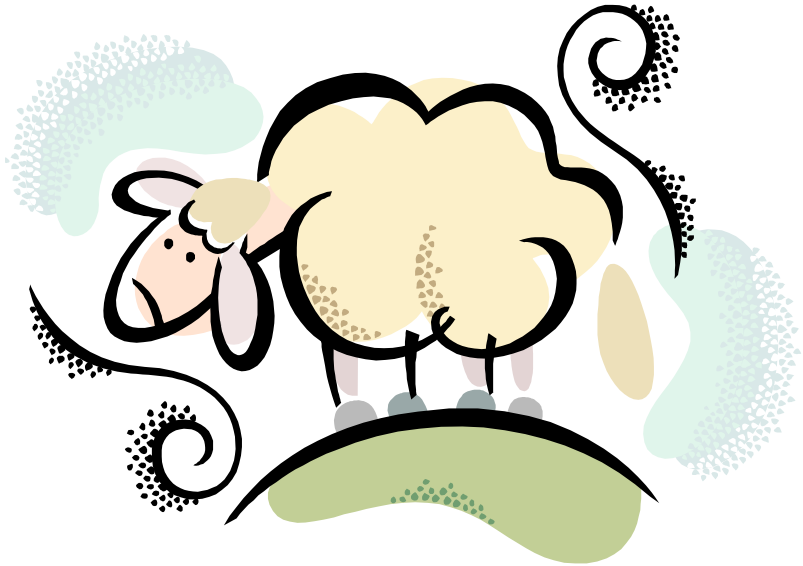


C# Programming IV-1:

System Namespace

Copyright ©2011 sheepsqueezers.com

Legal Stuff

This work may be reproduced and redistributed, in whole or in part, without alteration and without prior written permission, provided all copies contain the following statement:

```
Copyright ©2011 sheepsqueezers.com.  This
work is reproduced and distributed with the
permission of the copyright holder.
```

This presentation as well as other presentations and documents found on the sheepsqueezers.com website may contain quoted material from outside sources such as books, articles and websites. It is our intention to diligently reference all outside sources. Occasionally, though, a reference may be missed. No copyright infringement whatsoever is intended, and all outside source materials are copyright of their respective author(s).

.NET Lecture Series

	C# Programming I: Concepts of OOP	C# Programming II: Beginning C#	C# Programming III: Advanced C#	
C# Programming IV-1: System Namespace	C# Programming IV-2: System.Collections Namespace	C# Programming IV-3: System.Collections. Generic Namespace	C# Programming IV-4A: System.Data Namespace	C# Programming IV-4B: System.Data.Odbc Namespace
C# Programming IV-4C: System.Data.OleDb Namespace	C# Programming IV-4D: Oracle.DataAccess.Client Namespace	C# Programming IV-4E: System.Data.SqlClient Namespace	C# Programming IV-4F: System.Data.SqlTypes Namespace	C# Programming IV-5: System.Drawing/(2D) Namespace
C# Programming IV-6: System.IO Namespace	C# Programming IV-7: System.Numerics	C# Programming IV-8: System.Text and System.Text. RegularExpressions Namespaces	C# Programming V: Introduction to LINQ	

C# Self-Inflicted Project #1 — Address Cleaning

C# Self-Inflicted Project #2 — Large Intersection Problem

Copyright ©2011 sheepsqueezers.com

Charting Our Course

- The `System` Namespace
- What Next?

sheepsqueezers.com

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

The `System` Namespace is defined by Microsoft as follows:

The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

This one brief sentence just doesn't do the `System` namespace justification! In this presentation, we will explore many of the classes of the `System` namespace in detail. Note that the System namespace is contained in `mscorlib.dll`.

If you go to Microsoft's website and view the `System` namespace, you will see that they have broken up this namespace into classes, structures, interfaces, delegates and enumerations.

In the first section of this presentation, we will explore the classes in the `System` namespace. On Microsoft's website, all of the classes are in alphabetical order including any classes derived from the `Exception`, `Attribute` or `EventArgs` classes. In my opinion, this is unfortunate because it hides the real manly-man classes amongst the exceptions and attributes. So, below you will see that I have broken up the classes into three sections: *Classes Derived from the Exception Class*, *Classes Derived from the Attribute Class*, *Classes Derived from the EventArgs Class*, and *Classes Not Derived from the Exception, Attribute and EventArgs Classes*.

Note: Some authors indicate a class using the dot notation such as `System.Math` to indicate the `Math` class (tee-hee!) in the `System` namespace. I prefer to indicate this as *Math class in the System namespace,* but I'm sure this is because I'm so new to the .NET Framework and C#. (I may change my opinion later on once I know what I'm doing…if ever…)

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Derived from the `Exception` Class

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Exception` Class

We talked about how to handle exceptions in the C# Programming II presentation, so we won't repeat it here. Below are a list of exceptions available in the `System` namespace. Other namespaces will have their own exceptions and With the exception (tee-hee!) of the `Exception` class, all of the classes listed below extend the `Exception` class. You can create your own exception as well just by subclassing the `Exception` class.

- **Exception - Represents errors that occur during application execution.**
- AccessViolationException - The exception that is thrown when there is an attempt to read or write protected memory.
- AggregateException - Represents one or more errors that occur during application execution.
- AppDomainUnloadedException - The exception that is thrown when an attempt is made to access an unloaded application domain.
- ApplicationException - The exception that is thrown when a non-fatal application error occurs.
- ArgumentException - The exception that is thrown when one of the arguments provided to a method is not valid.
- ArgumentNullException - The exception that is thrown when a null reference (Nothing in Visual Basic) is passed to a method that does not accept it as a valid argument.
- ArgumentOutOfRangeException - The exception that is thrown when the value of an argument is outside the allowable range of values as defined by the invoked method.
- ArithmeticException - The exception that is thrown for errors in an arithmetic, casting, or conversion operation.
- ArrayTypeMismatchException - The exception that is thrown when an attempt is made to store an element of the wrong type within an array.
- BadImageFormatException - The exception that is thrown when the file image of a dynamic link library (DLL) or an executable program is invalid.
- CannotUnloadAppDomainException - The exception that is thrown when an attempt to unload an application domain fails.
- ContextMarshalException - The exception that is thrown when an attempt to marshal an object across a context boundary fails.
- DataMisalignedException - The exception that is thrown when a unit of data is read from or written to an address that is not a multiple of the data size. This class cannot be inherited.
- DivideByZeroException - The exception that is thrown when there is an attempt to divide an integral or decimal value by zero.
- DllNotFoundException - The exception that is thrown when a DLL specified in a DLL import cannot be found.
- DuplicateWaitObjectException - The exception that is thrown when an object appears more than once in an array of synchronization objects.
- EntryPointNotFoundException - The exception that is thrown when an attempt to load a class fails due to the absence of an entry method.

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Exception` Class

- ExecutionEngineException - Obsolete. The exception that is thrown when there is an internal error in the execution engine of the common language runtime. This class cannot be inherited.
- FieldAccessException - The exception that is thrown when there is an invalid attempt to access a private or protected field inside a class.
- FormatException - The exception that is thrown when the format of an argument does not meet the parameter specifications of the invoked method.
- IndexOutOfRangeException - The exception that is thrown when an attempt is made to access an element of an array with an index that is outside the bounds of the array. This class cannot be inherited.
- InsufficientExecutionStackException - The exception that is thrown when there is insufficient execution stack available to allow most methods to execute.
- InsufficientMemoryException - The exception that is thrown when a check for sufficient available memory fails. This class cannot be inherited.
- InvalidCastException - The exception that is thrown for invalid casting or explicit conversion.
- InvalidOperationException - The exception that is thrown when a method call is invalid for the object's current state.
- InvalidProgramException - The exception that is thrown when a program contains invalid Microsoft intermediate language (MSIL) or metadata. Generally this indicates a bug in the compiler that generated the program.
- InvalidTimeZoneException - The exception that is thrown when time zone information is invalid.
- MemberAccessException - The exception that is thrown when an attempt to access a class member fails.
- MethodAccessException - The exception that is thrown when there is an invalid attempt to access a method, such as accessing a private method from partially trusted code.
- MissingFieldException - The exception that is thrown when there is an attempt to dynamically access a field that does not exist.
- MissingMemberException - The exception that is thrown when there is an attempt to dynamically access a class member that does not exist.
- MissingMethodException - The exception that is thrown when there is an attempt to dynamically access a method that does not exist.
- NotFiniteNumberException - The exception that is thrown when a floating-point value is positive infinity, negative infinity, or Not-a-Number (NaN).
- NotImplementedException - The exception that is thrown when a requested method or operation is not implemented.
- NotSupportedException - The exception that is thrown when an invoked method is not supported, or when there is an attempt to read, seek, or write to a stream that does not support the invoked functionality.
- NullReferenceException - The exception that is thrown when there is an attempt to dereference a null object reference.
- ObjectDisposedException - The exception that is thrown when an operation is performed on a disposed object.
- MulticastNotSupportedException - The exception that is thrown when there is an attempt to combine two delegates based on the Delegate type instead of the MulticastDelegate type. This class cannot be inherited.

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Exception` Class

- OperationCanceledException - The exception that is thrown in a thread upon cancellation of an operation that the thread was executing.
- OutOfMemoryException - The exception that is thrown when there is not enough memory to continue the execution of a program.
- OverflowException - The exception that is thrown when an arithmetic, casting, or conversion operation in a checked context results in an overflow.
- PlatformNotSupportedException - The exception that is thrown when a feature does not run on a particular platform.
- RankException - The exception that is thrown when an array with the wrong number of dimensions is passed to a method.
- StackOverflowException - The exception that is thrown when the execution stack overflows because it contains too many nested method calls. This class cannot be inherited.
- SystemException - Defines the base class for predefined exceptions in the System namespace.
- TimeoutException - The exception that is thrown when the time allotted for a process or operation has expired.
- TimeZoneNotFoundException - The exception that is thrown when a time zone cannot be found.
- TypeAccessException - The exception that is thrown when a method attempts to use a type that it does not have access to.
- TypeInitializationException - The exception that is thrown as a wrapper around the exception thrown by the class initializer. This class cannot be inherited.
- TypeLoadException - The exception that is thrown when type-loading failures occur.
- TypeUnloadedException - The exception that is thrown when there is an attempt to access an unloaded class.
- UnauthorizedAccessException - The exception that is thrown when the operating system denies access because of an I/O error or a specific type of security error.
- UriFormatException - The exception that is thrown when an invalid Uniform Resource Identifier (URI) is detected.
- UriTemplateMatchException - Represents an error when matching a Uri to a UriTemplateTable.

Below is an example of the `OverflowException` when an integer value becomes to large to be held in an `Int32`. See next slide for the code. One thing you have to be aware of when using this exception is that when you compile the code you need to turn on the `/checked+` switch; otherwise, you will not receive any exceptions!

```
csc /checked+ myprog.cs
```

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Exception` Class


```
using System;

class MainProgram {

 public static void Main() {

  Int32 iValue = 2147483647; //This is the maximum value that can be stored in an Int32.

  //Attempt to add one to this value.
  try {

    iValue += 1;
    Console.WriteLine(iValue);

  }
  catch(OverflowException e) {
   Console.WriteLine("OverflowException Occurred!!");
   Console.WriteLine(e.Message);
  }
  catch(Exception e) {
   Console.WriteLine("General Exception Occurred!!");
   Console.WriteLine(e.Message);
  }
  finally {
   Console.WriteLine("In FINALLY!");
  }

 }

}
```

The output looks like this:

```
OverflowException Occurred!!
Arithmetic operation resulted in an overflow.
In FINALLY!
```

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Exception` Class

Now, in order to create your own exception class, you need to derive from the `Exception` class itself. Once you do this, your own exception class can override Below is an example:

```
using System;

//Create my own funky exception class
class MyFunkyException : Exception {

 public MyFunkyException() : base("A FUNKY EXCEPTION OCCURED!!") {
 }

}
```

…continued on next slide…

Note that the only constructor we deal with is the one with no parameters. There are other constructors you may want to code for.

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Exception` Class


```
class MainProgram {

 public static void Main() {

  Int32 iValue = 5;

  //Attempt to add one to this value.
  try {

    iValue += 1;
    if (iValue == 6) {
     throw new MyFunkyException();
    }
    else {
     Console.WriteLine(iValue);
    }

  }
  catch(MyFunkyException e) {
   Console.WriteLine("MyFunkyException Occurred!!");
   Console.WriteLine(e.Message);
  }
  catch(Exception e) {
   Console.WriteLine("General Exception Occurred!!");
   Console.WriteLine(e.Message);
  }
  finally {
   Console.WriteLine("In FINALLY!");
  }

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Derived from the `Attribute` Class

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Attribute` Class

We talked about how to use attributes in the C# Programming III presentation, so we won't repeat it here. Below is a list of classes derived from the `Attribute` class.

- **Attribute - Represents the base class for custom attributes.**
- AttributeUsageAttribute - Specifies the usage of another attribute class. This class cannot be inherited.
- CLSCompliantAttribute - Indicates whether a program element is compliant with the Common Language Specification (CLS). This class cannot be inherited.
- ContextStaticAttribute - Indicates that the value of a static field is unique for a particular context.
- FlagsAttribute - Indicates that an enumeration can be treated as a bit field; that is, a set of flags.
- LoaderOptimizationAttribute - Used to set the default loader optimization policy for the main method of an executable application.
- MTAThreadAttribute - Indicates that the COM threading model for an application is multithreaded apartment (MTA).
- NonSerializedAttribute - Indicates that a field of a serializable class should not be serialized. This class cannot be inherited.
- ObsoleteAttribute - Marks the program elements that are no longer in use. This class cannot be inherited.
- ParamArrayAttribute - Indicates that a method will allow a variable number of arguments in its invocation. This class cannot be inherited.
- SerializableAttribute - Indicates that a class can be serialized. This class cannot be inherited.
- STAThreadAttribute - Indicates that the COM threading model for an application is single-threaded apartment (STA).
- ThreadStaticAttribute - Indicates that the value of a static field is unique for each thread.

For those of you looking for the `ConditionalAttribute`, this is located in the `System.Diagnostics` namespace.

As an example, let's look at the `ParamArrayAttribute`. This attribute allows us to indicate that our method can take a variable number of parameters. See next slide for example.

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Attribute` Class


```
using System;

class MainProgram {

 //Create a SQL string pulling data from the database based on the drug codes
 public static String CreateSQLCode(params String[] psDrugCodes) {

  //Initialize the SQL string
  String sSQL = "SELECT * FROM FACT_TABLE WHERE NDC_CODE IN (";

  //Add individual drug codes delimited by a comma and surrounded by tick marks.
  for(Int32 indx=0;indx<psDrugCodes.Length;indx++) {
   sSQL = sSQL + "'" + psDrugCodes[indx] + "',";
  }

  //Remove ending comma
  sSQL = sSQL.Substring(0,sSQL.Length-1);

  //Add ending parenthesis to finalize the syntax
  sSQL += ")";

  return(sSQL);
 }

 public static void Main() {

  String sSQLCode1 = CreateSQLCode("123","456","789");
  String sSQLCode2 = CreateSQLCode("123","456","789","012","345","678");
  Console.WriteLine(sSQLCode1);
  Console.WriteLine(sSQLCode2);

 }

}
```

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `Attribute` Class

You'll note that we used the `params` keyword instead of the `ParamArrayAttribute` attribute. This is because C# requires the `params` keyword instead. If you attempt to use the attribute, you will receive the following error message:

```
error CS0674: Do not use 'System.ParamArrayAttribute'. Use the 'params' keyword instead.
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Derived from the `EventArgs` Class

Copyright ©2011 sheepsqueezers.com

Classes Derived from the `EventArgs` Class

The `EventArgs` class is the base class for classes containing event data. The following is a classes derived from the `EventArgs` class:

- **`EventArgs - EventArgs is the base class for classes containing event data.`**
- `AssemblyLoadEventArgs - Provides data for the AssemblyLoad event.`
- `ConsoleCancelEventArgs - Provides data for the Console.CancelKeyPress event. This class cannot be inherited.`
- `ResolveEventArgs - Provides data for loader resolution events, such as the TypeResolve, ResourceResolve, ReflectionOnlyAssemblyResolve, and AssemblyResolve events.`
- `UnhandledExceptionEventArgs - Provides data for the event that is raised when there is an exception that is not handled in any application domain.`

See the section on Events in the C# Programming II presentation for more on how to use these.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception`, `Attribute` or `EventArgs` Classes
    - → `Object`

Copyright ©2011 sheepsqueezers.com

Object

The `Object` class is the base class of all classes in the .NET Framework. All of the methods, properties, etc. within this class appear in all of the other classes in the framework. Here are the entities within the `Object` class:

Constructors

1. `public Object()` – initializes a new instance of the `Object` class. There is a very ominous statement on Microsoft's website: *This constructor is called by constructors in derived classes…* I'm confused about this comment…will come back to it later…

Methods (public)

1. `Equals(Object)` – determines whether the specified `Object` is equal to the current `Object`.
2. `Equals(Object1,Object2)` – (static) Determines whether the specified objects are equal.
3. `GetHashCode()` – returns a numeric value (`int`) that is used to identify an object during equality testing. It can serve as an index for an object in a collection. Microsoft's website goes on to say that *the default implementation of the GetHashCode method does not guarantee unique return values for different objects*. It continues with *the default implementation of this method must not be used as a unique object identifier for hashing purposes*.
4. `GetType()` – returns a `System.Type` object containing the exact runtime type of the object.

Copyright ©2011 sheepsqueezers.com

Object

5. `ReferenceEquals(Object1,Object2)` – (static) Determines whether the specified Object instances are the same instance and returns a Boolean.
6. `ToString()` – returns a string that represents the current object. Note that several other classes override this method to produce a more suitable string value.

Methods (protected)

1. `Finalize()` – allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. There is a note on the MS website: *Because the C# compiler does not allow you to directly implement the Finalize method, a C# destructor automatically calls the destructor of its base class.* And goes further: *Object.Finalize does nothing by default. It must be overridden by a derived class only if necessary, because reclamation during garbage collection tends to take much longer if a Finalize operation must be run.*
2. `MemberwiseClose()` – returns an Object that is a shallow copy of the current Object. *The MemberwiseClone method creates a shallow copy by creating a new object, and then copying the nonstatic fields of the current object to the new object. If a field is a value type, a bit-by-bit copy of the field is performed. If a field is a reference type, the reference is copied but the referred object is not; therefore, the original object and its clone refer to the same object.*

Note that you do not have to include ": Object" in your own class definitions since it is automatically included by the compiler.

