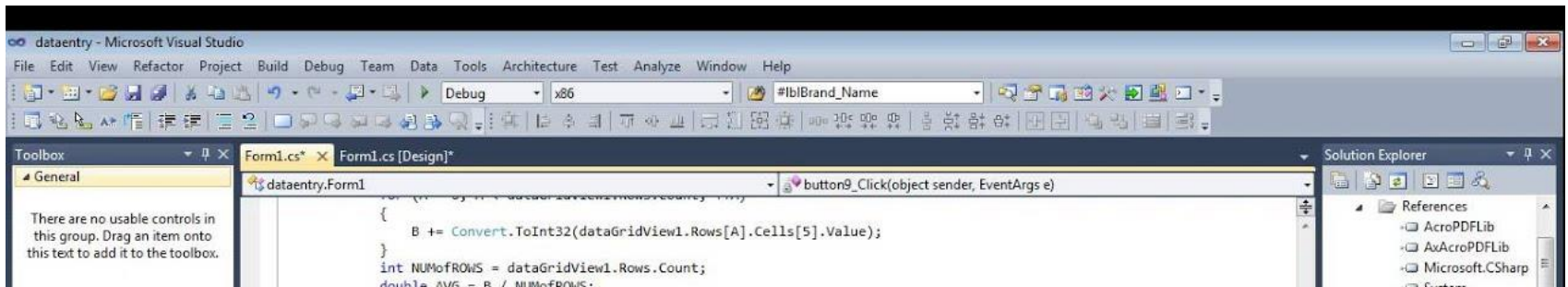
Dotnet Core

- Dependency Injector
 - Not really new in 3/3.1 was introduced in 2.0 but improved a lot in more recent versions of the framework.
 - In 3/3.1 now has hooks to hook your own DI framework into it, but still continue to use the same API.
 - Will now inject dependency chains all the way down into related class modules.

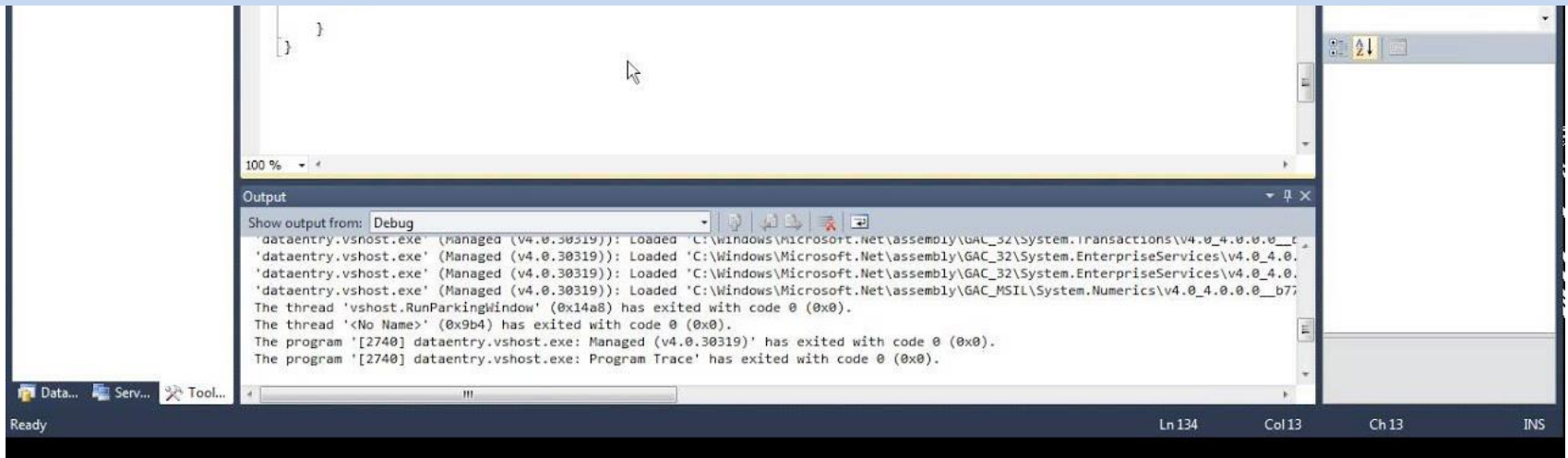
Dotnet Core

- `Span<T>` and `Memory<T>`
 - New managed types for high speed memory access without using unsafe code.
 - Biggest Advantage to normal LOB & App developers is speed of access in string manipulation and parsing.
 - Reduces the amount of string allocations needed on the heap when pulling strings apart into smaller strings.
 - Allows direct memory access to the underlying contiguous memory representation of the associated variable without blowing things up.