Copyright ©2011 sheepsqueezers.com

sheepsqueezers.com

One important thing you need to be aware of is boxing and unboxing. Boxing is the act of turning a value type (like `Int32` or `Boolean`) into an Object type. Unboxing does just the opposite. The act of boxing and unboxing may not be clear in your code (especially for beginners like me!), so you could unwittingly cause boxing and unboxing to occur causing a slow down in the execution of your code. Apparently, you can use the `ILDasm.exe` application (installed by default when you install Visual Studio) to see the Microsoft Intermediate Language (MSIL or IL) code and search for the word "box". Here is some example code on boxing an integer:

```
using System;

class MainProgram {

 public static void Main() {

  Int32 iValue = 45;
  Object oValue = (Object) iValue;

 }

}
```

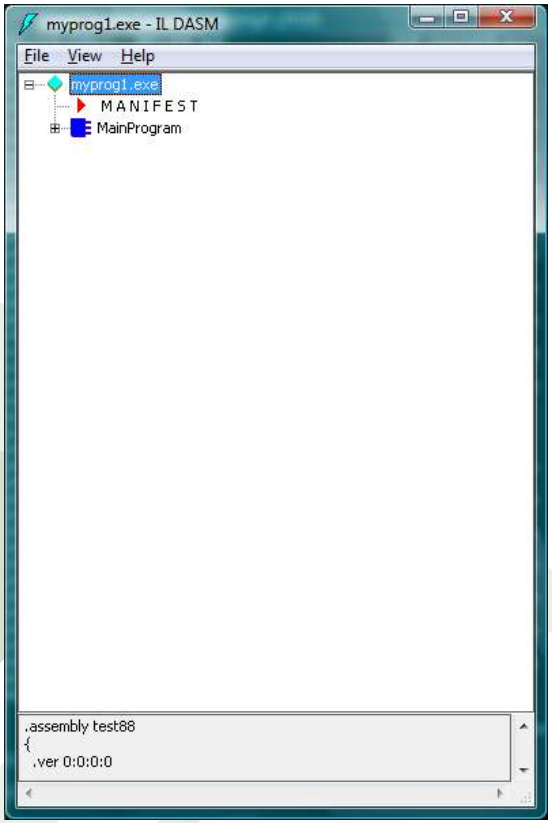
Now, compile this normally and then open up the .exe file using ILDasm.exe:

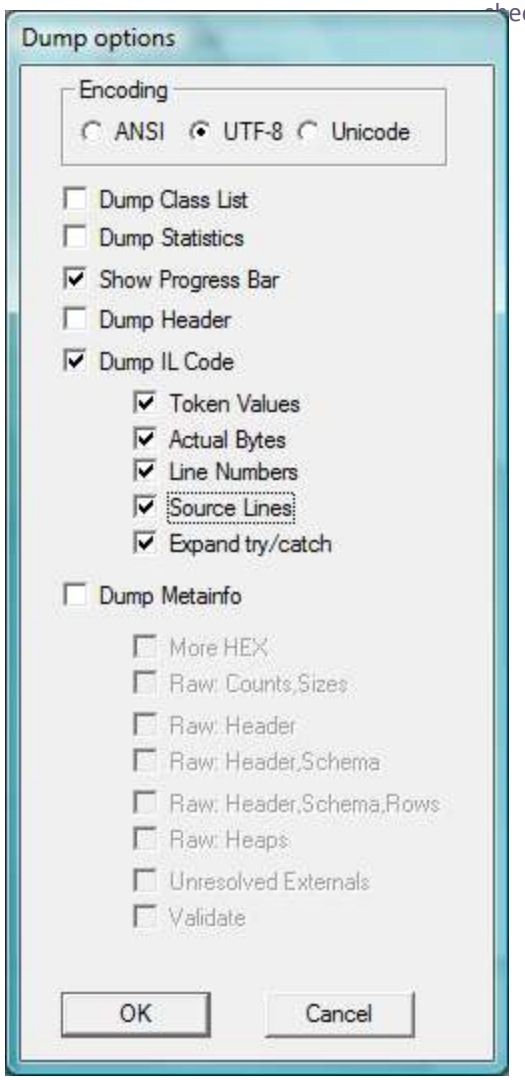
```
csc myprog1.cs
ILDasm myprog1.exe
```

Copyright ©2011 sheepsqueezers.com

sheepsqueezers.com

What you will see is the following GUI (on the left):





Click on File…Dump, and ensure that the following checkboxes are checked (see the right). Click the OK button and save to your hard-drive.

You'll notice that there are two files on your hard-drive, a .il file and a .res file. The .res file is the assembly metadata and the .il file is the code. Open up the .il file in Notepad.

Copyright ©2011 sheepsqueezers.com

Next, search for the word "box":

```
IL_0000:  /* 00    |                  */ nop
IL_0001:  /* 1F    | 2D               */ ldc.i4.s   45
IL_0003:  /* 0A    |                  */ stloc.0
IL_0004:  /* 06    |                  */ ldloc.0
IL_0005:  /* 8C    | (01)000004       */ box        [mscorlib/*23000001*/]System.Int32/*01000004*/
IL_000a:  /* 0B    |                  */ stloc.1
IL_000b:  /* 2A    |                  */ ret
```

As you see above, there is boxing occurring. Apparently, you should do something about it. ☺

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception`, `Attribute` or `EventArgs` Classes
    - → `AppDomain/AppDomainManager/AppDomainSetup`

Copyright ©2011 sheepsqueezers.com

AppDomain/AppDomainManager/AppDomainSetup

The `AppDomain` class *represents an application domain, which is an isolated environment where applications execute and help provide isolation, unloading, and security boundaries for executing managed code.* Use application domains to isolate tasks that might bring down a process. Microsoft's website goes on to say, *If an assembly is loaded into the default application domain, it cannot be unloaded from memory while the process is running. However, if you open a second application domain to load and execute the assembly, the assembly is unloaded when that application domain is unloaded. Use this technique to minimize the working set of long-running processes that occasionally use large DLLs.*

If you have not created a separate application domain, your program is running in the *current domain*. Shattering, I know, but it's true! ☺

Since this is a very advanced topic, I'll skip this for now. I will mention two properties that we will see in the next section: `ActivationContext` and `ApplicationIdentity`:

1. `ApplicationContext` – returns an `ActivationContext` object in the `System` namespace containing the activation context for the current domain. Returns `null` if the domain has no activation context.
2. `ApplicationIdentity` – returns an `ApplicationIdentity` object in the `System` namespace containing the identity of the application in the application domain.

Copyright ©2011 sheepsqueezers.com

AppDomain/AppDomainManager/AppDomainSetup

The `AppDomainManager` class provides a managed equivalent of an unmanaged host. According to Richter, *The purpose of the AppDomainManager-derived class is to allow a host to maintain control even when an add-in tried to create AppDomains of its own. When the code in the process tries to create a new AppDomain, the AppDomainManager-derived object in that AppDomain can modify security and configuration settings. It can also decide to fail an AppDomain creation, or it can decide to return a reference to an existing AppDomain instead. When a new AppDomain is created, the CLR creates a new AppDomainManager-derived object in the AppDomain. This object can also modify configuration settings, how execution context is flowed between threads, and permissions granted to an assembly.*

Yikes…this is confusing as well…will come back to this later…

Richter recommends the book *Customizing the Microsoft .NET Framework Common Language Runtime*, by Stephen Pratschner.

The `AppDomainSetup` class represents assembly binding information that can be added to an instance of `AppDomain`. Microsoft's website goes on to state: *Changing the properties of an AppDomainSetup instance does not affect any existing AppDomain. It can affect only the creation of a new AppDomain, when the CreateDomain method is called with the AppDomainSetup instance as a parameter.*

…I'll come back to this later on…

Copyright ©2011 sheepsqueezers.com

AppDomain/AppDomainManager/AppDomainSetup

Below is an example of how to get the `FriendlyName` (the name of the executable) of the currently running application. The static property `CurrentDomain` returns an `AppDomain` object which allows us to access the `FriendlyName` property:

```
using System;

class MainProgram {

 public static void Main() {

  String sFriendlyName = AppDomain.CurrentDomain.FriendlyName;
  Console.WriteLine(sFriendlyName);

  String sApplicationName = AppDomain.CurrentDomain.SetupInformation.ApplicationName;
  Console.WriteLine(sApplicationName);

 }

}
```

This returns "test88.exe" twice, the name of the executable running this program.

Notice that we can get the name of the application using the `ApplicationName` property of the `AppDomainSetup` object returned by the `SetupInformation` property. The `AppDomain.CurrentDomain` syntax returns an `AppDomain` object.

Copyright ©2011 sheepsqueezers.com

AppDomain/AppDomainManager/AppDomainSetup

Now, you can run assemblies (.exe files) by using the `AppDomain`'s `ExecuteAssembly` method as shown below. Here I just run one of the integration programs:

```
using System;

class MainProgram {

 public static void Main() {

  AppDomain oCD = AppDomain.CurrentDomain;
  oCD.ExecuteAssembly("test25.exe");

 }

}
```

Note that there are `Load()` methods in `AppDomain` which allow you to load assemblies. These methods return an `Assembly` object which we will talk about later on.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception`, `Attribute` or `EventArgs` Classes
    - → `ActivationContext/ApplicationIdentity/ApplicationId`

Copyright ©2011 sheepsqueezers.com

ActivationContext/ApplicationIdentity/ApplicationId

The `ActivationContext` class identifies the activation context for the current application. *The activation context is used during manifest-based activation to set up the domain policy and provide an application-based security model.*

The `ApplicationIdentity` class provides the ability to uniquely identify a manifest-activated application.

The `ApplicationId` class contains information used to uniquely identify a manifest-based application. This class contains the properties `Culture`, `Name`, `ProcessorArchitecture`, `PublicKeyToken` and `Version`. Below is an ex

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception`, `Attribute` or `EventArgs` Classes

→ `Activator`

Copyright ©2011 sheepsqueezers.com

Activator

The `Activator` class contains methods to create types of objects locally or remotely, or to obtain references to existing remote objects.

You can use the `CreateInstance` methods to create instances of types dynamically. Will come back to this…

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

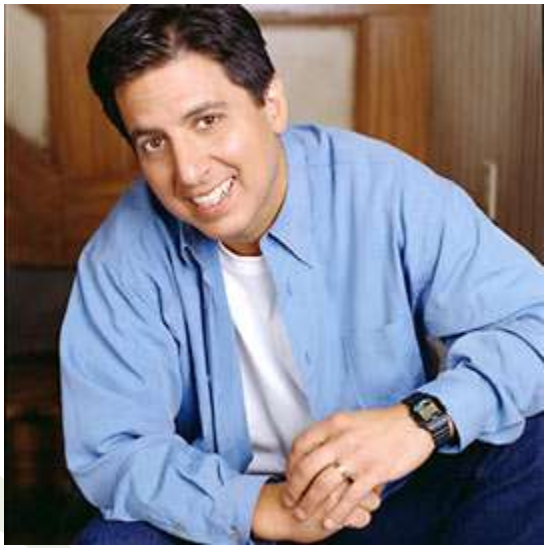
→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Array`

Copyright ©2011 sheepsqueezers.com

Array

The `Array` class provides methods for creating, manipulating, searching, and sorting arrays, thereby serving as the base class for all arrays in the common language runtime as well as making it a really bad-ass class! I mean, who doesn't love arrays. Here an array:



Here's another array:

…tee-hee…

sheepsqueezers.com

Copyright ©2011 sheepsqueezers.com

Array

From Microsoft's website: *The Array class is the base class for language implementations that support arrays. However, only the system and compilers can derive explicitly from the Array class. Users should employ the array constructs provided by the language.*

An element is a value in an Array. The length of an Array is the total number of elements it can contain. The rank of an Array is the number of dimensions in the Array. The lower bound of a dimension of an Array is the starting index of that dimension of the Array; a multidimensional Array can have different bounds for each dimension. ***An array can have a maximum of 32 dimensions.***

Type objects provide information about array type declarations. Array objects with the same array type share the same Type object.

Type.IsArray and Type.GetElementType might not return the expected results with Array because if an array is cast to the type Array, the result is an object, not an array. That is, typeof(System.Array).IsArray returns false, and typeof(System.Array).GetElementType returns null.

Unlike most classes, Array provides the CreateInstance method, instead of public constructors, to allow for late bound access.

The Array.Copy method copies elements not only between arrays of the same type but also between standard arrays of different types; it handles type casting automatically.

…continued on next slide…

Copyright ©2011 sheepsqueezers.com

Array

Some methods, such as CreateInstance, Copy, CopyTo, GetValue, and SetValue, provide overloads that accept 64-bit integers as parameters to accommodate large capacity arrays. LongLength and GetLongLength return 64-bit integers indicating the length of the array.

The Array is not guaranteed to be sorted. You must sort the Array prior to performing operations (such as BinarySearch) that require the Array to be sorted.

Using an Array object of pointers in native code is not supported and will throw a NotSupportedException for several methods.

Now, the `Array` class is defined as abstract, and I'm not sure why! I'll have to come back to this!

Properties

- IsFixedSize - Gets a value indicating whether the Array has a fixed size.
- IsReadOnly - Gets a value indicating whether the Array is read-only.
- IsSynchronized - Gets a value indicating whether access to the Array is synchronized (thread safe).
- Length - Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
- LongLength - Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
- Rank - Gets the rank (number of dimensions) of the Array.
- SyncRoot - Gets an object that can be used to synchronize access to the Array.

Note that the `SyncRoot` property returns an `Object` that can be used with the Monitor locking (lock). This is similar to creating your own object when using threads: `Object oLock = new Object();`.

Copyright ©2011 sheepsqueezers.com

Methods of the Array Class

- AsReadOnly<T> - Returns a read-only wrapper for the specified array.
- BinarySearch* - Searches an array for an object and returns the integer index if found; otherwise, a negative integer is returned. There are 8 overloads to this method.
- Clear - Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
- Clone - Creates a shallow copy of the Array.
- ConstrainedCopy - Copies a range of elements from an Array starting at the specified source index and pastes them to another Array starting at the specified destination index. Guarantees that all changes are undone if the copy does not succeed completely.
- ConvertAll<TInput, TOutput> - Converts an array of one type to an array of another type.
- Copy* - Copies a range of elements from one array into another array. There are 4 overloads to this method.
- CopyTo* - Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit/64-bit integer. There are 2 overloads for this method.
- CreateInstance* - Creates an Array of the specified Type and length, with zero-based indexing. There are 6 overloads to this method.
- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Exists<T> - Determines whether the specified array contains elements that match the conditions defined by the specified predicate.
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- Find<T> - Searches for an element that matches the conditions defined by the specified predicate, and returns the first occurrence within the entire Array.
- FindAll<T> - Retrieves all the elements that match the conditions defined by the specified predicate.
- FindIndex* - Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the first occurrence within the entire Array. There are 3 overloads to this method.
- FindLast<T> - Searches for an element that matches the conditions defined by the specified predicate, and returns the last occurrence within the entire Array.
- FindLastIndex* - Searches for an element that matches the conditions defined by the specified predicate, and returns the zero-based index of the last occurrence within the entire Array. There are 3 overloads to this method.
- ForEach<T> - Performs the specified action on each element of the specified array.
- GetEnumerator - Returns an IEnumerator for the Array.
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetLength - Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
- GetLongLength - Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
- GetLowerBound - Gets the lower bound of the specified dimension in the Array.

Copyright ©2011 sheepsqueezers.com

Methods of the Array Class (continued)

- GetType - Gets the Type of the current instance. (Inherited from Object.)
- GetUpperBound - Gets the upper bound of the specified dimension in the Array.
- GetValue* - Gets the value of the array at the specified position. There are 8 overloads to this method.
- IndexOf* - Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array. There are 6 overloads to this method.
- Initialize - Initializes every element of the value-type Array by calling the default constructor of the value type.
- LastIndexOf* - Searches for the specified object and returns the index of the last occurrence within the entire one-dimensional Array. There are 6 overloads to this method.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- Resize<T> - Changes the number of elements of an array to the specified new size.
- Reverse* - Reverses the sequence of the elements in the entire one-dimensional Array or a portion of it. There are 2 overloads to this method.
- SetValue* - Sets the value of the array at the specified position. There are 8 overloads to this method.
- Sort* - Sorts an array. There are 17 overloads to this method.
- ToString - Returns a string that represents the current object. (Inherited from Object.)
- TrueForAll<T> - Determines whether every element in the array matches the conditions defined by the specified predicate.

There are several mentions of a "predicate", so let's look into that to start with. A predicate is a delegate with the following definition:

```
public delegate bool Predicate<in T>(T obj)
```

Note that this delegate points to a method that will accept an `Object` obj of type T which should match the array's type T. This method returns a `Boolean`. For example, let's create an array containing `Double`s and the method that will have the same signature as the delegate. See the next slide for the example code.

Copyright ©2011 sheepsqueezers.com

Array


```
using System;

class MainProgram {

 public static void Main() {

  Double[] aDbls = new Double[] {3.1415D, 2.1828D, 1.4142D, 1.732D};
  Double dFirstFoundDoubleValue = Array.Find(aDbls,GreaterThanTwo);
  Console.WriteLine(dFirstFoundDoubleValue); //returns 3.1415 since it is the first found!

 }

 public static Boolean GreaterThanTwo(Double pDbl) {

  if (pDbl >= 2) {
   return(true);
  }
  else {
   return(false);
  }

 }

}
```

As you can see, the code in red declares the delegate method. There is no need to declare a proper delegate using the delegate keyword since the compiler will know what to do. Note that we have to use the syntax **`Array.`**`Find` since `Find` is a static method in the array class. Also, note that many of the methods in the Array class are static.

Let's see an example of how to sort an array. See next slide.

Copyright ©2011 sheepsqueezers.com

Array


```
using System;

class MainProgram {

 public static void Main() {

  Double[] aDbls = new Double[] {3.1415D, 2.1828D, 1.4142D, 1.732D};
  Array.Sort(aDbls);

  foreach(Double dVal in aDbls) {
   Console.WriteLine(dVal.ToString());
  }

 }

}
```

Note that we have to use the syntax **`Array`**`.Sort` since `Sort` is a static method in the array class.

Now, as you know, you can set an array element by using indexing and the equals-sign: `aDbls[5]=5.4321;`. But, there are several SetValue* methods that allow you to do something like this: `aDbls.SetValue(5.4321,5);`. I guess the choice is yours as to which you pick.

If you need to "reset" you array to zero, null or false, you can use the Clear method: `Array.Clear(aDbls,0,aDbls.Length);`. The second parameter is the starting array index and the third parameter is the length of the section you want cleared.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `BitConverter`

Copyright ©2011 sheepsqueezers.com

BitConverter

The `BitConverter` class is a completely static class that converts base data types to an array of bytes, and an array of bytes to base data types. Microsoft's website goes on to say, *The BitConverter class helps manipulate value types in their fundamental form, as a series of bytes. A byte is defined as an 8-bit unsigned integer. The BitConverter class includes static methods to convert each of the primitive types to and from an array of bytes, as the following table illustrates.*

Methods (static)

- DoubleToInt64Bits - Converts the specified double-precision floating point number to a 64-bit signed integer.
- GetBytes(Boolean) - Returns the specified Boolean value as an array of bytes.
- GetBytes(Char) - Returns the specified Unicode character value as an array of bytes.
- GetBytes(Double) - Returns the specified double-precision floating point value as an array of bytes.
- GetBytes(Int16) - Returns the specified 16-bit signed integer value as an array of bytes.
- GetBytes(Int32) - Returns the specified 32-bit signed integer value as an array of bytes.
- GetBytes(Int64) - Returns the specified 64-bit signed integer value as an array of bytes.
- GetBytes(Single) - Returns the specified single-precision floating point value as an array of bytes.
- GetBytes(UInt16) - Returns the specified 16-bit unsigned integer value as an array of bytes.
- GetBytes(UInt32) - Returns the specified 32-bit unsigned integer value as an array of bytes.
- GetBytes(UInt64) - Returns the specified 64-bit unsigned integer value as an array of bytes.
- Int64BitsToDouble - Converts the specified 64-bit signed integer to a double-precision floating point number.
- ToBoolean - Returns a Boolean value converted from one byte at a specified position in a byte array.
- ToChar - Returns a Unicode character converted from two bytes at a specified position in a byte array.
- ToDouble - Returns a double-precision floating point number converted from eight bytes at a specified position in a byte array.
- ToInt16 - Returns a 16-bit signed integer converted from two bytes at a specified position in a byte array.
- ToInt32 - Returns a 32-bit signed integer converted from four bytes at a specified position in a byte array.
- ToInt64 - Returns a 64-bit signed integer converted from eight bytes at a specified position in a byte array.
- ToSingle - Returns a single-precision floating point number converted from four bytes at a specified position in a byte array.

Copyright ©2011 sheepsqueezers.com

Methods (static) (continued)

- ToString(Byte[]) - Converts the numeric value of each element of a specified array of bytes to its equivalent hexadecimal string representation.
- ToString(Byte[], Int32) - Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.
- ToString(Byte[], Int32, Int32) - Converts the numeric value of each element of a specified subarray of bytes to its equivalent hexadecimal string representation.
- ToUInt16 - Returns a 16-bit unsigned integer converted from two bytes at a specified position in a byte array.
- ToUInt32 - Returns a 32-bit unsigned integer converted from four bytes at a specified position in a byte array.
- ToUInt64 - Returns a 64-bit unsigned integer converted from eight bytes at a specified position in a byte array.

Note that conversions other than byte conversions can be done using casting or using the methods of the `Convert` class.

For example, given a `Double` value, you can convert that value to an array of length 8 of bytes using the `BitConverter.GetDouble()` method and then you can then use the `BitConverter.ToString()` method to get the hex representation of that value.

```
using System;

class MainProgram {

 public static void Main() {

  Double dSqrtTwo = Math.Sqrt(2);
  Byte[] bArray = BitConverter.GetBytes(dSqrtTwo);
  Console.WriteLine(BitConverter.ToString(bArray)); //CD-3B-7F-66-9E-A0-F6-3F
  Console.WriteLine(BitConverter.ToDouble(bArray,0)); //1.4142135623731

 }

}
```

Copyright ©2011 sheepsqueezers.com

Fields (static)

- IsLittleEndian – Indicates the byte order ("endianness") in which data is stored in this computer architecture.

There is only one static field in the `BitConverter` class. It returns true if the architecture is little-endian; otherwise, it returns false. *Different computer architectures store data using different byte orders. "Big-endian" means the most significant byte is on the left end of a word. "Little-endian" means the most significant byte is on the right end of a word.*

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `Buffer`

Copyright ©2011 sheepsqueezers.com

Buffer

sheepsqueezers.com

The `Buffer` class manipulates arrays created from the *primitive* data types such as `Boolean`, `Char`, `SByte`, `Byte`, `Int16`, `UInt16`, `Int32`, `UInt32`, `Int64`, `UInt64`, `IntPtr`, `UIntPtr`, `Single`, and `Double`. According to Microsoft's website: *Each primitive type is treated as a series of bytes without regard to any behavior or limitation associated with the primitive type. Buffer provides methods to copy bytes from one array of primitive types to another array of primitive types, get a byte from an array, set a byte in an array, and obtain the length of an array. This class provides better performance for manipulating primitive types than similar methods in the System.Array class.*

Note that `Buffer` is a static class with four static methods, as detailed below.

Methods (static)

- BlockCopy - Copies a specified number of bytes from a source array starting at a particular offset to a destination array starting at a particular offset.
- ByteLength - Returns the number of bytes in the specified array.
- GetByte - Retrieves the byte at a specified location in a specified array.
- SetByte - Assigns a specified value to a byte at a particular location in a specified array.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `CharEnumerator`

Copyright ©2011 sheepsqueezers.com

CharEnumerator

The `CharEnumerator` class supports iterating over a `String` object and reading its individual characters. Microsoft's website states: *A CharEnumerator provides read-only access to the characters in a referenced String object. For example, the foreach statement of the Microsoft Visual Basic and C# programming languages, which iterates through the elements of a collection, retrieves a CharEnumerator from a String object in order to iterate through the characters in that object.* ***There is no public constructor for CharEnumerator. Instead, call a String object's GetEnumerator method to obtain a CharEnumerator that is initialized to reference the string.*** *A CharEnumerator maintains an internal index to the characters in the string the CharEnumerator references. The state of the index is invalid when it references a character position logically before the first character or after the last character in the string, and valid when it references a character within the string. The index is initialized to a position logically before the first character, and is set to a position after the last character when the iteration is complete. An exception is thrown if you attempt to access a character while the index is invalid. The MoveNext method increments the index by one, so the first and subsequent characters are accessed in turn. The Reset method sets the index to a position logically before the first character. The Current property retrieves the character currently referenced by index. The Clone method creates a copy of the CharEnumerator.*

Properties

- Current – Gets the currently referenced character in the string enumerated by the CharEnumerator object.

Copyright ©2011 sheepsqueezers.com

CharEnumerator

Methods (instance)

- Clone - Creates a copy of the current CharEnumerator object.
- Dispose - Releases all resources used by the current instance of the CharEnumerator class.
- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- MoveNext - Increments the internal index of the current CharEnumerator object to the next character of the enumerated string.
- Reset - Initializes the index to a position logically before the first character of the enumerated string.
- ToString - Returns a string that represents the current object. (Inherited from Object.)

Note that the two most important methods are `MoveNext` and `Reset`. Here is an example:

```
using System;

class MainProgram {

 public static void Main() {

  String sSaying = "Now is the time for all good men to come to the aid of their country!";
  CharEnumerator ceSaying = sSaying.GetEnumerator();
  while(ceSaying.MoveNext()) {
   Console.WriteLine(ceSaying.Current);
  }

 }

}
```

Copyright ©2011 sheepsqueezers.com

CharEnumerator

According to the manual, you can do a similar thing with the foreach construct:

```
using System;

class MainProgram {

 public static void Main() {

  String sSaying = "Now is the time for all good men to come to the aid of their country!";
  foreach(Char ch in sSaying) {
   Console.WriteLine(ch);
  }

 }

}
```

Which is better or faster, I cannot say at this point.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Console`

Copyright ©2011 sheepsqueezers.com

Console

The `Console` class represents the standard input, output, and error streams for console applications. This `Console` class is static. According to Microsoft's website: *The console is an operating system window where users interact with the operating system or a text-based console application by entering text input through the computer keyboard, and reading text output from the computer terminal. For example, in Windows the console is called the command prompt window and accepts MS-DOS commands. The Console class provides basic support for applications that read characters from, and write characters to, the console.*

Note also that: *When a console application starts, the operating system automatically associates three I/O streams with the console. Your application can read user input from the standard input stream; write normal data to the standard output stream; and write error data to the standard error output stream. These streams are presented to your application as the values of the In, Out, and Error properties. By default, the value of the In property is a System.IO.TextReader object, and the values of the Out and Error properties are System.IO.TextWriter objects. However, you can set these properties to streams that do not represent the console; for example, you can set these properties to streams that represent files. To redirect the standard input, standard output, or standard error stream, call the SetIn, SetOut, or SetError method, respectively. I/O operations using these streams are synchronized, which means multiple threads can read from, or write to, the streams.* This is similar to StdIn, StdOut and StdError in *nix Platforms.

Copyright ©2011 sheepsqueezers.com

Properties (static)

- BackgroundColor - Gets or sets the background color of the console.
- BufferHeight - Gets or sets the height of the buffer area.
- BufferWidth - Gets or sets the width of the buffer area.
- CapsLock - Gets a value indicating whether the CAPS LOCK keyboard toggle is turned on or turned off.
- CursorLeft - Gets or sets the column position of the cursor within the buffer area.
- CursorSize - Gets or sets the height of the cursor within a character cell.
- CursorTop - Gets or sets the row position of the cursor within the buffer area.
- CursorVisible - Gets or sets a value indicating whether the cursor is visible.
- Error - Gets the standard error output stream.
- ForegroundColor - Gets or sets the foreground color of the console.
- In - Gets the standard input stream.
- InputEncoding - Gets or sets the encoding the console uses to read input.
- KeyAvailable - Gets a value indicating whether a key press is available in the input stream.
- LargestWindowHeight - Gets the largest possible number of console window rows, based on the current font and screen resolution.
- LargestWindowWidth - Gets the largest possible number of console window columns, based on the current font and screen resolution.
- NumberLock - Gets a value indicating whether the NUM LOCK keyboard toggle is turned on or turned off.
- Out - Gets the standard output stream.
- OutputEncoding - Gets or sets the encoding the console uses to write output.
- Title - Gets or sets the title to display in the console title bar.
- TreatControlCAsInput - Gets or sets a value indicating whether the combination of the Control modifier key and C console key (CTRL+C) is treated as ordinary input or as an interruption that is handled by the operating system.
- WindowHeight - Gets or sets the height of the console window area.
- WindowLeft - Gets or sets the leftmost position of the console window area relative to the screen buffer.
- WindowTop - Gets or sets the top position of the console window area relative to the screen buffer.
- WindowWidth - Gets or sets the width of the console window.

Copyright ©2011 sheepsqueezers.com

Methods (static)

- Beep() - Plays the sound of a beep through the console speaker. This is not supported on 64-bit Windows Vista and Windows XP.
- Beep(Int32, Int32) - Plays the sound of a beep of a specified frequency and duration through the console speaker.
- Clear - Clears the console buffer and corresponding console window of display information.
- MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32) - Copies a specified source area of the screen buffer to a specified destination area.
- MoveBufferArea(Int32, Int32, Int32, Int32, Int32, Int32, Char, ConsoleColor, ConsoleColor) - Copies a specified source area of the screen buffer to a specified destination area.
- OpenStandardError() - Acquires the standard error stream.
- OpenStandardError(Int32) - Acquires the standard error stream, which is set to a specified buffer size.
- OpenStandardInput() - Acquires the standard input stream.
- OpenStandardInput(Int32) - Acquires the standard input stream, which is set to a specified buffer size.
- OpenStandardOutput() - Acquires the standard output stream.
- OpenStandardOutput(Int32) - Acquires the standard output stream, which is set to a specified buffer size.
- Read - Reads the next character from the standard input stream.
- ReadKey() - Obtains the next character or function key pressed by the user. The pressed key is displayed in the console window.
- ReadKey(Boolean) - Obtains the next character or function key pressed by the user. The pressed key is optionally displayed in the console window.
- ReadLine - Reads the next line of characters from the standard input stream.
- ResetColor - Sets the foreground and background console colors to their defaults.
- SetBufferSize - Sets the height and width of the screen buffer area to the specified values.
- SetCursorPosition - Sets the position of the cursor.
- SetError - Sets the Error property to the specified TextWriter object.
- SetIn - Sets the In property to the specified TextReader object.
- SetOut - Sets the Out property to the specified TextWriter object.
- SetWindowPosition - Sets the position of the console window relative to the screen buffer.
- SetWindowSize - Sets the height and width of the console window to the specified values.
- Write - writes the text representations of the specified parameter to the stadard output stream. There are 18 overloads to this method.
- WriteLine - writes the text representation of the specified parameter to the standard output stream. If no parameter is provided, a newline is written out. There are 19 overloads to this method.

Copyright ©2011 sheepsqueezers.com

Console

Here is a simple example that prints the entered name to the console screen at a certain position:

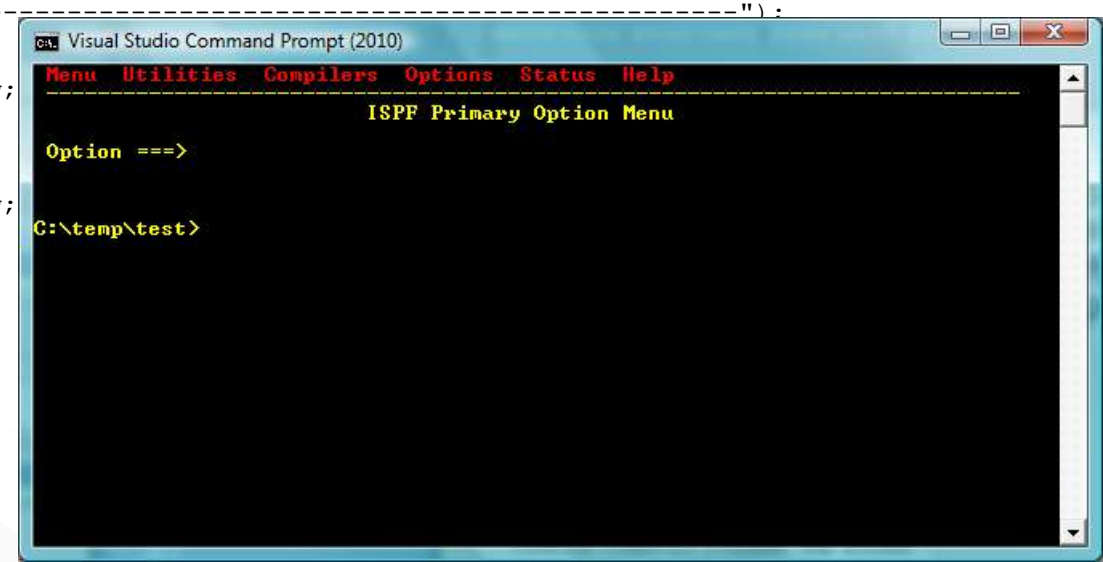
```
using System;

class MainProgram {

 public static void Main() {

  Console.Clear(); //clear the screen
  Console.WriteLine("Enter your name and hit the enter key now!"); //prompt for input
  String sName = Console.ReadLine();
  Console.Clear(); //clear the screen
  Console.SetCursorPosition(10,10);
  Console.Write("Your name is: {0}!",sName);

 }

}
```

Copyright ©2011 sheepsqueezers.com

Console

Now, you can also set the color of the background and foreground using the `ConsoleColor` enumerator which contains the following members: `Black`, `DarkBlue`, `DarkGreen`, `DarkCyan`, `DarkRed`, `DarkMagenta`, `DarkYellow`, `Gray`, `DarkGray`, `Blue`, `Green`, `Cyan`, `Red`, `Magenta`, `Yellow`, `White`. Here is an example:

```
using System;

class MainProgram {

 public static void Main() {

  Console.Clear(); //clear the screen
  Console.SetCursorPosition(1,0); //(left,top)
  Console.ForegroundColor = ConsoleColor.Red;
  Console.Write("Menu  Utilities  Compilers  Options  Status  Help");
  Console.SetCursorPosition(1,1); //(left,top)
  Console.ForegroundColor = ConsoleColor.Yellow;
  Console.Write("--------------------------------------------------------------------------------");
  Console.SetCursorPosition(26,2); //(left,top)
  Console.ForegroundColor = ConsoleColor.Yellow;
  Console.Write("ISPF Primary Option Menu");
  Console.SetCursorPosition(1,4); //(left,top)
  Console.ForegroundColor = ConsoleColor.Yellow;
  Console.Write("Option ===> ");

  Console.WriteLine();
  Console.WriteLine();
  Console.WriteLine();
 }

}
```



Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `ContextBoundObject`

Copyright ©2011 sheepsqueezers.com

ContextBoundObject

The `ContextBoundObject` class defines the base class for all context-bound objects. According to Microsoft's website: *A context is a set of properties or usage rules that define an environment where a collection of objects resides. The rules are enforced when the objects are entering or leaving a context. Objects that are not context-bound are called agile objects. Objects that reside in a context and are bound to the context rules are called context-bound objects. Contexts are created during object activation. A new object is placed into an existing context or into a new context created using the attributes included in the metadata of the type. Context-bound classes are marked with a ContextAttribute that provides the usage rules. The context properties that can be added include policies regarding synchronization and transactions.*

According to Microsoft's website, the `ContextAttribute` is not intended to be used directly from your own code.

See the following web pages for more on context bound objects:

1. http://blogs.msdn.com/b/tilovell/archive/2011/02/07/contextboundobject-part-1-making-contexts.aspx
2. http://blogs.msdn.com/b/tilovell/archive/2011/02/10/contextboundobject-2-contexts-and-interception.aspx
3. http://www.codeproject.com/KB/architecture/aopsimplestscenario.aspx

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Convert`

Copyright ©2011 sheepsqueezers.com

Convert

The `Convert` class converts a base data type to another base data type. This is a completely static class. According to Microsoft's website: *The static methods of the Convert class are used to support conversion to and from the base data types in the .NET Framework. The supported base types are Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, DateTime and String.* Be aware of the following three exceptions that could occur when using this class's methods: `InvalidCastException`, `FormatException`, `OverflowException`. Note that: *In addition to supporting conversions between the base types,* ***the Convert method supports conversion of any custom type to any base type****. To do this, the custom type must implement the IConvertible interface, which defines methods for converting the implementing type to each of the base types. Conversions that are not supported by a particular type should throw an InvalidCastException.*

Fields (static)

- DBNull A constant that represents a database column that is absent of data; that is, database null.

Methods (static)

- ChangeType - Returns an object of the specified type and whose value is equivalent to the specified object. There are 4 overloads to this method.
- FromBase64CharArray - Converts a subset of a Unicode character array, which encodes binary data as base-64 digits, to an equivalent 8-bit unsigned integer array. Parameters specify the subset in the input array and the number of elements to convert.
- FromBase64String - Converts the specified string, which encodes binary data as base-64 digits, to an equivalent 8-bit unsigned integer array.
- GetTypeCode - Returns the TypeCode for the specified object.
- IsDBNull - Returns an indication whether the specified object is of type DBNull

Copyright ©2011 sheepsqueezers.com

Methods (static)

- ToBase64CharArray - Converts a subset of an 8-bit unsigned integer array to an equivalent subset of a Unicode character array encoded with base-64 digits. Parameters specify the subsets as offsets in the input and output arrays, and the number of elements in the input array to convert. There are 2 overloads to this method.
- ToBase64String - Converts an array of 8-bit unsigned integers to its equivalent string representation that is encoded with base-64 digits. There are 4 overloads to this method.
- ToBoolean - Converts the parameter to a Boolean, if possible, and may throw an InvalidCastException. There are 18 overloads to this method.
- ToByte - Converts the parameter to a Byte, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToChar - Converts the parameter to a Char, if possible, and may throw an InvalidCastException. There are 18 overloads to this method.
- ToDateTime - Converts the parameter to a date and time, if possible, and may throw an InvalidCastException. There are 18 overloads to this method many of which throw an InvalidCastException.
- ToDecimal - Converts the parameter to a Decimal, if possible, and may throw an InvalidCastException. There are 18 overloads to this method.
- ToDouble - Converts the parameter to a Double, if possible, and may throw an InvalidCastException. There are 18 overloads to this method.
- ToInt16 - Converts the parameter to a 16-bit signed integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToInt32 - Converts the parameter to a 32-bit signed integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToInt64 - Converts the parameter to a 64-bit signed integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToSByte - Converts the parameter to a 8-bit unsigned integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToSingle - Converts the parameter to a Single, if possible, and may throw an InvalidCastException. There are 18 overloads to this method.
- ToString - Converts the parameter to a String, if possible, and may throw an InvalidCastException. There are 36 overloads to this method.
- ToUInt16 - Converts the parameter to a 16-bit unsigned integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToUInt32 - Converts the parameter to a 32-bit unsigned integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.
- ToUInt64 - Converts the parameter to a 64-bit unsigned integer, if possible, and may throw an InvalidCastException. There are 19 overloads to this method.

Copyright ©2011 sheepsqueezers.com

Convert

One nice feature is to convert strings which contain numbers to numbers like Double, Int32, etc. For example,

```
using System;

class MainProgram {

 public static void Main() {

  String sValue1 = "3.1415";
  String sValue2 = "1024";
  Double dValue1 = Convert.ToDouble(sValue1);
  Int32 iValue2 = Convert.ToInt32(sValue2);

  Console.WriteLine("{0} - {1}",sValue1,dValue1); //3.1415 - 3.1415
  Console.WriteLine("{0} - {1}",sValue2,iValue2); //1024 - 1024

 }

}
```

Note that some of the methods take an IFormatProvider as a parameter to the method. This allows you to specify culture-specific information when writing out the value. For example, in the United States, we use a decimal point (e.g., 3.1415) whereas in France, say, they use a comma (e.g., 3,1415). An example follows on the next slide.