Demo Time...



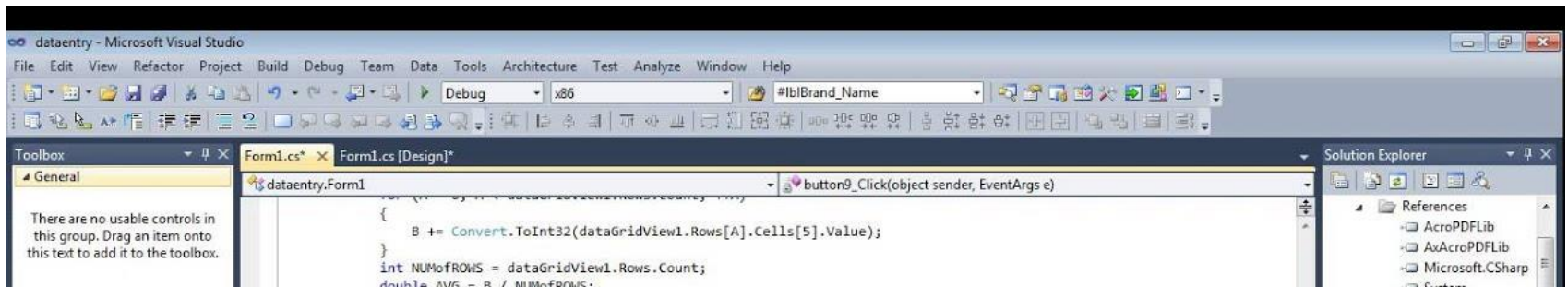
Span<T> and Memory<T>



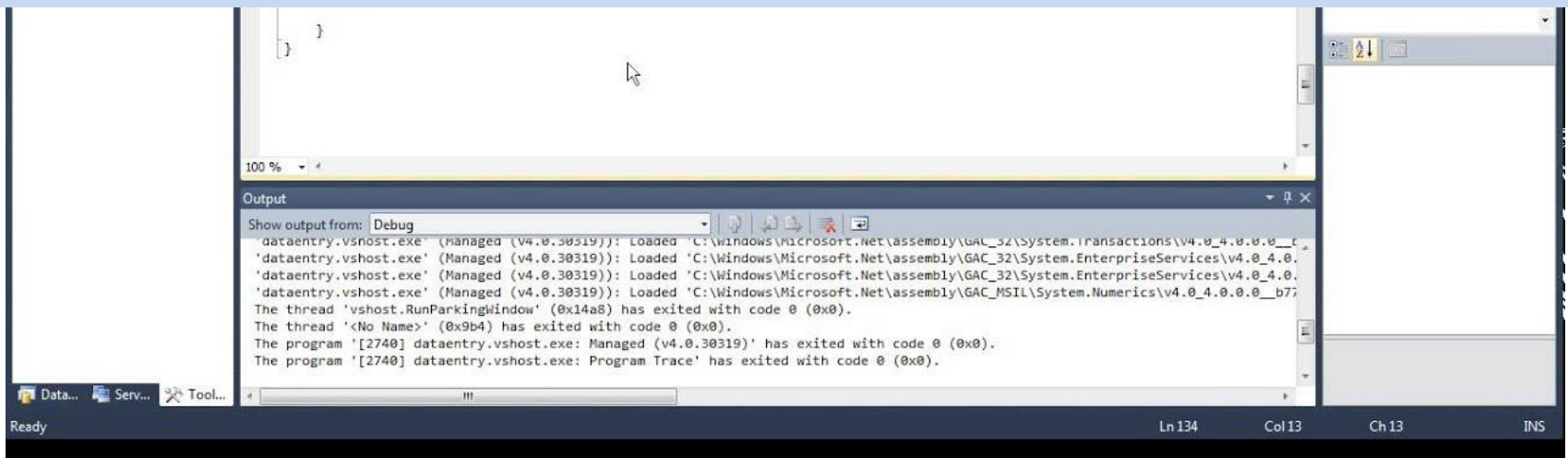
Dotnet Core

- JSON Parsing
 - What was wrong with Newtonsoft JSON.NET?
 - Why build a new parser?
 - Very, Very vastly improved, and extremely fast thanks to the new span and memory types.
 - Has a model not too dissimilar to the XmlDocument and XElement classes so for those constructing JSON at a low level you have an unprecedented amount of control.
 - API is very, very similar (Identical in some places) to newtonsoft's