Copyright ©2011 sheepsqueezers.com

Convert


```
using System;

class MainProgram {

 public static void Main() {

  Double dValue = 1234.56789;
  String sCultureUS = "en-US"; //United States culture
  String sCultureFR = "fr-FR"; //France culture

  String sValueUS = Convert.ToString(dValue,new System.Globalization.CultureInfo(sCultureUS));
  String sValueFR = Convert.ToString(dValue,new System.Globalization.CultureInfo(sCultureFR));

  Console.WriteLine(sValueUS); //1234.56789
  Console.WriteLine(sValueFR); //1234,56789

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `DBNull`

Copyright ©2011 sheepsqueezers.com

DBNull

The `DBNull` class represents a non-existent value. According to Microsoft's website: *The DBNull class represents a nonexistent value. In a database, for example, a column in a row of a table might not contain any data whatsoever. That is, the column is considered to not exist at all instead of merely not having a value. A DBNull object represents the nonexistent column. Additionally, COM interop uses the DBNull class to distinguish between a VT_NULL variant, which indicates a nonexistent value, and a VT_EMPTY variant, which indicates an unspecified value.*

The DBNull type is a singleton class, which means only one DBNull object exists. The DBNull.Value member represents the sole DBNull object. DBNull.Value can be used to explicitly assign a nonexistent value to a database field, although most ADO.NET data providers automatically assign values of DBNull when a field does not have a valid value. You can determine whether a value retrieved from a database field is a DBNull value by passing the value of that field to the DBNull.Value.Equals method. However, some languages and database objects supply methods that make it easier to determine whether the value of a database field is DBNull.Value. These include the Visual Basic IsDBNull function, the Convert.IsDBNull method, the DataTableReader.IsDBNull method, and the IDataRecord.IsDBNull method.

Do not confuse the notion of null in an object-oriented programming language with a DBNull object. In an object-oriented programming language, null means the absence of a reference to an object. DBNull represents an uninitialized variant or nonexistent database column.

Copyright ©2011 sheepsqueezers.com

DBNull

It seems like the only important member is the field Value:

Fields (static)

- Value – represents the sole instance of the DBNull class.

Note that you can use the `Equals()` method along with the `DBNull.Value` field in order to compare it to another value: `DBNull.Value.Equals(myObject);`.

I thought that you could now use the "?" after data types in order to specify nulls when reading in data from a database: `Int32?`. Must look into this (`System.Data` Namespace??)!

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `Delegate`

Copyright ©2011 sheepsqueezers.com

Delegate

The `Delegate` class represents a delegate, which is a data structure that refers to a static method or to a class instance and an instance method of that class. Microsoft's website states: *The Delegate class is the base class for delegate types. However, only the system and compilers can derive explicitly from the Delegate class or from the MulticastDelegate class. It is also not permissible to derive a new type from a delegate type. The Delegate class is not considered a delegate type; it is a class used to derive delegate types. Most languages implement a delegate keyword, and compilers for those languages are able to derive from the MulticastDelegate class; therefore,* ***users should use the delegate keyword provided by the language.***

Please see the section on Delegates in the *C# Programming III* presentation.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Enum`

Copyright ©2011 sheepsqueezers.com

Enum

The `Enum` class provides the base class for enumerations. Microsoft's website states: *An enumeration is a set of named constants whose underlying type is any integral type except Char. If no underlying type is explicitly declared, Int32 is used. Enum is the base class for all enumerations in the .NET Framework. Enum provides methods for comparing instances of this class, converting the value of an instance to its string representation, converting the string representation of a number to an instance of this class, and creating an instance of a specified enumeration and value. You can also treat an enumeration as a bit field. For more information, see the Non-Exclusive Members and the Flags Attribute section and the FlagsAttribute topic.* ***Programming languages typically provide syntax to declare an enumeration that consists of a set of named constants and their values.***

Enumeration Best Practices

- We recommend that you use the following best practices when you define enumeration types:
- If you have not defined an enumeration member whose value is 0, consider creating a None enumerated constant. By default, the memory used for the enumeration is initialized to zero by the common language runtime. Consequently, if you do not define a constant whose value is zero, the enumeration will contain an illegal value when it is created.
- If there is an obvious default case that your application has to represent, consider using an enumerated constant whose value is zero to represent it. If there is no default case, consider using an enumerated constant whose value is zero to specify the case that is not represented by any of the other enumerated constants.
- Do not specify enumerated constants that are reserved for future use.
- When you define a method or property that takes an enumerated constant as a value, consider validating the value. The reason is that you can cast a numeric value to the enumeration type even if that numeric value is not defined in the enumeration.

Please see the section on enumerations in the *C# Programming II* presentation.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `Environment`

Copyright ©2011 sheepsqueezers.com

Environment

The `Environment` class provides information about, and means to manipulate, the current environment and platform. It is a static class. Microsoft's website states: *Use the Environment class to retrieve information such as command-line arguments, the exit code, environment variable settings, contents of the call stack, time since last system boot, and the version of the common language runtime.*

Properties (static)

- CommandLine - Gets the command line for this process.
- CurrentDirectory - Gets or sets the fully qualified path of the current working directory.
- ExitCode - Gets or sets the exit code of the process.
- HasShutdownStarted - Gets a value indicating whether the common language runtime (CLR) is shutting down.
- Is64BitOperatingSystem - Determines whether the current operating system is a 64-bit operating system.
- Is64BitProcess - Determines whether the current process is a 64-bit process.
- MachineName - Gets the NetBIOS name of this local computer.
- NewLine - Gets the newline string defined for this environment.
- OSVersion - Gets an OperatingSystem object that contains the current platform identifier and version number.
- ProcessorCount - Gets the number of processors on the current machine.
- StackTrace - Gets current stack trace information.
- SystemDirectory - Gets the fully qualified path of the system directory.
- SystemPageSize - Gets the amount of memory for an operating system's page file.
- TickCount - Gets the number of milliseconds elapsed since the system started.
- UserDomainName - Gets the network domain name associated with the current user.
- UserInteractive - Gets a value indicating whether the current process is running in user interactive mode.
- UserName - Gets the user name of the person who is currently logged on to the Windows operating system.
- Version - Gets a Version object that describes the major, minor, build, and revision numbers of the common language runtime.
- WorkingSet - Gets the amount of physical memory mapped to the process context.

Copyright ©2011 sheepsqueezers.com

Environment

Methods (static)

- Exit - Terminates this process and gives the underlying operating system the specified exit code.
- ExpandEnvironmentVariables - Replaces the name of each environment variable embedded in the specified string with the string equivalent of the value of the variable, then returns the resulting string.
- FailFast(String) - Immediately terminates a process after writing a message to the Windows Application event log, and then includes the message in error reporting to Microsoft.
- FailFast(String, Exception) - Immediately terminates a process after writing a message to the Windows Application event log, and then includes the message and exception information in error reporting to Microsoft.
- GetCommandLineArgs - Returns a string array containing the command-line arguments for the current process.
- GetEnvironmentVariable(String) - Retrieves the value of an environment variable from the current process.
- GetEnvironmentVariable(String, EnvironmentVariableTarget) - Retrieves the value of an environment variable from the current process or from the Windows operating system registry key for the current user or local machine.
- GetEnvironmentVariables() - Retrieves all environment variable names and their values from the current process.
- GetEnvironmentVariables(EnvironmentVariableTarget) - Retrieves all environment variable names and their values from the current process, or from the Windows operating system registry key for the current user or local machine.
- GetFolderPath(Environment.SpecialFolder) - Gets the path to the system special folder that is identified by the specified enumeration.
- GetFolderPath(Environment.SpecialFolder, Environment.SpecialFolderOption) - Gets the path to the system special folder that is identified by the specified enumeration, and uses a specified option for accessing special folders.
- GetLogicalDrives - Returns an array of string containing the names of the logical drives on the current computer.
- SetEnvironmentVariable(String, String) - Creates, modifies, or deletes an environment variable stored in the current process.
- SetEnvironmentVariable(String, String, EnvironmentVariableTarget) - Creates, modifies, or deletes an environment variable stored in the current process or in the Windows operating system registry key reserved for the current user or local machine.

As an example, below I display the number of processors on my machine as well as list the number of logical drives. See next slide for the code.

Copyright ©2011 sheepsqueezers.com

Environment


```
using System;

class MainProgram {

 public static void Main() {

  Console.WriteLine(Environment.ProcessorCount); // 2
  String[] sDrives = Environment.GetLogicalDrives();

  //Displays: List of Drives: C:\ D:\ E:\ F:\
  Console.Write("List of Drives: ");
  foreach(String sDrive in sDrives) {
   Console.Write("{0} ",sDrive);
  }
  Console.WriteLine();

 }

}
```

Note that you can use the method `GetCommandLineArgs()` to retrieve a string array of the parsed command lines. How this, as well as `CommandLine()`, differs from accessing the command line arguments through the `Main` parameters, I cannot say.

Be aware of the enumerations `Environment.SpecialFolder` and `Environment.SpecialFolderOption`.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Uri`

Copyright ©2011 sheepsqueezers.com

Uri

The `Uri` class provides an object representation of a uniform resource identifier (URI) and easy access to the parts of the URI. From Microsoft's website: *A URI is a compact representation of a resource available to your application on the intranet or Internet. The Uri class defines the properties and methods for handling URIs, including parsing, comparing, and combining. The Uri class properties are read-only; to create a modifiable object, use the UriBuilder class. Relative URIs (for example, "/new/index.htm") must be expanded with respect to a base URI so that they are absolute. The MakeRelative method is provided to convert absolute URIs to relative URIs when necessary.*

Constructors

- Uri – Initializes a new instance of the Uri class. There are 7 overloads to this constructor (2 of which are obsolete).

Properties

- AbsolutePath - Gets the absolute path of the URI.
- AbsoluteUri - Gets the absolute URI.
- Authority - Gets the Domain Name System (DNS) host name or IP address and the port number for a server.
- DnsSafeHost - Gets an unescaped host name that is safe to use for DNS resolution.
- Fragment - Gets the escaped URI fragment.
- Host - Gets the host component of this instance.
- HostNameType - Gets the type of the host name specified in the URI.
- IsAbsoluteUri - Gets whether the Uri instance is absolute.
- IsDefaultPort - Gets whether the port value of the URI is the default for this scheme.
- IsFile - Gets a value indicating whether the specified Uri is a file URI.
- IsLoopback - Gets whether the specified Uri references the local host.
- IsUnc - Gets whether the specified Uri is a universal naming convention (UNC) path.
- LocalPath - Gets a local operating-system representation of a file name.
- OriginalString - Gets the original URI string that was passed to the Uri constructor.

Copyright ©2011 sheepsqueezers.com

Properties (continued)

- PathAndQuery - Gets the AbsolutePath and Query properties separated by a question mark (?).
- Port - Gets the port number of this URI.
- Query - Gets any query information included in the specified URI.
- Scheme - Gets the scheme name for this URI.
- Segments - Gets an array containing the path segments that make up the specified URI.
- UserEscaped - Indicates that the URI string was completely escaped before the Uri instance was created.
- UserInfo - Gets the user name, password, or other user-specific information associated with the specified URI.

Methods

- Canonicalize - Infrastructure. Obsolete. Converts the internally stored URI to canonical form.
- CheckHostName - Determines whether the specified host name is a valid DNS name.
- CheckSchemeName - Determines whether the specified scheme name is valid.
- CheckSecurity - Infrastructure. Obsolete. Calling this method has no effect.
- Compare - Compares the specified parts of two URIs using the specified comparison rules.
- Equals - Compares two Uri instances for equality. (Overrides Object.Equals(Object).)
- Escape - Infrastructure. Obsolete. Converts any unsafe or reserved characters in the path component to their hexadecimal character representations.
- EscapeDataString - Converts a string to its escaped representation.
- EscapeString - Obsolete. Converts a string to its escaped representation.
- EscapeUriString - Converts a URI string to its escaped representation.
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- FromHex - Gets the decimal value of a hexadecimal digit.
- GetComponents - Gets the specified components of the current instance using the specified escaping for special characters.
- GetHashCode - Gets the hash code for the URI. (Overrides Object.GetHashCode().)
- GetLeftPart - Gets the specified portion of a Uri instance.
- GetObjectData - Returns the data needed to serialize the current instance.
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- HexEscape - Converts a specified character into its hexadecimal equivalent.
- HexUnescape - Converts a specified hexadecimal representation of a character to the character.

Copyright ©2011 sheepsqueezers.com

Methods (continued)

- IsBadFileSystemCharacter - Infrastructure. Obsolete. Gets whether a character is invalid in a file system name.
- IsBaseOf - Determines whether the current Uri instance is a base of the specified Uri instance.
- IsExcludedCharacter - Infrastructure. Obsolete. Gets whether the specified character should be escaped.
- IsHexDigit - Determines whether a specified character is a valid hexadecimal digit.
- IsHexEncoding - Determines whether a character in a string is hexadecimal encoded.
- IsReservedCharacter - Infrastructure. Obsolete. Gets whether the specified character is a reserved character.
- IsWellFormedOriginalString - Indicates whether the string used to construct this Uri was well-formed and is not required to be further escaped.
- IsWellFormedUriString - Indicates whether the string is well-formed by attempting to construct a URI with the string and ensures that the string does not require further escaping.
- MakeRelative - Obsolete. Determines the difference between two Uri instances.
- MakeRelativeUri - Determines the difference between two Uri instances.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- Parse - Infrastructure. Obsolete. Parses the URI of the current instance to ensure it contains all the parts required for a valid URI.
- ToString - Gets a canonical string representation for the specified Uri instance. (Overrides Object.ToString().)
- TryCreate(String, UriKind, Uri) - Creates a new Uri using the specified String instance and a UriKind.
- TryCreate(Uri, String, Uri) - Creates a new Uri using the specified base and relative String instances.
- TryCreate(Uri, Uri, Uri) - Creates a new Uri using the specified base and relative Uri instances.
- Unescape - Infrastructure. Obsolete. Converts the specified string by replacing any escape sequences with their unescaped representation.
- UnescapeDataString - Converts a string to its unescaped representation.

Operators

- Equality - Determines whether two Uri instances have the same value.
- Inequality - Determines whether two Uri instances do not have the same value.

Fields

- SchemeDelimiter - Specifies the characters that separate the communication protocol scheme from the address portion of the URI. This field is read-only.
- UriSchemeFile - Specifies that the URI is a pointer to a file. This field is read-only.

Copyright ©2011 sheepsqueezers.com

Fields (continued)

- UriSchemeFtp - Specifies that the URI is accessed through the File Transfer Protocol (FTP). This field is read-only.
- UriSchemeGopher - Specifies that the URI is accessed through the Gopher protocol. This field is read-only.
- UriSchemeHttp - Specifies that the URI is accessed through the Hypertext Transfer Protocol (HTTP). This field is read-only.
- UriSchemeHttps - Specifies that the URI is accessed through the Secure Hypertext Transfer Protocol (HTTPS). This field is read-only.
- UriSchemeMailto - Specifies that the URI is an e-mail address and is accessed through the Simple Mail Transport Protocol (SMTP). This field is read-only.
- UriSchemeNetPipe - Specifies that the URI is accessed through the NetPipe scheme used by Windows Communication Foundation (WCF). This field is read-only.
- UriSchemeNetTcp - Specifies that the URI is accessed through the NetTcp scheme used by Windows Communication Foundation (WCF). This field is read-only.
- UriSchemeNews - Specifies that the URI is an Internet news group and is accessed through the Network News Transport Protocol (NNTP). This field is read-only.
- UriSchemeNntp - Specifies that the URI is an Internet news group and is accessed through the Network News Transport Protocol (NNTP). This field is read-only.

Here is an example:

```
using System;

class MainProgram {

 public static void Main() {

  Uri uriSPAM = new Uri("http://www.youtube.com/watch?v=anwy2MPT5RE");
  Console.WriteLine("AbsolutePath={0}",uriSPAM.AbsolutePath);
  Console.WriteLine("AbsoluteUri={0}",uriSPAM.AbsoluteUri);
  Console.WriteLine("Authority={0}",uriSPAM.Authority);
  Console.WriteLine("Host={0}",uriSPAM.Host);
  Console.WriteLine("PathAndQuery={0}",uriSPAM.PathAndQuery);
```

…continued on next slide…

Copyright ©2011 sheepsqueezers.com

Uri


```
Console.WriteLine("Port={0}",uriSPAM.Port);
  Console.WriteLine("Query={0}",uriSPAM.Query);
  Console.WriteLine("Scheme={0}",uriSPAM.Scheme);

  String[] sSegments = uriSPAM.Segments;
  foreach(String sSegment in sSegments) {
   Console.WriteLine("Segement={0}",sSegment);
  }

 }

}
```

Here are the results:

```
AbsolutePath=/watch
AbsoluteUri=http://www.youtube.com/watch?v=anwy2MPT5RE
Authority=www.youtube.com
Host=www.youtube.com
PathAndQuery=/watch?v=anwy2MPT5RE
Port=80
Query=?v=anwy2MPT5RE
Scheme=http
Segement=/
Segement=watch
```

Now, be aware of the `UriPartial` enumeration for use with the `GetLeftPart()` method:

- Scheme - The scheme segment of the URI.
- Authority - The scheme and authority segments of the URI.
- Path - The scheme, authority, and path segments of the URI.
- Query - The scheme, authority, path, and query segments of the URI.

Copyright ©2011 sheepsqueezers.com

Here is an example using the `GetLeftPart()` method:

```
using System;

class MainProgram {

 public static void Main() {

  Uri uriSPAM = new Uri("http://www.youtube.com/watch?v=anwy2MPT5RE");
  Console.WriteLine("Scheme={0}",uriSPAM.GetLeftPart(UriPartial.Scheme));
  Console.WriteLine("Authority={0}",uriSPAM.GetLeftPart(UriPartial.Authority));
  Console.WriteLine("Path={0}",uriSPAM.GetLeftPart(UriPartial.Path));
  Console.WriteLine("Query={0}",uriSPAM.GetLeftPart(UriPartial.Query));

 }

}
```

Here are the results:

```
Scheme=http://
Authority=http://www.youtube.com
Path=http://www.youtube.com/watch
Query=http://www.youtube.com/watch?v=anwy2MPT5RE
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `UriBuilder`

Copyright ©2011 sheepsqueezers.com

UriBuilder

The `UriBuilder` class provides a custom constructor for uniform resource identifiers (URIs) and modifies URIs for the Uri class. From Microsoft's website: *The UriBuilder class provides a convenient way to modify the contents of a Uri instance without creating a new Uri instance for each modification. The UriBuilder properties provide read/write access to the read-only Uri properties so that they can be modified.*

See the Microsoft website for more on this class.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `UriParser` (as well as the following classes derived from UriParser: `FileStyleUriParser`, `FtpStyleUriParser`, `GenericUriParser`, `GopherStyleUriParser`, `HttpStyleUriParser`, `LdapStyleUriParser`, `NetPipeStyleUriParser`, `NetTcpStyleUriParser`, `NewsStyleUriParser`)

Copyright ©2011 sheepsqueezers.com

UriParser

The `UriParser` class parses a new URI scheme. Microsoft's website states: *The UriParser class enables you to create parsers for new URI schemes. You can write these parsers in their entirety, or the parsers can be derived from well-known schemes (HTTP, FTP, and other schemes based on network protocols). If you want to create a completely new parser, inherit from GenericUriParser. If you want to create a parser that extends a well-known URI scheme, inherit from FtpStyleUriParser, HttpStyleUriParser, FileStyleUriParser, GopherStyleUriParser, or LdapStyleUriParser. Microsoft strongly recommends that you use a parser shipped with the .NET Framework. Building your own parser increases the complexity of your application, and will not perform as well as the shipped parsers.*

Constructors

- UriParser – Constructs a default URI parser.

Methods

- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- GetComponents - Gets the components from a URI.
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- InitializeAndValidate - Initialize the state of the parser and validate the URI.
- IsBaseOf - Determines whether baseUri is a base URI for relativeUri.
- IsKnownScheme - Indicates whether the parser for a scheme is registered.
- IsWellFormedOriginalString - Indicates whether a URI is well-formed.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- OnNewUri - Invoked by a Uri constructor to get a UriParser instance
- OnRegister - Invoked by the Framework when a UriParser method is registered.
- Register - Associates a scheme and port number with a UriParser.
- Resolve - Called by Uri constructors and Uri.TryCreate() to resolve a relative URI.
- ToString - Returns a string that represents the current object. (Inherited from Object.)