Demo Time...



JSON Parser



Dotnet Core

- Crypto improvements
 - More algorithms than you can shake your fist at some common, some not so common.
 - Development and usage model, as well as API greatly simplified from previous versions.
 - Public/Private key encryption, and key generation can now be performed in about 10 lines of code!
 - Keys can be saved and loaded easily using standard file operations.

Dotnet Core - Crypto

Version

.NET Core 3.1

Search

- > Rijndael
- > RijndaelManaged
- > RNGCryptoServiceProvider
- > RSA
- > RSACng
- > RSACryptoServiceProvider
- > RSAEncryptionPadding
- > RSAEncryptionPaddingMode
- > RSAOAEPKeyExchangeDeformatter
- > RSAOAEPKeyExchangeFormatter
- > RSAPParameters
- > RSAPKCS1KeyExchangeDeformatter
- > RSAPKCS1KeyExchangeFormatter
- > RSAPKCS1SignatureDeformatter
- > RSAPKCS1SignatureFormatter
- > RSASignaturePadding
- > RSASignaturePaddingMode
- > SHA1
- > SHA1CryptoServiceProvider
- > SHA1Managed
- > SHA256
- > SHA256CryptoServiceProvider
- > SHA256Managed
- > SHA384
- > SHA384CryptoServiceProvider
- > SHA384Managed
- > SHA512
- > SHA512CryptoServiceProvider
- > SHA512Managed
- > SignatureDescription

Download PDF

System.Security.Cryptography Namespace

The [System.Security.Cryptography](#) namespace provides cryptographic services, including secure encoding and decoding of data, as well as many other operations, such as hashing, random number generation, and message authentication. For more information, see [Cryptographic Services](#).

Classes

Aes	Represents the abstract base class from which all implementations of the Advanced Encryption Standard (AES) must inherit.
AesCcm	Represents an Advanced Encryption Standard (AES) key to be used with the Counter with CBC-MAC (CCM) mode of operation.
AesCng	Provides a Cryptography Next Generation (CNG) implementation of the Advanced Encryption Standard (AES) algorithm.
AesCryptoServiceProvider	Performs symmetric encryption and decryption using the Cryptographic Application Programming Interfaces (CAPI) implementation of the Advanced Encryption Standard (AES) algorithm.
AesGcm	Represents an Advanced Encryption Standard (AES) key to be used with the Galois/Counter Mode (GCM) mode of operation.
AesManaged	Provides a managed implementation of the Advanced Encryption Standard (AES) symmetric algorithm.
AsnEncodedData	Represents Abstract Syntax Notation One (ASN.1)-encoded data.
AsnEncodedDataCollection	Represents a collection of AsnEncodedData objects. This class cannot be inherited.
AsnEncodedDataEnumerator	Provides the ability to navigate through an AsnEncodedDataCollection object. This class cannot be inherited.
AsymmetricAlgorithm	Represents the abstract base class from which all implementations of asymmetric algorithms must inherit.
AsymmetricKeyExchangeDeformatter	Represents the base class from which all asymmetric key exchange deformatters derive.
AsymmetricKeyExchangeFormatter	Represents the base class from which all asymmetric key exchange formatters derive.
AsymmetricSignatureDeformatter	Represents the abstract base class from which all implementations of asymmetric signature deformatters derive.
AsymmetricSignatureFormatter	Represents the base class from which all implementations of asymmetric signature formatters derive.

Is this page helpful?