Copyright ©2011 sheepsqueezers.com

UriParser

See the `UriParser.Register()` method for how to register your new Uri parser.

See Microsoft's website for more on this class.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `UriTemplate, UriTemplateEquivalenceComparer, UriTemplateMatch, UriTemplateTable`

Copyright ©2011 sheepsqueezers.com

UriTemplate*

The `UriTemplate` class represents a Uniform Resource Identifier template. From Microsoft's website: *A URI template allows you to define a set of structurally similar URIs. Templates are composed of two parts, a path and a query. A path consists of a series of segments delimited by a slash (/). Each segment can have a literal value, a variable value (written within curly braces [{ }], constrained to match the contents of exactly one segment), or a wildcard (written as an asterisk [*], which matches "the rest of the path"), which must appear at the end of the path. The query expression can be omitted entirely. If present, it specifies an unordered series of name/value pairs. Elements of the query expression can be either literal pairs (?x=2) or variable pairs (?x={val}). Unpaired values are not permitted. The following examples show valid template strings:*

- `"weather/WA/Seattle"`
- `"weather/{state}/{city}"`
- `"weather/*"`
- `"weather/{state}/{city}?forecast=today`
- `"weather/{state}/{city}?forecast={day}`

The preceding URI templates might be used for organizing weather reports. Segments enclosed in curly braces are variables, everything else is a literal. You can convert a UriTemplate instance into a Uri by replacing variables with actual values. For example, taking the template "weather/{state}/{city}" and putting in values for the variables "{state}" and "{city}" gives you "weather/WA/Seattle". Given a candidate URI, you can test whether it matches a given URI template by calling Match(Uri, Uri). You can also use UriTemplate instances to create a Uri from a set of variable values by calling BindByName(Uri, NameValueCollection) or BindByPosition(Uri, String[]).

Copyright ©2011 sheepsqueezers.com

UriTemplate*

Constructors

- UriTemplateString – Initializes a new instance of the UriTemplate class. There are 4 overloads to this method.

Properties

- Defaults - Gets a collection of name/value pairs for any default parameter values.
- IgnoreTrailingSlash - Specifies whether trailing slashes "/" in the template should be ignored when matching candidate URIs.
- PathSegmentVariableNames - Gets a collection of variable names used within path segments in the template.
- QueryValueVariableNames - Gets a collection of variable names used within the query string in the template.

Methods

- BindByName(Uri, IDictionary<String, String>) - Creates a new URI from the template and the collection of parameters.
- BindByName(Uri, NameValueCollection) - Creates a new URI from the template and the collection of parameters.
- BindByName(Uri, IDictionary<String, String>, Boolean) - Creates a new URI from the template and the collection of parameters.
- BindByName(Uri, NameValueCollection, Boolean) - Creates a new URI from the template and the collection of parameters.
- BindByPosition - Creates a new URI from the template and an array of parameter values.
- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- IsEquivalentTo - Indicates whether a UriTemplate is structurally equivalent to another.
- Match - Attempts to match a URI to a UriTemplate.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- ToString - Returns a string representation of the UriTemplate instance. (Overrides Object.ToString().)

The UriTemplate class and its features seem like a great way to create URIs based on a template a bunch of data. Below is an example.

Copyright ©2011 sheepsqueezers.com

UriTemplate*


```
using System;
using System.Collections.Generic;

class MainProgram {

 public static void Main() {

  Uri uWebsite = new Uri("http://www.amazon.com"); //Initial the website name as a Uri
  UriTemplate utBookTemplate = new UriTemplate("{book}/dp/{isbn}"); //Create the template as a UriTemplate

  //Build a dictionary of book names and ISBN numbers.  Presumably, you would pull these from a database
  //and place them in a Dataset.
  Dictionary<String,String> dBookISBN = new Dictionary<String,String>();
  dBookISBN.Add("Go-F-Sleep-Adam-Mansbach","1617750255");
  dBookISBN.Add("Heaven-Real-Little-Astounding-Story","0849946158");
  dBookISBN.Add("Prescription-Excellence-Leadership-Creating-Experience","0071773541");
  dBookISBN.Add("Garden-Beasts-Terror-American-Hitlers","0307408841");
  dBookISBN.Add("George-Martins-Thrones-4-Book-Boxed","0345529057");

  //Create a Dictionary for use with the template
  Dictionary<String,String> dThisBook = new Dictionary<String,String>();

  //Loop around for each entry in the database.
  foreach(KeyValuePair<String,String> dKVP in dBookISBN) {

   //Add this book from the database into the dictionary.
   dThisBook.Add("book",dKVP.Key);
   dThisBook.Add("isbn",dKVP.Value);

   //Create the final Uri
   Uri uThisBook = utBookTemplate.BindByName(uWebsite,dThisBook);

   …continued on next slide…
```

Copyright ©2011 sheepsqueezers.com


```
    //Write out this iteration`s book.
    Console.WriteLine(uThisBook.ToString());

    //Clear the dictionary.
    dThisBook.Clear();

   }

  }

 }
```

The output looks like this:

```
http://www.amazon.com/Go-F-Sleep-Adam-Mansbach/dp/1617750255
http://www.amazon.com/Heaven-Real-Little-Astounding-Story/dp/0849946158
http://www.amazon.com/Prescription-Excellence-Leadership-Creating-Experience/dp/0071773541
http://www.amazon.com/Garden-Beasts-Terror-American-Hitlers/dp/0307408841
http://www.amazon.com/George-Martins-Thrones-4-Book-Boxed/dp/0345529057
```

The `UriTemplateEquivalenceComparer` class (and, yes, that's a long name for a class) is used to compare `UriTemplate` instances for structural (instead of reference) equivalence. See Microsoft's website for more on this class.

The `UriTemplateMatch` class represents the results of a match operation on a `UriTemplate` instance when using the `UriTemplate`'s `Match(Uri,Uri)` method. See Microsoft's website for more on this class.

Copyright ©2011 sheepsqueezers.com

UriTemplate*

The `UriTemplateTable` class represents an associative set of `UriTemplate` objects. This allows you to add several `UriTemplate` classes into a collection and then determine which template matched against an incoming URI.

You can think of this as similar to using regular expressions to determine which RegEx matched against an incoming U.S. address. Once you know which address matched, you can then parse that data appropriately.

See Microsoft's website for more on this class.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `GC`

Copyright ©2011 sheepsqueezers.com

GC

The GC class controls the system garbage collector, a service that automatically reclaims unused memory. GC is a static class. Microsoft's website states: *The methods in this class influence when garbage collection is performed on an object and when resources allocated by an object are released. Properties in this class provide information about the total amount of memory available in the system and the age category, or generation, of memory allocated to an object.*

The garbage collector tracks and reclaims objects allocated in managed memory. Periodically, the garbage collector performs garbage collection to reclaim memory allocated to objects for which there are no valid references. Garbage collection happens automatically when a request for memory cannot be satisfied using available free memory. Alternatively, an application can force garbage collection using the Collect method.

Garbage collection consists of the following steps:

1. *The garbage collector searches for managed objects that are referenced in managed code.*
2. *The garbage collector tries to finalize objects that are not referenced.*
3. *The garbage collector frees objects that are not referenced and reclaims their memory.*

During a collection, the garbage collector will not free an object if it finds one or more references to the object in managed code. However, the garbage collector does not recognize references to an object from unmanaged code, and might free objects that are being used exclusively in unmanaged code unless explicitly prevented from doing so.

Copyright ©2011 sheepsqueezers.com

The KeepAlive method provides a mechanism that prevents the garbage collector from collecting objects that are still in use in unmanaged code.

Aside from managed memory allocations, implementations of the garbage collector do not maintain information about resources held by an object, such as file handles or database connections. When a type uses unmanaged resources that must be released before instances of the type are reclaimed, the type can implement a finalizer.

In most cases, finalizers are implemented by overriding the Object.Finalize method; however, types written in C# or C++ implement destructors, which compilers turn into an override of Object.Finalize. In most cases, if an object has a finalizer, the garbage collector calls it prior to freeing the object. However, the garbage collector is not required to call finalizers in all situations; for example, the SuppressFinalize method explicitly prevents a finalizer from being called. Also, the garbage collector is not required to use a specific thread to finalize objects, or guarantee the order in which finalizers are called for objects that reference each other but are otherwise available for garbage collection.

In scenarios where resources must be released at a specific time, classes can implement the IDisposable interface, which contains the IDisposable.Dispose method that performs resource management and cleanup tasks. Classes that implement Dispose must specify, as part of their class contract, if and when class consumers call the method to clean up the object. The garbage collector does not, by default, call the Dispose method; however, implementations of the Dispose method can call methods in the GC class to customize the finalization behavior of the garbage collector.

Copyright ©2011 sheepsqueezers.com

sheepsqueezers.com

It is recommended, but not required, that garbage collectors support object aging using generations. A generation is a unit of measure of the relative age of objects in memory. The generation number, or age, of an object indicates the generation to which an object belongs. Objects created more recently are part of newer generations, and have lower generation numbers than objects created earlier in the application life cycle. Objects in the most recent generation are in generation zero.
Notes to Implementers This implementation of the garbage collector supports three generations of objects.
MaxGeneration is used to determine the maximum generation number supported by the system. Object aging allows applications to target garbage collection at a specific set of generations rather than requiring the garbage collector to evaluate all generations.

Properties (static)

- MaxGeneration - Gets the maximum number of generations that the system currently supports.

Methods (static)

- AddMemoryPressure - Informs the runtime of a large allocation of unmanaged memory that should be taken into account when scheduling garbage collection.
- CancelFullGCNotification - Cancels the registration of a garbage collection notification.
- Collect() - Forces an immediate garbage collection of all generations.
- Collect(Int32) - Forces an immediate garbage collection from generation zero through a specified generation.
- Collect(Int32, GCCollectionMode) - Forces a garbage collection from generation zero through a specified generation, at a time specified by a GCCollectionMode value.
- CollectionCount - Returns the number of times garbage collection has occurred for the specified generation of objects.
- GetGeneration(Object) - Returns the current generation number of the specified object.
- GetGeneration(WeakReference) - Returns the current generation number of the target of a specified weak reference.
- GetTotalMemory - Retrieves the number of bytes currently thought to be allocated. A parameter indicates whether this method can wait a short interval before returning, to allow the system to collect garbage and finalize objects.

Copyright ©2011 sheepsqueezers.com

Methods (static)

- KeepAlive - References the specified object, which makes it ineligible for garbage collection from the start of the current routine to the point where this method is called.
- RegisterForFullGCNotification - Specifies that a garbage collection notification should be raised when conditions favor full garbage collection and when the collection has been completed.
- RemoveMemoryPressure - Informs the runtime that unmanaged memory has been released and no longer needs to be taken into account when scheduling garbage collection.
- ReRegisterForFinalize - Requests that the system call the finalizer for the specified object for which SuppressFinalize has previously been called.
- SuppressFinalize - Requests that the system not call the finalizer for the specified object.
- WaitForFullGCApproach() - Returns the status of a registered notification for determining whether a full garbage collection by the common langauge runtime is imminent.
- WaitForFullGCApproach(Int32) - Returns, in a specified time-out period, the status of a registered notification for determining whether a full garbage collection by the common language runtime is imminent.
- WaitForFullGCComplete() - Returns the status of a registered notification for determining whether a full garbage collection by the common language runtime has completed.
- WaitForFullGCComplete(Int32) - Returns, in a specified time-out period, the status of a registered notification for determining whether a full garbage collection by common language the runtime has completed.
- WaitForPendingFinalizers - Suspends the current thread until the thread that is processing the queue of finalizers has emptied that queue.

See the *C# Programming III* presentation for more about the Garbage Collector.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Lazy`

Copyright ©2011 sheepsqueezers.com

Lazy

The `Lazy(Of T)` class supports lazy initialization. Microsoft's website states: *Lazy initialization occurs the first time the Lazy<T>.Value property is accessed or the Lazy<T>.ToString method is called. Use an instance of Lazy<T> to defer the creation of a large or resource-intensive object or the execution of a resource-intensive task, particularly when such creation or execution might not occur during the lifetime of the program.*

The `Lazy(Of T, TMetadata)` class provides a lazy indirect reference to an object and its associated metadata for use by the Managed Extensibility Framework.

See the next slide for a cheap example.

Also, see Microsoft's website for more on these classes.

Also, see the following very nice article for more on *Lazy Initialization*: http://msdn.microsoft.com/en-us/library/dd997286.aspx.

Copyright ©2011 sheepsqueezers.com

Lazy


```
using System;

class MyClass {

 private Int32 iNumber;

 public MyClass() {
  iNumber=42;
 }

 public Int32 Number{

  set {
   iNumber=value;
  }

  get {
   return(iNumber);
  }

 }

}

class MainProgram {

 public static void Main() {

  Boolean bTF = true;
  Lazy<MyClass> lMyClass = new Lazy<MyClass>();

  if (bTF) {
   MyClass oMC = lMyClass.Value;
   Console.WriteLine(oMC.Number.ToString());
  }
  else {
   Console.WriteLine("Did not lazy create stuff!");
  }

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `LocalDataStoreSlot`

Copyright ©2011 sheepsqueezers.com

LocalDataStoreSlot

The `LocalDataStoreSlot` class encapsulates a memory slot to store local data. Microsoft's website states: *The .NET Framework provides two mechanisms for using thread local storage (TLS): thread-relative static fields and data slots.*

Thread-relative static fields are static fields (Shared fields in Visual Basic) that are marked with the ThreadStaticAttribute attribute. They provide better performance than data slots, and enable compile-time type checking.

Data slots are slower and more awkward to use than thread-relative static fields. Also, data is stored as type Object, so you must cast it to the correct type before using it. However, you can use data slots when you have insufficient information at compile time to allocate static fields.

For more information about using TLS, see Thread Local Storage: Thread-Relative Static Fields and Data Slots.

Similarly, the .NET Framework provides two mechanisms for using context local storage: context-relative static fields and data slots. Context-relative static fields are static fields that are marked with the ContextStaticAttribute attribute. The trade-offs between using these two mechanisms are similar to the tradeoffs between using thread-relative static fields and data slots.

Copyright ©2011 sheepsqueezers.com

LocalDataStoreSlot

The LocalDataStoreSlot structure serves as a local store memory mechanism that threads and contexts can use to store thread-specific and context-specific data, respectively. The common language runtime allocates a multi-slot data store array to each process when it is created. The thread or context calls various functions to allocate a data slot in the data store, to store and retrieve a data value in the slot, and to free a data slot for reuse after the thread or context object expires.

The data slots are unique per thread or context; their values are not shared between the thread or context objects. Data slots can be allocated by a name or by an index number.

For more information about storing local data, see Thread or Context. The LocalDataStoreSlot class is used with methods such as Thread.AllocateNamedDataSlot, Context.AllocateNamedDataSlot, Thread.GetData, and Context.GetData; it does not have any methods of its own that you need to use.

See the following article for more on *Thread Local Storage*: http://msdn.microsoft.com/en-us/library/6sby1byh.aspx.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `MarshalByRefObject`

Copyright ©2011 sheepsqueezers.com

MarshalByRefObject

The `MarshalByRefObject` class enables access to objects across application domain boundaries in applications that support remoting. Microsoft's website states: *An application domain is a partition in an operating system process where one or more applications reside. Objects in the same application domain communicate directly. Objects in different application domains communicate either by transporting copies of objects across application domain boundaries, or by using a proxy to exchange messages.*

MarshalByRefObject is the base class for objects that communicate across application domain boundaries by exchanging messages using a proxy. Objects that do not inherit from MarshalByRefObject are implicitly marshal by value. When a remote application references a marshal by value object, a copy of the object is passed across application domain boundaries.

MarshalByRefObject objects are accessed directly within the boundaries of the local application domain. The first time an application in a remote application domain accesses a MarshalByRefObject, a proxy is passed to the remote application. Subsequent calls on the proxy are marshaled back to the object residing in the local application domain.

Types must inherit from MarshalByRefObject when the type is used across application domain boundaries, and the state of the object must not be copied because the members of the object are not usable outside the application domain where they were created.

Copyright ©2011 sheepsqueezers.com

MarshalByRefObject

Constructors

- MarshalByRefObject - Initializes a new instance of the MarshalByRefObject class.

Methods

- CreateObjRef - Creates an object that contains all the relevant information required to generate a proxy used to communicate with a remote object.
- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetLifetimeService - Retrieves the current lifetime service object that controls the lifetime policy for this instance.
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- InitializeLifetimeService - Obtains a lifetime service object to control the lifetime policy for this instance.
- MemberwiseClone() - Creates a shallow copy of the current Object. (Inherited from Object.)
- MemberwiseClone(Boolean) - Creates a shallow copy of the current MarshalByRefObject object.
- ToString - Returns a string that represents the current object. (Inherited from Object.)

I thought I read somewhere that `ContextBoundObject` and `MarshalByRefObject` are complements to each other…I'll have to check on this!!

See the example of Microsoft's website at http://msdn.microsoft.com/en-us/library/system.marshalbyrefobject.aspx.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Math`

Copyright ©2011 sheepsqueezers.com

Math

The `Math` class provides constants and static methods for trigonometric, logarithmic, and other common mathematical functions. The `Math` class is completely static.

Fields (static)

- E - Represents the natural logarithmic base, specified by the constant, e. Returns a Double.
- PI - Represents the ratio of the circumference of a circle to its diameter, specified by the constant, p. Returns a Double.

Methods (static)

- Abs - Returns the absolute value of the numeric parameter. There are 7 overloads to this method.
- Acos - Returns the angle whose cosine is the specified number.
- Asin - Returns the angle whose sine is the specified number.
- Atan - Returns the angle whose tangent is the specified number.
- Atan2 - Returns the angle whose tangent is the quotient of two specified numbers.
- BigMul - Produces the full product of two 32-bit numbers.
- Ceiling - Returns the smallest integral value that is greater than or equal to the specified number. There are 2 overloads to this method.
- Cos - Returns the cosine of the specified angle.
- Cosh - Returns the hyperbolic cosine of the specified angle.
- DivRem - Calculates the quotient of two 32-bit or 64-bit signed integers and also returns the remainder in an output parameter. There are 2 overloads to this method.
- Exp - Returns e raised to the specified power.
- Floor - Returns the largest integer less than or equal to the specified decimal number.
- IEEERemainder Returns the remainder resulting from the division of a specified number by another specified number.
- Log - Returns the natural (base e or a chosen base) logarithm of a specified number. There are 2 overloads to this method.
- Log10 - Returns the base 10 logarithm of a specified number.
- Max - Returns the larger of the two numeric parameters. There are 11 overloads to this method.
- Min - Returns the smaller of the two numeric parameters. There are 11 overloads to this method.
- Pow - Returns a specified number raised to the specified power.
- Round - Rounds a numeric value to the specified number of fractional digits. A parameter specifies how to round the value if it is midway between two other numbers. There are 8 overloads to this method.
- Sign - Returns a value indicating the sign of a decimal number.
- Sin - Returns the sine of the specified angle. There are 7 overloads to this method.
- Sinh - Returns the hyperbolic sine of the specified angle.
- Sqrt - Returns the square root of a specified number.
- Tan - Returns the tangent of the specified angle.
- Tanh - Returns the hyperbolic tangent of the specified angle.
- Truncate - Calculates the integral part of a specified decimal or double number. There are 2 overloads to this method.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `MulticastDelegate`

Copyright ©2011 sheepsqueezers.com

MulticastDelegate

The `MulticastDelegate` class represents a multicast delegate; that is, a delegate that can have more than one element in its invocation list. According to Microsoft's website: *MulticastDelegate is a special class. Compilers and other tools can derive from this class, but you cannot derive from it explicitly. The same is true of the Delegate class. In addition to the methods that delegate types inherit from MulticastDelegate, the common language runtime provides two special methods: BeginInvoke and EndInvoke. For more information about these methods, see Calling Synchronous Methods Asynchronously. A MulticastDelegate has a linked list of delegates, called an invocation list, consisting of one or more elements. When a multicast delegate is invoked, the delegates in the invocation list are called synchronously in the order in which they appear. If an error occurs during execution of the list then an exception is thrown.*

See more at http://msdn.microsoft.com/en-us/library/system.multicastdelegate.aspx.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Nullable`

Copyright ©2011 sheepsqueezers.com

Nullable

The `Nullable` class supports a **value type** that can be assigned null like a reference type. This is a static class. According to Microsoft's website: *A type is said to be nullable if it can be assigned a value or can be assigned null, which means the type has no value whatsoever. Consequently, a nullable type can express a value, or that no value exists. For example, a reference type such as String is nullable, whereas a value type such as Int32 is not. A value type cannot be nullable because it has enough capacity to express only the values appropriate for that type; it does not have the additional capacity required to express a value of null. The Nullable class provides complementary support for the Nullable<T> structure. The Nullable class supports obtaining the underlying type of a nullable type, and comparison and equality operations on pairs of nullable types whose underlying value type does not support generic comparison and equality operations.*

Scenario

Use nullable types to represent things that exist or do not exist depending on the circumstance. For example, an optional attribute of an HTML tag might exist in one tag but not another, or a nullable column of a database table might exist in one row of the table but not another. You can represent the attribute or column as a field in a class and you can define the field as a value type. The field can contain all the valid values for the attribute or column, but cannot accommodate an additional value that means the attribute or column does not exist. In this case, define the field to be a nullable type instead of a value type.

Copyright ©2011 sheepsqueezers.com

Nullable

Boxing and Unboxing

When a nullable type is boxed, the common language runtime automatically boxes the underlying value of the Nullable<T> object, not the Nullable<T> object itself. That is, if the HasValue property is true, the contents of the Value property is boxed. If the HasValue property is false, null is boxed. When the underlying value of a nullable type is unboxed, the common language runtime creates a new Nullable<T> structure initialized to the underlying value.

Methods (static)

- Compare<T> - Compares the relative values of two Nullable<T> objects.
- Equals<T> - Indicates whether two specified Nullable<T> objects are equal.
- GetUnderlyingType - Returns the underlying type argument of the specified nullable type.

The `Nullable<T>` structure represents an object whose underlying type is a value type that can also be assigned null like a reference type. Despite the structure, you can use a question mark (?) after the value type to indicate that the type is a nullable type. Both iValue1 and iValue2 are `Nullable<Int32>` objects below.