Yes No

In this article

[Classes](#)

[Structs](#)

[Interfaces](#)

[Enums](#)

Dotnet Core - Crypto

```
Program.cs*  -  X
DotNetCore3ForProgrammers  -  DotNetCore3ForProgrammers.Program  -  EncryptAndEncodeB64(string plainText, byte[] Key, byte[] IV)

1  using System;
2  using System.IO;
3  using System.Security.Cryptography;
4  using System.Threading.Tasks;
5
6  namespace DotNetCore3ForProgrammers
7  {
8      0 references
9      class Program
10     {
11         0 references
12         static void Main(string[] args) ...
13
14         0 references
15         static string EncryptAndEncodeB64(string plainText, byte[] Key, byte[] IV)
16         {
17             byte[] encrypted;
18
19             using (RijndaelManaged encryptionAlgo = new RijndaelManaged())
20             {
21                 encryptionAlgo.Key = Key;
22                 encryptionAlgo.IV = IV;
23
24                 encryptionAlgo.BlockSize = 128;
25                 encryptionAlgo.Mode = CipherMode.CBC;
26                 encryptionAlgo.Padding = PaddingMode.PKCS7;
27
28                 ICryptoTransform encryptor = encryptionAlgo.CreateEncryptor(encryptionAlgo.Key, encryptionAlgo.IV);
29
30                 using MemoryStream msEncrypt = new MemoryStream();
31                 using CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write);
32                 using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
33                 {
34                     swEncrypt.Write(plainText);
35                 }
36                 encrypted = msEncrypt.ToArray();
37             }
38             return Convert.ToBase64String(encrypted);
39         }
40     }
41 }
42
43
44
45
46
47
```

Dotnet Core - Crypto

```
Program.cs*  ▢  X
DotNetCore3ForProgrammers  ▾  DotNetCore3ForProgrammers.Program  ▾  EncryptAndEncodeB64(s
1  using System;
2  using System.IO;
3  using System.Security.Cryptography;
4  using System.Threading.Tasks;
5
6  namespace DotNetCore3ForProgrammers
7  {
8      0 references
9      class Program
10     {
11         0 references
12         static void Main()
13         {
14             string notEncryptedText = "I am some test text to encrypt";
15             string encryptedText = "";
16
17             byte[] encryptionKey; // Our actual encryption Decryption key
18             byte[] initializationVector; // The salt value/IV
19
20             using (RijndaelManaged initEncryptor = new RijndaelManaged())
21             {
22                 initEncryptor.BlockSize = 128;
23                 initEncryptor.Mode = CipherMode.CBC;
24                 initEncryptor.Padding = PaddingMode.PKCS7;
25
26                 initEncryptor.GenerateKey();
27                 initEncryptor.GenerateIV();
28
29                 encryptionKey = initEncryptor.Key;
30                 initializationVector = initEncryptor.IV;
31             }
32
33             Console.WriteLine("Encryption test");
34             Console.WriteLine($"Original Plaintext : {notEncryptedText}");
35
36             encryptedText = EncryptAndEncodeB64(notEncryptedText, encryptionKey, initializationVector);
37
38             Console.WriteLine($"Encrypted and Encoded to B64 : {encryptedText}");
39         }
40     }
```


Dotnet Core - Crypto

```
Program.cs*  X
DotNetCore3ForProgrammers  keyencapp.Program  Main(str

1  using System;
2  using System.Security.Cryptography;
3  using System.Text;
4
5  namespace keyencapp
6  {
7      0 references
8      class Program
9      {
10         0 references
11         static void Main(string[] args)
12         {
13             var rsa = new RSACryptoServiceProvider();
14
15             byte[] privateKey = rsa.ExportRSAPrivateKey();
16             byte[] publicKey = rsa.ExportRSAPublicKey();
17
18             string dataToEncrypt = "This is some simple data to encrypt";
19             byte[] dataToEncryptBytes = Encoding.ASCII.GetBytes(dataToEncrypt);
20
21             byte[] encryptedDataBytes = rsa.Encrypt(dataToEncryptBytes, false);
22
23             string hexString = BitConverter.ToString(encryptedDataBytes);
24
25             Console.WriteLine($"ORIGINAL DATA                : {dataToEncrypt}");
26             Console.WriteLine($"ENCRYPTED DATA (displayed as hex string): {hexString}");
27         }
28     }
29 }
30
31
```

Dotnet Core

- Brotli Compression
 - A new form of compression, aimed primarily at web server software.
 - Designed to be efficient at streaming compression esp in HTTP2, but equally good at compressing static resources too.
 - Better compression ratios than the current GZip algorithm used by most servers, used since the 1970's.
 - Supported in ALL current browsers.
 - If your interested in knowing more about 'Brotli' the Wikipedia article is very informative :
<https://en.wikipedia.org/wiki/Brotli>

Dotnet Core – Brotli Compression

Program.cs

DotNetCore3ForProgrammers

brotlicompression.Program

Main()

```
1 using System;
2 using System.IO;
3 using System.IO.Compression;
4
5 namespace brotlicompression
6 {
7     0 references
8     class Program
9     {
10         0 references
11         static void Main()
12         {
13             string inputFileName = @"P:\brotli\tiftest.tif";
14             string outputFileName = inputFileName + ".br";
15
16             using (FileStream input = File.OpenRead(inputFileName))
17             {
18                 Console.WriteLine("Opening Input File");
19
20                 using (FileStream output = File.Create(outputFileName))
21                 {
22                     Console.WriteLine("Opening Output File");
23
24                     using (BrotliStream compressor = new BrotliStream(output, CompressionMode.Compress))
25                     {
26                         Console.WriteLine("Copy input to output.");
27                         input.CopyTo(compressor);
28                     }
29                 }
30             }
31             Console.WriteLine("Complete");
32         }
33     }
34 }
```

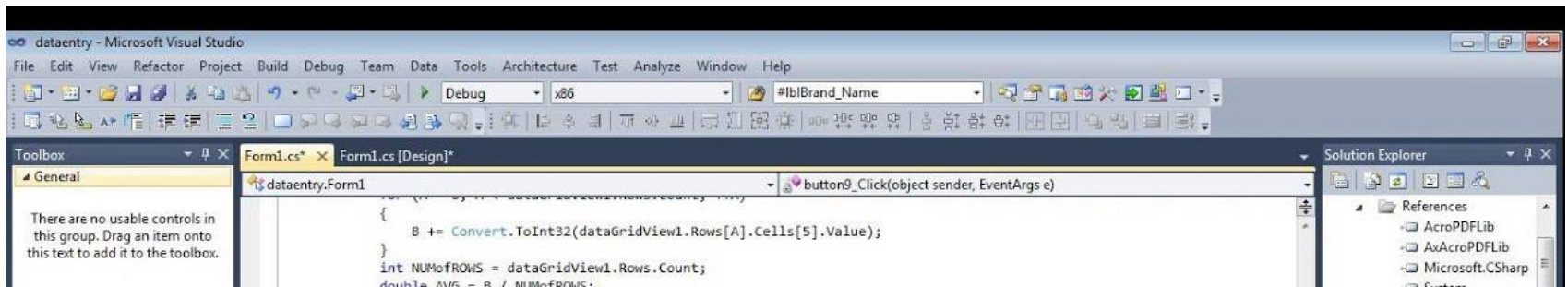
Dotnet Core

- Brotli Compression
 - Using it is incredibly simple.
 - Use it to compress individual files as the code shows.
 - Use it to encrypt and stream on the fly direct from an ASP.NET MVC controller.
 - To use it all you need is a “System.IO.Stream” to write your data into.