```
class MainProgram {

 public static void Main() {

  Int32? iValue1 = 10; //Question Mark indicates a Nullable<T> structure.
  Int32? iValue2 = null; //Question Mark indicates a Nullable<T> structure.

  //Use the ?? syntax to print out a -1 if iValue1 or iValue2 is null.
  Console.WriteLine(iValue1 ?? -1); //Prints 10
  Console.WriteLine(iValue2 ?? -1); //Prints -1

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `OperatingSystem`

Copyright ©2011 sheepsqueezers.com

OperatingSystem

The `OperatingSystem` class represents information about an operating system such as the version and platform identifier.

Constructors

- OperatingSystem(PlatformID,Version) - Initializes a new instance of the OperatingSystem class, using the specified platform identifier value and version object. PlatformID is an enumeration containing the following members: Win32S, Win32Windows, Win32NT, WinCE, Unix, Xbox, MacOSX. Version is a class itself and is described in its own section below.

Properties

- Platform - Gets a System.PlatformID enumeration value that identifies the operating system platform.
- ServicePack - Gets the service pack version represented by this OperatingSystem object.
- Version - Gets a System.Version object that identifies the operating system.
- VersionString - Gets the concatenated string representation of the platform identifier, version, and service pack that are currently installed on the operating system.

Methods

- Clone - Creates an OperatingSystem object that is identical to this instance.
- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetObjectData - Populates a System.Runtime.Serialization.SerializationInfo object with the data necessary to deserialize this instance.
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- ToString - Converts the value of this OperatingSystem object to its equivalent string representation. (Overrides Object.ToString().)

An example follows on the next slide. Note that this class is probably not used to determine the operating system information for the machine on which your program is running. I believe you should use Environment instead for that information.

Copyright ©2011 sheepsqueezers.com

OperatingSystem


```
using System;

class MainProgram {

 public static void Main() {

  PlatformID ePID = PlatformID.Win32S;
  Version oVER = new Version(1,2,3,4); //Build Version: 1.2.3.4
  OperatingSystem oOS = new OperatingSystem(ePID,oVER);
  Console.WriteLine("Platform={0}",oOS.Platform); //Platform=Win32S
  Console.WriteLine("ServicePack={0}",oOS.ServicePack); //ServicePack=blank
  Console.WriteLine("Version={0}",oOS.Version); //Version=1.2.3.4
  Console.WriteLine("VersionString={0}",oOS.VersionString); //VersionString=Microsoft Win32S 1.2.3.4

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `Random`

Copyright ©2011 sheepsqueezers.com

Random

The `Random` class represents a pseudo-random number generator, a device that produces a sequence of numbers that meet certain statistical requirements for randomness. According to Microsoft's website: *The random number generation starts from a seed value. If the same seed is used repeatedly, the same series of numbers is generated. One way to produce different sequences is to make the seed value time-dependent, thereby producing a different series with each new instance of Random. By default, the parameterless constructor of the Random class uses the system clock to generate its seed value, while its parameterized constructor can take an Int32 value based on the number of ticks in the current time. However, because the clock has finite resolution, using the parameterless constructor to create different Random objects in close succession creates random number generators that produce identical sequences of random numbers. The following example illustrates that two Random objects that are instantiated in close succession generate an identical series of random numbers.*

Constructors

- Random() - Initializes a new instance of the Random class, using a time-dependent default seed value.
- Random(Int32) - Initializes a new instance of the Random class, using the specified seed value.

Methods

- Next() - Returns a nonnegative random number.
- Next(Int32) - Returns a nonnegative random number less than the specified maximum.
- Next(Int32, Int32) - Returns a random number within a specified range.
- NextBytes - Fills the elements of a specified array of bytes with random numbers.
- NextDouble - Returns a random number between 0.0 and 1.0.
- Sample - Returns a random number between 0.0 and 1.0. Protected method!!

Copyright ©2011 sheepsqueezers.com

An example follows:

```
using System;

class MainProgram {

 public static void Main() {

  Random oRND = new Random(); //use system clock
  for(Int32 indx=0;indx<10;indx++) {
   Console.WriteLine(oRND.NextDouble().ToString());
  }

 }

}
```

This program produces the following values:

```
0.00780280586695429
0.779589653378161
0.568998454869258
0.208842755392586
0.702663717187319
0.377111448150646
0.325864296092589
0.48730795760048
0.588847587159298
0.82416316998385
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `String`

Copyright ©2011 sheepsqueezers.com

String

The `String` class represents text as a series of Unicode characters. Note that this class is sealed, so you cannot inherit from it, and implements several interfaces. Here is the definition of the `String` class:

```
public sealed class String : IComparable, ICloneable, IConvertible, IComparable<string>,
                             IEnumerable<char>, IEnumerable, IEquatable<string>
```

Note that strings are immutable; that is, read-only. See the `StringBuilder` class.

Constructors

- String(Char*) - Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters.
- String(Char[]) - Initializes a new instance of the String class to the value indicated by an array of Unicode characters.
- String(SByte*) - Initializes a new instance of the String class to the value indicated by a pointer to an array of 8-bit signed integers.
- String(Char, Int32) - Initializes a new instance of the String class to the value indicated by a specified Unicode character repeated a specified number of times.
- String(Char*, Int32, Int32) - Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters, a starting character position within that array, and a length.
- String(Char[], Int32, Int32) - Initializes a new instance of the String class to the value indicated by an array of Unicode characters, a starting character position within that array, and a length.
- String(SByte*, Int32, Int32) - Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting character position within that array, and a length.
- String(SByte*, Int32, Int32, Encoding) - Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting character position within that array, a length, and an Encoding object.

Note that you do not need to use a constructor to create a string, but you can use the equal-sign: `String sText = "Where is that damn cat?";`.

Copyright ©2011 sheepsqueezers.com

String

Properties

- Chars - Gets the Char object at a specified position in the current String object.
- Length - Gets the number of characters in the current String object.

Operators (static)

- Equality - Determines whether two specified strings have the same value. Can use the "==" symbol.
- Inequality - Determines whether two specified strings have different values. Can use the "!=" symbol.

Fields (static)

- Empty – Represents the empty string and is read-only.

Methods

- Clone - Returns a reference to this instance of String.
- Compare(String, String) - Compares two specified String objects and returns an integer that indicates their relative position in the sort order.
- Compare(String, String, Boolean) - Compares two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
- Compare(String, String, StringComparison) - Compares two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
- Compare(String, String, Boolean, CultureInfo) - Compares two specified String objects, ignoring or honoring their case, and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.
- Compare(String, String, CultureInfo, CompareOptions) - Compares two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two strings to each other in the sort order.
- Compare(String, Int32, String, Int32, Int32) - Compares substrings of two specified String objects and returns an integer that indicates their relative position in the sort order.
- Compare(String, Int32, String, Int32, Int32, Boolean) - Compares substrings of two specified String objects, ignoring or honoring their case, and returns an integer that indicates their relative position in the sort order.
- Compare(String, Int32, String, Int32, Int32, StringComparison) - Compares substrings of two specified String objects using the specified rules, and returns an integer that indicates their relative position in the sort order.
- Compare(String, Int32, String, Int32, Int32, Boolean, CultureInfo) - Compares substrings of two specified String objects, ignoring or honoring their case and using culture-specific information to influence the comparison, and returns an integer that indicates their relative position in the sort order.
- Compare(String, Int32, String, Int32, Int32, CultureInfo, CompareOptions) - Compares substrings of two specified String objects using the specified comparison options and culture-specific information to influence the comparison, and returns an integer that indicates the relationship of the two substrings to each other in the sort order.

Copyright ©2011 sheepsqueezers.com

Methods (continued)

- CompareOrdinal(String, String) - Compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string.
- CompareOrdinal(String, Int32, String, Int32, Int32) - Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring.
- CompareTo(Object) - Compares this instance with a specified Object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified Object.
- CompareTo(String) - Compares this instance with a specified String object and indicates whether this instance precedes, follows, or appears in the same position in the sort order as the specified String.
- Concat(Object) - Creates the string representation of a specified object.
- Concat(Object[]) - Concatenates the string representations of the elements in a specified Object array.
- Concat(IEnumerable<String>) - Concatenates the members of a constructed IEnumerable<T> collection of type String.
- Concat(String[]) - Concatenates the elements of a specified String array.
- Concat(Object, Object) - Concatenates the string representations of two specified objects.
- Concat(String, String) - Concatenates two specified instances of String.
- Concat(Object, Object, Object) - Concatenates the string representations of three specified objects.
- Concat(String, String, String) - Concatenates three specified instances of String.
- Concat(Object, Object, Object, Object) - Concatenates the string representations of four specified objects and any objects specified in an optional variable length parameter list.
- Concat(String, String, String, String) - Concatenates four specified instances of String.
- Concat<T>(IEnumerable<T>) - Concatenates the members of an IEnumerable<T> implementation.
- Contains - Returns a value indicating whether the specified String object occurs within this string.
- Copy - Creates a new instance of String with the same value as a specified String.
- CopyTo - Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters.
- EndsWith(String) - Determines whether the end of this string instance matches the specified string.
- EndsWith(String, StringComparison) - Determines whether the end of this string instance matches the specified string when compared using the specified comparison option.
- EndsWith(String, Boolean, CultureInfo) - Determines whether the end of this string instance matches the specified string when compared using the specified culture.
- Equals(Object) - Determines whether this instance and a specified object, which must also be a String object, have the same value. (Overrides Object.Equals(Object).)
- Equals(String) - Determines whether this instance and another specified String object have the same value.
- Equals(String, String) - Determines whether two specified String objects have the same value.
- Equals(String, StringComparison) - Determines whether this string and a specified String object have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.
- Equals(String, String, StringComparison) - Determines whether two specified String objects have the same value. A parameter specifies the culture, case, and sort rules used in the comparison.

Copyright ©2011 sheepsqueezers.com

sheepsqueezers.com

Methods (continued)

- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- Format(String, Object) - Replaces one or more format items in a specified string with the string representation of a specified object.
- Format(String, Object[]) - Replaces the format item in a specified string with the string representation of a corresponding object in a specified array.
- Format(IFormatProvider, String, Object[]) - Replaces the format item in a specified string with the string representation of a corresponding object in a specified array. A specified parameter supplies culture-specific formatting information.
- Format(String, Object, Object) - Replaces the format items in a specified string with the string representation of two specified objects.
- Format(String, Object, Object, Object) - Replaces the format items in a specified string with the string representation of three specified objects.
- GetEnumerator - Retrieves an object that can iterate through the individual characters in this string.
- GetHashCode - Returns the hash code for this string. (Overrides Object.GetHashCode().)
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- GetTypeCode - Returns the TypeCode for class String.
- IndexOf(Char) - Reports the index of the first occurrence of the specified Unicode character in this string.
- IndexOf(String) - Reports the index of the first occurrence of the specified string in this instance.
- IndexOf(Char, Int32) - Reports the index of the first occurrence of the specified Unicode character in this string. The search starts at a specified character position.
- IndexOf(String, Int32) - Reports the index of the first occurrence of the specified string in this instance. The search starts at a specified character position.
- IndexOf(String, StringComparison) - Reports the index of the first occurrence of the specified string in the current String object. A parameter specifies the type of search to use for the specified string.
- IndexOf(Char, Int32, Int32) - Reports the index of the first occurrence of the specified character in this instance. The search starts at a specified character position and examines a specified number of character positions.
- IndexOf(String, Int32, Int32) - Reports the index of the first occurrence of the specified string in this instance. The search starts at a specified character position and examines a specified number of character positions.
- IndexOf(String, Int32, StringComparison) - Reports the index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string and the type of search to use for the specified string.
- IndexOf(String, Int32, Int32, StringComparison) - Reports the index of the first occurrence of the specified string in the current String object. Parameters specify the starting search position in the current string, the number of characters in the current string to search, and the type of search to use for the specified string.
- IndexOfAny(Char[]) - Reports the index of the first occurrence in this instance of any character in a specified array of Unicode characters.
- IndexOfAny(Char[], Int32) - Reports the index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position.
- IndexOfAny(Char[], Int32, Int32) - Reports the index of the first occurrence in this instance of any character in a specified array of Unicode characters. The search starts at a specified character position and examines a specified number of character positions.
- Insert - Inserts a specified instance of String at a specified index position in this instance.

Copyright ©2011 sheepsqueezers.com

Methods (continued)

- Intern - Retrieves the system's reference to the specified String.
- IsInterned - Retrieves a reference to a specified String.
- IsNormalized() - Indicates whether this string is in Unicode normalization form C.
- IsNormalized(NormalizationForm) - Indicates whether this string is in the specified Unicode normalization form.
- IsNullOrEmpty - Indicates whether the specified string is null or an Empty string.
- IsNullOrWhiteSpace - Indicates whether a specified string is null, empty, or consists only of white-space characters.
- Join(String, IEnumerable<String>) - Concatenates the members of a constructed IEnumerable<T> collection of type String, using the specified separator between each member.
- Join(String, Object[]) - Concatenates the elements of an object array, using the specified separator between each element.
- Join(String, String[]) - Concatenates all the elements of a string array, using the specified separator between each element.
- Join(String, String[], Int32, Int32) - Concatenates the specified elements of a string array, using the specified separator between each element.
- Join<T>(String, IEnumerable<T>) - Concatenates the members of a collection, using the specified separator between each member.
- LastIndexOf(Char) - Reports the index position of the last occurrence of a specified Unicode character within this instance.
- LastIndexOf(String) - Reports the index position of the last occurrence of a specified string within this instance.
- LastIndexOf(Char, Int32) - Reports the index position of the last occurrence of a specified Unicode character within this instance. The search starts at a specified character position.
- LastIndexOf(String, Int32) - Reports the index position of the last occurrence of a specified string within this instance. The search starts at a specified character position.
- LastIndexOf(String, StringComparison) - Reports the index of the last occurrence of a specified string within the current String object. A parameter specifies the type of search to use for the specified string.
- LastIndexOf(Char, Int32, Int32) - Reports the index position of the last occurrence of the specified Unicode character in a substring within this instance. The search starts at a specified character position and examines a specified number of character positions.
- LastIndexOf(String, Int32, Int32) - Reports the index position of the last occurrence of a specified string within this instance. The search starts at a specified character position and examines a specified number of character positions.
- LastIndexOf(String, Int32, StringComparison) - Reports the index of the last occurrence of a specified string within the current String object. Parameters specify the starting search position in the current string, and type of search to use for the specified string.
- LastIndexOf(String, Int32, Int32, StringComparison) - Reports the index position of the last occurrence of a specified string within this instance. Parameters specify the starting search position in the current string, the number of characters in the current string to search, and the type of search to use for the specified string.
- LastIndexOfAny(Char[]) - Reports the index position of the last occurrence in this instance of one or more characters specified in a Unicode array.
- LastIndexOfAny(Char[], Int32) - Reports the index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position.
- LastIndexOfAny(Char[], Int32, Int32) - Reports the index position of the last occurrence in this instance of one or more characters specified in a Unicode array. The search starts at a specified character position and examines a specified number of character positions.

Copyright ©2011 sheepsqueezers.com

Methods (continued)

- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- Normalize() - Returns a new string whose textual value is the same as this string, but whose binary representation is in Unicode normalization form C.
- Normalize(NormalizationForm) - Returns a new string whose textual value is the same as this string, but whose binary representation is in the specified Unicode normalization form.
- PadLeft(Int32) - Returns a new string that right-aligns the characters in this instance by padding them with spaces on the left, for a specified total length.
- PadLeft(Int32, Char) - Returns a new string that right-aligns the characters in this instance by padding them on the left with a specified Unicode character, for a specified total length.
- PadRight(Int32) - Returns a new string that left-aligns the characters in this string by padding them with spaces on the right, for a specified total length.
- PadRight(Int32, Char) - Returns a new string that left-aligns the characters in this string by padding them on the right with a specified Unicode character, for a specified total length.
- Remove(Int32) - Deletes all the characters from this string beginning at a specified position and continuing through the last position.
- Remove(Int32, Int32) - Deletes a specified number of characters from this instance beginning at a specified position.
- Replace(Char, Char) - Returns a new string in which all occurrences of a specified Unicode character in this instance are replaced with another specified Unicode character.
- Replace(String, String) - Returns a new string in which all occurrences of a specified string in the current instance are replaced with another specified string.
- Split(Char[]) - Returns a string array that contains the substrings in this instance that are delimited by elements of a specified Unicode character array.
- Split(Char[], Int32) - Returns a string array that contains the substrings in this instance that are delimited by elements of a specified Unicode character array. A parameter specifies the maximum number of substrings to return.
- Split(Char[], StringSplitOptions) - Returns a string array that contains the substrings in this string that are delimited by elements of a specified Unicode character array. A parameter specifies whether to return empty array elements.
- Split(String[], StringSplitOptions) - Returns a string array that contains the substrings in this string that are delimited by elements of a specified string array. A parameter specifies whether to return empty array elements.
- Split(Char[], Int32, StringSplitOptions) - Returns a string array that contains the substrings in this string that are delimited by elements of a specified Unicode character array. Parameters specify the maximum number of substrings to return and whether to return empty array elements.
- Split(String[], Int32, StringSplitOptions) - Returns a string array that contains the substrings in this string that are delimited by elements of a specified string array. Parameters specify the maximum number of substrings to return and whether to return empty array elements.
- StartsWith(String) - Determines whether the beginning of this string instance matches the specified string.
- StartsWith(String, StringComparison) - Determines whether the beginning of this string instance matches the specified string when compared using the specified comparison option.
- StartsWith(String, Boolean, CultureInfo) - Determines whether the beginning of this string instance matches the specified string when compared using the specified culture.

Copyright ©2011 sheepsqueezers.com

Methods (continued)

- Substring(Int32) - Retrieves a substring from this instance. The substring starts at a specified character position.
- Substring(Int32, Int32) - Retrieves a substring from this instance. The substring starts at a specified character position and has a specified length.
- ToCharArray() - Copies the characters in this instance to a Unicode character array.
- ToCharArray(Int32, Int32) - Copies the characters in a specified substring in this instance to a Unicode character array.
- ToLower() - Returns a copy of this string converted to lowercase.
- ToLower(CultureInfo) - Returns a copy of this string converted to lowercase, using the casing rules of the specified culture.
- ToLowerInvariant - Returns a copy of this String object converted to lowercase using the casing rules of the invariant culture.
- ToString() - Returns this instance of String; no actual conversion is performed. (Overrides Object.ToString().)
- ToString(IFormatProvider) - Returns this instance of String; no actual conversion is performed.
- ToUpper() - Returns a copy of this string converted to uppercase.
- ToUpper(CultureInfo) - Returns a copy of this string converted to uppercase, using the casing rules of the specified culture.
- ToUpperInvariant - Returns a copy of this String object converted to uppercase using the casing rules of the invariant culture.
- Trim() - Removes all leading and trailing white-space characters from the current String object.
- Trim(Char[]) - Removes all leading and trailing occurrences of a set of characters specified in an array from the current String object.
- TrimEnd - Removes all trailing occurrences of a set of characters specified in an array from the current String object.
- TrimStart - Removes all leading occurrences of a set of characters specified in an array from the current String object.

Now, most of these are clear as to what they do, but the `Format` methods may not be. Please see the following document for more information about how to create format strings: http://msdn.microsoft.com/en-us/library/txafckwd.aspx. Note that these formats can be used in the `Console.WriteLine` and `Console.Write` methods and are available wherever fine products are sold...tee-hee…

For example:

```
//0 indicates first variable, iValue1, and 1 indicates second variable, iValue2.
Console.WriteLine("Here is this value={0} and here is that value={1}.",iValue1,iValue2);
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `StringComparer`

Copyright ©2011 sheepsqueezers.com

`StringComparer`

The `StringComparer` class represents a string comparison operation that uses specific case and culture-based or ordinal comparison rules. See Microsoft's website for more on this class.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `TimeZone, TimeZoneInfo, TimeZoneInfo.AdjustmentRule`

Copyright ©2011 sheepsqueezers.com

TimeZone, TimeZoneInfo, TimeZoneInfo.AdjustmentRule

The `TimeZone` class represents a time zone. Microsoft recommends using the `TimeZoneInfo` class instead of the `TimeZone` class.

The `TimeZoneInfo` class represents any time zone in the world. An instance of the `TimeZoneInfo` class is immutable; that is, once instantiated, you cannot modify it.

Properties

- BaseUtcOffset - Gets the time difference between the current time zone's standard time and Coordinated Universal Time (UTC).
- DaylightName - Gets the localized display name for the current time zone's daylight saving time.
- DisplayName - Gets the localized general display name that represents the time zone.
- Id - Gets the time zone identifier.
- Local - Gets a TimeZoneInfo object that represents the local time zone.
- StandardName - Gets the localized display name for the time zone's standard time.
- SupportsDaylightSavingTime - Gets a value indicating whether the time zone has any daylight saving time rules.
- Utc - Gets a TimeZoneInfo object that represents the Coordinated Universal Time (UTC) zone.

Methods

- ClearCachedData - Clears cached time zone data.
- ConvertTime - Converts a time to the time in a particular time zone. There are 3 overloads to this method.
- ConvertTimeBySystemTimeZoneId - Converts a time to the time in another time zone based on the time zone's identifier. There are 3 overloads to this method.
- ConvertTimeFromUtc - Converts a Coordinated Universal Time (UTC) to the time in a specified time zone.
- ConvertTimeToUtc(DateTime) - Converts the current date and time to Coordinated Universal Time (UTC).
- ConvertTimeToUtc(DateTime, TimeZoneInfo) - Converts the time in a specified time zone to Coordinated Universal Time (UTC).
- CreateCustomTimeZone - Creates a custom time zone. There are 3 overloads to this method.
- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Equals(TimeZoneInfo) - Determines whether the current TimeZoneInfo object and another TimeZoneInfo object are equal.
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- FindSystemTimeZoneById - Retrieves a TimeZoneInfo object from the registry based on its identifier.
- FromSerializedString - Deserializes a string to re-create an original serialized TimeZoneInfo object.
- GetAdjustmentRules - Retrieves an array of TimeZoneInfo.AdjustmentRule objects that apply to the current TimeZoneInfo object.

Copyright ©2011 sheepsqueezers.com

TimeZone, TimeZoneInfo, TimeZoneInfo.AdjustmentRule

Methods (continued)

- GetAmbiguousTimeOffsets(DateTime) - Returns information about the possible dates and times that an ambiguous date and time can be mapped to.
- GetAmbiguousTimeOffsets(DateTimeOffset) - Returns information about the possible dates and times that an ambiguous date and time can be mapped to.
- GetHashCode - Serves as a hash function for hashing algorithms and data structures such as hash tables. (Overrides Object.GetHashCode().)
- GetSystemTimeZones - Returns a sorted collection of all the time zones about which information is available on the local system.
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- GetUtcOffset(DateTime) - Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.
- GetUtcOffset(DateTimeOffset) - Calculates the offset or difference between the time in this time zone and Coordinated Universal Time (UTC) for a particular date and time.
- HasSameRules - Indicates whether the current object and another TimeZoneInfo object have the same adjustment rules.
- IsAmbiguousTime(DateTime) - Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.
- IsAmbiguousTime(DateTimeOffset) - Determines whether a particular date and time in a particular time zone is ambiguous and can be mapped to two or more Coordinated Universal Time (UTC) times.
- IsDaylightSavingTime(DateTime) - Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.
- IsDaylightSavingTime(DateTimeOffset) - Indicates whether a specified date and time falls in the range of daylight saving time for the time zone of the current TimeZoneInfo object.
- IsInvalidTime - Indicates whether a particular date and time is invalid.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- ToSerializedString - Converts the current TimeZoneInfo object to a serialized string.
- ToString - Returns the current TimeZoneInfo object's display name. (Overrides Object.ToString().)

The `TimeZoneInfo.AdjustmentRule` class provides information about a time zone adjustment, such as the transition to and from daylight saving time. See Microsoft's website for more on this class.

Below is an example displaying the `TimeZoneInfo` properties on my computer. See next slide.

Copyright ©2011 sheepsqueezers.com

TimeZone, TimeZoneInfo, TimeZoneInfo.AdjustmentRule


```
using System;

class MainProgram {

 public static void Main() {

  TimeZoneInfo oTZI = TimeZoneInfo.Local; //Local property returns a TimeZoneInfo object
  Console.WriteLine("DaylightName={0}",oTZI.DaylightName);
  Console.WriteLine("DisplayName={0}",oTZI.DisplayName);
  Console.WriteLine("StandardName={0}",oTZI.StandardName);
  Console.WriteLine("Id={0}",oTZI.Id);
  Console.WriteLine("Supports Daylight Savings Time?={0}",oTZI.SupportsDaylightSavingTime);

 }

}
```

The results are:

```
DaylightName=Eastern Daylight Time
DisplayName=(GMT-05:00) Eastern Time (US & Canada)
StandardName=Eastern Standard Time
Id=Eastern Standard Time
Supports Daylight Savings Time?=True
```

You can also use some of the static methods to convert datetimes to other time zones. For example,

```
 DateTime oDT_LOCAL = DateTime.Now; //Current date and time
 DateTime oDT_UTC = TimeZoneInfo.ConvertTimeToUtc(oDT_LOCAL); //Current date and time at UTC.
 Console.WriteLine("The local current date/time is {0}, but the UTC current date/time is
{1}",oDT_LOCAL.ToString(),oDT_UTC.ToString());
```

…returns the following data on my computer…

```
The local current date/time is 5/23/2011 12:05:52 PM, but the UTC current date/time is 5/23/2011 4:05:52 PM
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Tuple`

Copyright ©2011 sheepsqueezers.com

Tuple

The `Tuple` classes (all nine of them!) provides methods for creating tuple objects. According to Microsoft's website: *A tuple is a data structure that has a specific number and sequence of values. The Tuple<T1, T2> class represents a 2-tuple, or pair, which is a tuple that has two components. A 2-tuple is similar to a KeyValuePair<TKey, TValue> structure. You can instantiate a Tuple<T1, T2> object by calling either the Tuple<T1, T2> constructor or the static Tuple.Create<T1, T2>(T1, T2) method. You can retrieve the values of the tuple's components by using the read-only Item1 and Item2 instance properties. Similar for the other Tuple classes.*

Tuples are commonly used in four different ways:

- *To represent a single set of data. For example, a tuple can represent a record in a database, and its components can represent that record's fields.*
- *To provide easy access to, and manipulation of, a data set. For example, defining an array of Tuple<T1, T2> objects that contain the names of students and their corresponding test scores. You can then iterate through the array to calculate the mean test score.*
- *To return multiple values from a method without the use of out parameters (in C#) or ByRef parameters (in Visual Basic). For example, using a Tuple<T1, T2> object to return the quotient and the remainder that result from integer division.*
- *To pass multiple values to a method through a single parameter. For example, the Thread.Start(Object) method has a single parameter that lets you supply one value to the method that the thread executes at startup. If you supply a Tuple<T1, T2> object as the method argument, you can supply the thread's startup routine with two items of data.*

Copyright ©2011 sheepsqueezers.com

Tuple

Note that the `Tuple` class provides solely *static* methods. The remaining `Tuple` methods provide non-static methods and properties. Below we list the members for the `Tuple` class.

Methods (static)

- Create<T1>(T1) - Creates a new 1-tuple, or singleton.
- Create<T1, T2>(T1, T2) - Creates a new 2-tuple, or pair.
- Create<T1, T2, T3>(T1, T2, T3) - Creates a new 3-tuple, or triple.
- Create<T1, T2, T3, T4>(T1, T2, T3, T4) - Creates a new 4-tuple, or quadruple.
- Create<T1, T2, T3, T4, T5>(T1, T2, T3, T4, T5) - Creates a new 5-tuple, or quintuple.
- Create<T1, T2, T3, T4, T5, T6>(T1, T2, T3, T4, T5, T6) - Creates a new 6-tuple, or sextuple.
- Create<T1, T2, T3, T4, T5, T6, T7>(T1, T2, T3, T4, T5, T6, T7) - Creates a new 7-tuple, or septuple.
- Create<T1, T2, T3, T4, T5, T6, T7, T8>(T1, T2, T3, T4, T5, T6, T7, T8) - Creates a new 8-tuple, or octuple.