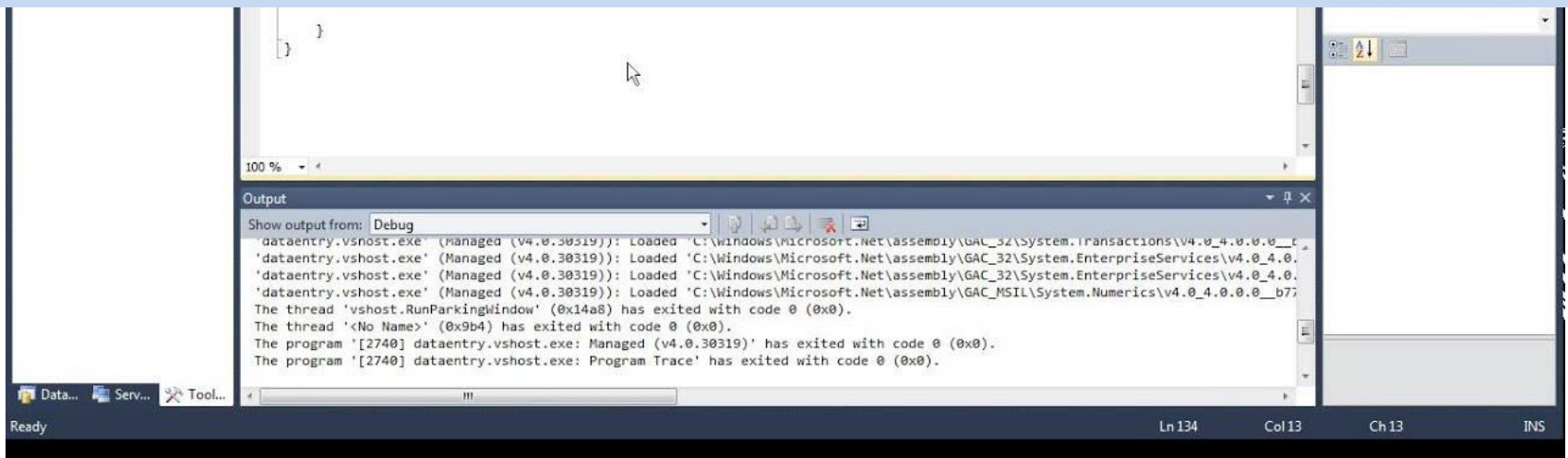
Dotnet Core

- Desktop Applications
 - Windows Forms applications can now be ported to and created on Dotnet core.
 - WPF applications can also be handled.
 - There is still no graphical designer by default, the first time you open a windows forms class you will be asked if you want to use the preview of it.
 - Desktop apps work under Windows ONLY, are not going to be supported beyond core 3.1

Demo Time...



Desktop Apps



Dotnet Core

- Custom Code Injection
 - Runs before the “Main” method in which ever application it’s linked to.
 - Has FULL access to every variable, object and class in the application being intercepted.
 - Is controlled by a simple environment variable.
 - Use with extreme caution and audit any running application environments regularly to ensure it’s not being abused.

Dotnet Core

- Custom Code Injection

```
{  
  "profiles": {  
    "StartHookTest": {  
      "commandName": "Project",  
      "environmentVariables": {  
        "DOTNET_STARTUP_HOOKS": ""  
      }  
    }  
  }  
}
```

This example shows the use of “LaunchSettings.json” to set the variable, but you can set anyway your target OS allows you too.

Dotnet Core – Code Injection

```
using System;

public class StartupHook
{
    public static void Initialize()
    {
        Console.WriteLine("This line of text came from the 'Startup hook' attached to this application.");
    }
}
```

The class to implement your hook is very simple, but there are some rules that MUST be Followed.

- 1) The ACTUAL Class name must be “StartupHook” with a capital “S” and “H”
- 2) The class MUST include one public method called “Initialize”
- 3) Your class MUST NOT have a namespace of any kind.

Other classes in other namespaces can be used and instantiated as normal, but the hook itself Must look like the code above and follow these rules.

Dotnet Core

- Custom Code Injection

```
{  
  "profiles": {  
    "StartHookTest": {  
      "commandName": "Project",  
      "environmentVariables": {  
        "DOTNET_STARTUP_HOOKS":  
          "C:\\Users\\shawty\\source\\repos\\StartHookApp\\TheStartupHook\\bin\\Debug\\netcoreapp3.0\\TheStartupHook.dll"  
      }  
    }  
  }  
}
```

Once you set the name of the class in the variable, you **MUST** use the full absolute path name, even if the DLL is in the same folder as the application being intercepted.

Summary

- Dotnet Core 3/3.1 and C#8 have a huge amount of new API's and features, far to many to list them all here.
- More will be added in future versions, from now on DotNet will be an evolving platform.
- MS Are listening to the community on this one, and new features are very much a case of things developers have been asking for.

Summary

- Please make sure you thank Hainton.NET for making these sessions available to you by spreading the word, software development is about sharing knowledge and experiences.
- Contacting me is easy:
 - Twitter @shawty_ds
 - Email shawty.d.ds@gmail.com
 - Linked-in The LiDNUG user group (lidnug.org)