Note that you don't have to provide the generics for the methods listed above since the compiler will determine that. One place that you could use a tuple is when gathering HTTP Request Header information such as HTTPRequest, Referrer, UserAgent, etc. For example:

```
using System;
using System.Collections.Generic;

class MainProgram {

 public static void Main() {

  //Create a tuple to hold the 8 fields containing the HTTP Header Info: Request, Accept, Accept-Encoding,
  // Accept-Language, Connection, Host, Referrer, User-Agent
  var tHTTPHeaderInfo = Tuple.Create("GET /sw/m/php/test.php HTTP/1.1","image/gif, image/jpeg",
                                     "gzip, deflate","en-us","Keep-Alive", "www.bobs.com",
                                     "http://www.sheepsqueezers.com", "Mozilla/4.0");

  //To pull a single piece of information, use the corresponding Item# property for the tuple.
  Console.WriteLine("Referrer={0}",tHTTPHeaderInfo.Item7); //Referrer=http://www.sheepsqueezers.com

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Type`

Copyright ©2011 sheepsqueezers.com

Type

The `Type` class represents type declarations: class types, interface types, array types, value types, enumeration types, type parameters, generic type definitions, and open or closed constructed generic types. According to Microsoft's website: *Type is the root of the System.Reflection functionality and is the primary way to access metadata. Use the members of Type to get information about a type declaration, such as the constructors, methods, fields, properties, and events of a class, as well as the module and the assembly in which the class is deployed.*

The C# typeof operator (GetType operator in Visual Basic, typeid operator in Visual C++) returns a Type object.

A Type object that represents a type is unique; that is, two Type object references refer to the same object if and only if they represent the same type. This allows for comparison of Type objects using reference equality.

Constructors (protected)

- Type - Initializes a new instance of the Type class. You can use the typeof operator to return a Type object instead of instantiating with this constructor.

Properties

- Assembly - Gets the Assembly in which the type is declared. For generic types, gets the Assembly in which the generic type is defined.
- AssemblyQualifiedName - Gets the assembly-qualified name of the Type, which includes the name of the assembly from which the Type was loaded.
- Attributes - Gets the attributes associated with the Type.
- BaseType - Gets the type from which the current Type directly inherits.
- ContainsGenericParameters - Gets a value indicating whether the current Type object has type parameters that have not been replaced by specific types.
- DeclaringMethod - Gets a MethodBase that represents the declaring method, if the current Type represents a type parameter of a generic method.
- DeclaringType - Gets the type that declares the current nested type or generic type parameter. (Overrides MemberInfo.DeclaringType.)
- DefaultBinder - Gets a reference to the default binder, which implements internal rules for selecting the appropriate members to be called by InvokeMember.
- FullName - Gets the fully qualified name of the Type, including the namespace of the Type but not the assembly.

Copyright ©2011 sheepsqueezers.com

Type

Properties (continued)

- GenericParameterAttributes - Gets a combination of GenericParameterAttributes flags that describe the covariance and special constraints of the current generic type parameter.
- GenericParameterPosition - Gets the position of the type parameter in the type parameter list of the generic type or method that declared the parameter, when the Type object represents a type parameter of a generic type or a generic method.
- GUID - Gets the GUID associated with the Type.
- HasElementType - Gets a value indicating whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.
- IsAbstract - Gets a value indicating whether the Type is abstract and must be overridden.
- IsAnsiClass - Gets a value indicating whether the string format attribute AnsiClass is selected for the Type.
- IsArray - Gets a value indicating whether the Type is an array.
- IsAutoClass - Gets a value indicating whether the string format attribute AutoClass is selected for the Type.
- IsAutoLayout - Gets a value indicating whether the class layout attribute AutoLayout is selected for the Type.
- IsByRef - Gets a value indicating whether the Type is passed by reference.
- IsClass - Gets a value indicating whether the Type is a class; that is, not a value type or interface.
- IsCOMObject - Gets a value indicating whether the Type is a COM object.
- IsContextful - Gets a value indicating whether the Type can be hosted in a context.
- IsEnum - Gets a value indicating whether the current Type represents an enumeration.
- IsExplicitLayout - Gets a value indicating whether the class layout attribute ExplicitLayout is selected for the Type.
- IsGenericParameter - Gets a value indicating whether the current Type represents a type parameter in the definition of a generic type or method.
- IsGenericType - Gets a value indicating whether the current type is a generic type.
- IsGenericTypeDefinition - Gets a value indicating whether the current Type represents a generic type definition, from which other generic types can be constructed.
- IsImport - Gets a value indicating whether the Type has a ComImportAttribute attribute applied, indicating that it was imported from a COM type library.
- IsInterface - Gets a value indicating whether the Type is an interface; that is, not a class or a value type.
- IsLayoutSequential - Gets a value indicating whether the class layout attribute SequentialLayout is selected for the Type.
- IsMarshalByRef - Gets a value indicating whether the Type is marshaled by reference.
- IsNested - Gets a value indicating whether the current Type object represents a type whose definition is nested inside the definition of another type.
- IsNestedAssembly - Gets a value indicating whether the Type is nested and visible only within its own assembly.
- IsNestedFamANDAssem - Gets a value indicating whether the Type is nested and visible only to classes that belong to both its own family and its own assembly.
- IsNestedFamily - Gets a value indicating whether the Type is nested and visible only within its own family.
- IsNestedFamORAssem - Gets a value indicating whether the Type is nested and visible only to classes that belong to either its own family or to its own assembly.
- IsNestedPrivate - Gets a value indicating whether the Type is nested and declared private.
- IsNestedPublic - Gets a value indicating whether a class is nested and declared public.
- IsNotPublic - Gets a value indicating whether the Type is not declared public.

Copyright ©2011 sheepsqueezers.com

Type

Properties (continued)

- IsPointer - Gets a value indicating whether the Type is a pointer.
- IsPrimitive - Gets a value indicating whether the Type is one of the primitive types.
- IsPublic - Gets a value indicating whether the Type is declared public.
- IsSealed - Gets a value indicating whether the Type is declared sealed.
- IsSecurityCritical - Gets a value that indicates whether the current type is security-critical or security-safe-critical at the current trust level, and therefore can perform critical operations.
- IsSecuritySafeCritical - Gets a value that indicates whether the current type is security-safe-critical at the current trust level; that is, whether it can perform critical operations and can be accessed by transparent code.
- IsSecurityTransparent - Gets a value that indicates whether the current type is transparent at the current trust level, and therefore cannot perform critical operations.
- IsSerializable - Gets a value indicating whether the Type is serializable.
- IsSpecialName - Gets a value indicating whether the Type has a name that requires special handling.
- IsUnicodeClass - Gets a value indicating whether the string format attribute UnicodeClass is selected for the Type.
- IsValueType - Gets a value indicating whether the Type is a value type.
- IsVisible - Gets a value indicating whether the Type can be accessed by code outside the assembly.
- MemberType - Gets a MemberTypes value indicating that this member is a type or a nested type. (Overrides MemberInfo.MemberType.)
- MetadataToken - Gets a value that identifies a metadata element. (Inherited from MemberInfo.)
- Module - Gets the module (the DLL) in which the current Type is defined.
- Name - Gets the name of the current member. (Inherited from MemberInfo.)
- Namespace - Gets the namespace of the Type.
- ReflectedType - Gets the class object that was used to obtain this member. (Overrides MemberInfo.ReflectedType.)
- StructLayoutAttribute - Gets a StructLayoutAttribute that describes the layout of the current type.
- TypeHandle - Gets the handle for the current Type.
- TypeInitializer - Gets the initializer for the Type.
- UnderlyingSystemType - Indicates the type provided by the common language runtime that represents this type.

Fields (static)

- Delimiter - Separates names in the namespace of the Type. This field is read-only.
- EmptyTypes - Represents an empty array of type Type. This field is read-only.
- FilterAttribute - Represents the member filter used on attributes. This field is read-only.
- FilterName - Represents the case-sensitive member filter used on names. This field is read-only.
- FilterNameIgnoreCase - Represents the case-insensitive member filter used on names. This field is read-only.
- Missing - Represents a missing value in the Type information. This field is read-only.

Operators (static)

- Equality - Indicates whether two Type objects are equal.
- Inequality - Indicates whether two Type objects are not equal.

Copyright ©2011 sheepsqueezers.com

Type

Methods

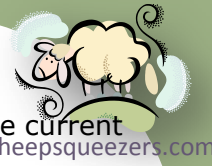
- Equals(Object) - Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Object. (Overrides MemberInfo.Equals(Object).)
- Equals(Type) - Determines if the underlying system type of the current Type is the same as the underlying system type of the specified Type.
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- FindInterfaces - Returns an array of Type objects representing a filtered list of interfaces implemented or inherited by the current Type.
- FindMembers - Returns a filtered array of MemberInfo objects of the specified member type.
- GetArrayRank - Gets the number of dimensions in an Array.
- GetAttributeFlagsImpl - When overridden in a derived class, implements the Attributes property and gets a bitmask indicating the attributes associated with the Type.
- GetConstructor(Type[]) - Searches for a public instance constructor whose parameters match the types in the specified array.
- GetConstructor(BindingFlags, Binder, Type[], ParameterModifier[]) - Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints.
- GetConstructor(BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[]) - Searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
- GetConstructorImpl - When overridden in a derived class, searches for a constructor whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
- GetConstructors() - Returns all the public constructors defined for the current Type.
- GetConstructors(BindingFlags) - When overridden in a derived class, searches for the constructors defined for the current Type, using the specified BindingFlags.
- GetCustomAttributes(Boolean) - When overridden in a derived class, returns an array of all custom attributes applied to this member. (Inherited from MemberInfo.)
- GetCustomAttributes(Type, Boolean) - When overridden in a derived class, returns an array of custom attributes applied to this member and identified by Type. (Inherited from MemberInfo.)
- GetCustomAttributesData - Returns a list of CustomAttributeData objects representing data about the attributes that have been applied to the target member. (Inherited from MemberInfo.)
- GetDefaultMembers - Searches for the members defined for the current Type whose DefaultMemberAttribute is set.
- GetElementType - When overridden in a derived class, returns the Type of the object encompassed or referred to by the current array, pointer or reference type.
- GetEnumName - Returns the name of the constant that has the specified value, for the current enumeration type.
- GetEnumNames - Returns the names of the members of the current enumeration type.
- GetEnumUnderlyingType - Returns the underlying type of the current enumeration type.
- GetEnumValues - Returns an array of the values of the constants in the current enumeration type.
- GetEvent(String) - Returns the EventInfo object representing the specified public event.
- GetEvent(String, BindingFlags) - When overridden in a derived class, returns the EventInfo object representing the specified event, using the specified binding constraints.
- GetEvents() - Returns all the public events that are declared or inherited by the current Type.

sheepsqueezers.com

Copyright ©2011 sheepsqueezers.com

Type

Methods (continued)



- GetEvents(BindingFlags) - When overridden in a derived class, searches for events that are declared or inherited by the current Type, using the specified binding constraints.
- GetField(String) - Searches for the public field with the specified name.
- GetField(String, BindingFlags) - Searches for the specified field, using the specified binding constraints.
- GetFields() - Returns all the public fields of the current Type.
- GetFields(BindingFlags) - When overridden in a derived class, searches for the fields defined for the current Type, using the specified binding constraints.
- GetGenericArguments - Returns an array of Type objects that represent the type arguments of a generic type or the type parameters of a generic type definition.
- GetGenericParameterConstraints - Returns an array of Type objects that represent the constraints on the current generic type parameter.
- GetGenericTypeDefinition - Returns a Type object that represents a generic type definition from which the current generic type can be constructed.
- GetHashCode - Returns the hash code for this instance. (Overrides MemberInfo.GetHashCode().)
- GetInterface(String) - Searches for the interface with the specified name.
- GetInterface(String, Boolean) - When overridden in a derived class, searches for the specified interface, specifying whether to do a case-insensitive search for the interface name.
- GetInterfaceMap - Returns an interface mapping for the specified interface type.
- GetInterfaces - When overridden in a derived class, gets all the interfaces implemented or inherited by the current Type.
- GetMember(String) - Searches for the public members with the specified name.
- GetMember(String, BindingFlags) - Searches for the specified members, using the specified binding constraints.
- GetMember(String, MemberTypes, BindingFlags) - Searches for the specified members of the specified member type, using the specified binding constraints.
- GetMembers() - Returns all the public members of the current Type.
- GetMembers(BindingFlags) - When overridden in a derived class, searches for the members defined for the current Type, using the specified binding constraints.
- GetMethod(String) - Searches for the public method with the specified name.
- GetMethod(String, BindingFlags) - Searches for the specified method, using the specified binding constraints.
- GetMethod(String, Type[]) - Searches for the specified public method whose parameters match the specified argument types.
- GetMethod(String, Type[], ParameterModifier[]) - Searches for the specified public method whose parameters match the specified argument types and modifiers.
- GetMethod(String, BindingFlags, Binder, Type[], ParameterModifier[]) - Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints.
- GetMethod(String, BindingFlags, Binder, CallingConventions, Type[], ParameterModifier[]) - Searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.
- GetMethodImpl - When overridden in a derived class, searches for the specified method whose parameters match the specified argument types and modifiers, using the specified binding constraints and the specified calling convention.

Copyright ©2011 sheepsqueezers.com

Type

Methods (continued)

- GetMethods() - Returns all the public methods of the current Type.
- GetMethods(BindingFlags) - When overridden in a derived class, searches for the methods defined for the current Type, using the specified binding constraints.
- GetNestedType(String) - Searches for the public nested type with the specified name.
- GetNestedType(String, BindingFlags) - When overridden in a derived class, searches for the specified nested type, using the specified binding constraints.
- GetNestedTypes() - Returns the public types nested in the current Type.
- GetNestedTypes(BindingFlags) - When overridden in a derived class, searches for the types nested in the current Type, using the specified binding constraints.
- GetProperties() - Returns all the public properties of the current Type.
- GetProperties(BindingFlags) - When overridden in a derived class, searches for the properties of the current Type, using the specified binding constraints.
- GetProperty(String) - Searches for the public property with the specified name.
- GetProperty(String, BindingFlags) - Searches for the specified property, using the specified binding constraints.
- GetProperty(String, Type) - Searches for the public property with the specified name and return type.
- GetProperty(String, Type[]) - Searches for the specified public property whose parameters match the specified argument types.
- GetProperty(String, Type, Type[]) - Searches for the specified public property whose parameters match the specified argument types.
- GetProperty(String, Type, Type[], ParameterModifier[]) - Searches for the specified public property whose parameters match the specified argument types and modifiers.
- GetProperty(String, BindingFlags, Binder, Type, Type[], ParameterModifier[]) - Searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
- GetPropertyImpl - When overridden in a derived class, searches for the specified property whose parameters match the specified argument types and modifiers, using the specified binding constraints.
- GetType() - Gets the current Type. In XNA Framework 3.0, this member is inherited from Object.GetType(). In Portable Class Library Portable Class Library, this member is inherited from Object.GetType().
- GetType(String) - Gets the Type with the specified name, performing a case-sensitive search.
- GetType(String, Boolean) - Gets the Type with the specified name, performing a case-sensitive search and specifying whether to throw an exception if the type is not found.
- GetType(String, Boolean, Boolean) - Gets the Type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found.
- GetType(String, Func<AssemblyName, Assembly>, Func<Assembly, String, Boolean, Type>) - Gets the type with the specified name, optionally providing custom methods to resolve the assembly and the type.
- GetType(String, Func<AssemblyName, Assembly>, Func<Assembly, String, Boolean, Type>, Boolean) - Gets the type with the specified name, specifying whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
- GetType(String, Func<AssemblyName, Assembly>, Func<Assembly, String, Boolean, Type>, Boolean, Boolean) - Gets the type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found, and optionally providing custom methods to resolve the assembly and the type.
- GetTypeArray - Gets the types of the objects in the specified array.
- GetTypeCode - Gets the underlying type code of the specified Type.

sheepsqueezers.com

Copyright ©2011 sheepsqueezers.com

Type

Methods (continued)

- GetTypeCodeImpl - Returns the underlying type code of the specified Type.
- GetTypeFromCLSID(Guid) - Gets the type associated with the specified class identifier (CLSID).
- GetTypeFromCLSID(Guid, Boolean) - Gets the type associated with the specified class identifier (CLSID), specifying whether to throw an exception if an error occurs while loading the type.
- GetTypeFromCLSID(Guid, String) - Gets the type associated with the specified class identifier (CLSID) from the specified server.
- GetTypeFromCLSID(Guid, String, Boolean) - Gets the type associated with the specified class identifier (CLSID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.
- GetTypeFromHandle - Gets the type referenced by the specified type handle.
- GetTypeFromProgID(String) - Gets the type associated with the specified program identifier (ProgID), returning null if an error is encountered while loading the Type.
- GetTypeFromProgID(String, Boolean) - Gets the type associated with the specified program identifier (ProgID), specifying whether to throw an exception if an error occurs while loading the type.
- GetTypeFromProgID(String, String) - Gets the type associated with the specified program identifier (progID) from the specified server, returning null if an error is encountered while loading the type.
- GetTypeFromProgID(String, String, Boolean) - Gets the type associated with the specified program identifier (progID) from the specified server, specifying whether to throw an exception if an error occurs while loading the type.
- GetTypeHandle - Gets the handle for the Type of a specified object.
- HasElementTypeImpl - When overridden in a derived class, implements the HasElementType property and determines whether the current Type encompasses or refers to another type; that is, whether the current Type is an array, a pointer, or is passed by reference.
- InvokeMember(String, BindingFlags, Binder, Object, Object[]) - Invokes the specified member, using the specified binding constraints and matching the specified argument list.
- InvokeMember(String, BindingFlags, Binder, Object, Object[], CultureInfo) - Invokes the specified member, using the specified binding constraints and matching the specified argument list and culture.
- InvokeMember(String, BindingFlags, Binder, Object, Object[], ParameterModifier[], CultureInfo, String[]) - When overridden in a derived class, invokes the specified member, using the specified binding constraints and matching the specified argument list, modifiers and culture.
- IsArrayImpl - When overridden in a derived class, implements the IsArray property and determines whether the Type is an array.
- IsAssignableFrom - Determines whether an instance of the current Type can be assigned from an instance of the specified Type.
- IsByRefImpl - When overridden in a derived class, implements the IsByRef property and determines whether the Type is passed by reference.
- IsCOMObjectImpl - When overridden in a derived class, implements the IsCOMObject property and determines whether the Type is a COM object.
- IsContextfulImpl - Implements the IsContextful property and determines whether the Type can be hosted in a context.
- IsDefined - When overridden in a derived class, indicates whether one or more attributes of the specified type or of its derived types is applied to this member. (Inherited from MemberInfo.)
- IsEnumDefined - Returns a value that indicates whether the specified value exists in the current enumeration type.
- IsEquivalentTo - Determines whether two COM types have the same identity and are eligible for type equivalence.
- IsInstanceOfType - Determines whether the specified object is an instance of the current Type.
- IsMarshalByRefImpl - Implements the IsMarshalByRef property and determines whether the Type is marshaled by reference.

Copyright ©2011 sheepsqueezers.com

Methods (continued)

- IsPointerImpl - When overridden in a derived class, implements the IsPointer property and determines whether the Type is a pointer.
- IsPrimitiveImpl - When overridden in a derived class, implements the IsPrimitive property and determines whether the Type is one of the primitive types.
- IsSubclassOf - Determines whether the class represented by the current Type derives from the class represented by the specified Type.
- IsValueTypeImpl - Implements the IsValueType property and determines whether the Type is a value type; that is, not a class or an interface.
- MakeArrayType() - Returns a Type object representing a one-dimensional array of the current type, with a lower bound of zero.
- MakeArrayType(Int32) - Returns a Type object representing an array of the current type, with the specified number of dimensions.
- MakeByRefType - Returns a Type object that represents the current type when passed as a ref parameter (ByRef parameter in Visual Basic).
- MakeGenericType - Substitutes the elements of an array of types for the type parameters of the current generic type definition and returns a Type object representing the resulting constructed type.
- MakePointerType - Returns a Type object that represents a pointer to the current type.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- ReflectionOnlyGetType - Gets the Type with the specified name, specifying whether to perform a case-sensitive search and whether to throw an exception if the type is not found. The type is loaded for reflection only, not for execution.
- ToString - Returns a String representing the name of the current Type. (Overrides Object.ToString().)

Here is a simple example:

```
using System;

class MainProgram {

 enum MyEnum {Fred=1, Wilma, Pebbles, Dino, Barney, Betty, BammBamm, Hoppy};

  public static void Main() {

   Type tMyEnumType = typeof(MyEnum);
   String[] asEnumNames = tMyEnumType.GetEnumNames();
   foreach(String sEnumName in asEnumNames) {
    Console.WriteLine("Character={0}",sEnumName);
   }

  }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

- → Classes
  - → Classes Not Derived from the `Exception` and `Attribute` Classes
    - → `UriTypeConverter`

Copyright ©2011 sheepsqueezers.com

UriTypeConverter

The `UriTypeConverter` class converts a `String` type to a `Uri` type, and vice versa. This is a strange class…see Microsoft's website for more on this class. Also, see the `TypeConverter` as well as the `TypeDescriptor` class within the `System.ComponentModel` namespace. One useful example I've seen is to convert enum names to enum values and vice versa.

```
using System;
using System.ComponentModel;

class MainProgram {

 enum MyEnum {Fred=1, Wilma, Pebbles, Dino, Barney, Betty, BammBamm, Hoppy};

 public static void Main() {

  //Write out the enum MyEnum.Fred as text.
  MyEnum eFred = MyEnum.Fred;
  Console.WriteLine(TypeDescriptor.GetConverter(eFred).ConvertToString(eFred)); //Fred

  //Given the text, return the enum's value.
  Int32 eWilma = (Int32)TypeDescriptor.GetConverter(typeof(MyEnum)).ConvertFromString("Wilma");
  Console.WriteLine(eWilma); //2

 }

}
```

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `ValueType`

Copyright ©2011 sheepsqueezers.com

ValueType

The `ValueType` class provides the base class for value types. According to Microsoft's website: *ValueType overrides the virtual methods from Object with more appropriate implementations for value types. See also Enum, which inherits from ValueType.*

Data types are separated into value types and reference types. Value types are either stack-allocated or allocated inline in a structure. Reference types are heap-allocated. Both reference and value types are derived from the ultimate base class Object. In cases where it is necessary for a value type to behave like an object, a wrapper that makes the value type look like a reference object is allocated on the heap, and the value type's value is copied into it. The wrapper is marked so the system knows that it contains a value type. This process is known as boxing, and the reverse process is known as unboxing. Boxing and unboxing allow any type to be treated as an object.

Aside from serving as the base class for value types in the .NET Framework, the ValueType structure is generally not used directly in code. However, it can be used as a parameter in method calls to restrict possible arguments to value types instead of all objects, or to permit a method to handle a number of different value types.

See the description and examples at http://msdn.microsoft.com/en-us/library/system.valuetype.aspx.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `Version`

Copyright ©2011 sheepsqueezers.com

Version

The `Version` class represents the version number of an assembly, operating system, or the common language runtime. According to Microsoft's website: *Version numbers consist of two to four components: major, minor, build, and revision. The major and minor components are required; the build and revision components are optional, but the build component is required if the revision component is defined. All defined components must be integers greater than or equal to 0. The format of the version number is as follows (optional components are shown in square brackets ([and]):*

major.minor[.build[.revision]]

The components are used by convention as follows:

Major: Assemblies with the same name but different major versions are not interchangeable. A higher version number might indicate a major rewrite of a product where backward compatibility cannot be assumed.

Minor: If the name and major version number on two assemblies are the same, but the minor version number is different, this indicates significant enhancement with the intention of backward compatibility. This higher minor version number might indicate a point release of a product or a fully backward-compatible new version of a product.

…continued on next slide…

Copyright ©2011 sheepsqueezers.com

Version

Build: A difference in build number represents a recompilation of the same source. Different build numbers might be used when the processor, platform, or compiler changes.

Revision: Assemblies with the same name, major, and minor version numbers but different revisions are intended to be fully interchangeable. A higher revision number might be used in a build that fixes a security hole in a previously released assembly.

Ordinarily, the Version class is not used to assign a version number to an assembly. Instead, the AssemblyVersionAttribute class is used to define an assembly's version: [assembly:AssemblyVersionAttribute("2.0.1")].

You can find more on this class at: http://msdn.microsoft.com/en-us/library/system.version.aspx.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Classes

→ Classes Not Derived from the `Exception` and `Attribute` Classes

→ `WeakReference`

Copyright ©2011 sheepsqueezers.com

WeakReference

The `WeakReference` class represents a weak reference, which references an object while still allowing that object to be reclaimed by garbage collection. According to Microsoft's website: *A weak reference allows the garbage collector to collect an object while still allowing an application to access the object. If you need the object, you can still obtain a strong reference to it and prevent it from being collected.*

Constructors

- WeakReference(Object) - Initializes a new instance of the WeakReference class, referencing the specified object.
- WeakReference(Object, Boolean) - Initializes a new instance of the WeakReference class, referencing the specified object and using the specified resurrection tracking.
- WeakReference(SerializationInfo, StreamingContext) - Initializes a new instance of the WeakReference class, using deserialized data from the specified serialization and stream objects.

Properties

- IsAlive - Gets an indication whether the object referenced by the current WeakReference object has been garbage collected.
- Target - Gets or sets the object (the target) referenced by the current WeakReference object.
- TrackResurrection - Gets an indication whether the object referenced by the current WeakReference object is tracked after it is finalized.

Methods

- Equals(Object) - Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
- Finalize - Discards the reference to the target represented by the current WeakReference object. (Overrides Object.Finalize().)
- GetHashCode - Serves as a hash function for a particular type. (Inherited from Object.)
- GetObjectData - Populates a SerializationInfo object with all the data needed to serialize the current WeakReference object.
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- ToString - Returns a string that represents the current object. (Inherited from Object.)

You can find more here: http://msdn.microsoft.com/en-us/library/system.weakreference.aspx.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Structures

Copyright ©2011 sheepsqueezers.com

Structures

The structures of the `System` namespace are listed below.

Structures

- ArgIterator - Represents a variable-length argument list; that is, the parameters of a function that takes a variable number of arguments.
- ArraySegment<T> - Delimits a section of a one-dimensional array.
- Boolean - Represents a Boolean value.
- Byte - Represents an 8-bit unsigned integer.
- Char - Represents a Unicode character.
- ConsoleKeyInfo - Describes the console key that was pressed, including the character represented by the console key and the state of the SHIFT, ALT, and CTRL modifier keys.
- DateTime - Represents an instant in time, typically expressed as a date and time of day.
- DateTimeOffset - Represents a point in time, typically expressed as a date and time of day, relative to Coordinated Universal Time (UTC).
- Decimal - Represents a decimal number.
- Double - Represents a double-precision floating-point number.
- Guid - Represents a globally unique identifier (GUID).
- Int16 - Represents a 16-bit signed integer.
- Int32 - Represents a 32-bit signed integer.
- Int64 - Represents a 64-bit signed integer.
- IntPtr - A platform-specific type that is used to represent a pointer or a handle.
- ModuleHandle - Represents a runtime handle for a module.
- Nullable<T> - Represents an object whose underlying type is a value type that can also be assigned null like a reference type.
- RuntimeArgumentHandle - References a variable-length argument list.
- RuntimeFieldHandle - Represents a field using an internal metadata token.
- RuntimeMethodHandle - RuntimeMethodHandle is a handle to the internal metadata representation of a method.
- RuntimeTypeHandle - Represents a type using an internal metadata token.
- SByte - Represents an 8-bit signed integer.
- Single - Represents a single-precision floating-point number.
- TimeSpan - Represents a time interval.
- TimeZoneInfo.TransitionTime - Provides information about a specific time change, such as the change from daylight saving time to standard time or vice versa, in a particular time zone.
- TypedReference - Describes objects that contain both a managed pointer to a location and a runtime representation of the type that may be stored at that location.
- UInt16 - Represents a 16-bit unsigned integer.
- UInt32 - Represents a 32-bit unsigned integer.
- UInt64 - Represents a 64-bit unsigned integer.
- UIntPtr - A platform-specific type that is used to represent a pointer or a handle.
- Void - Specifies a return value type for a method that does not return a value.

Copyright ©2011 sheepsqueezers.com

Structures

It may seem strange that, say, `Int32` is represented as a structure. Why isn't it just a…number? Well, the `Int32` structure contains methods and fields. We won't list all of these out for every structure, but will show you those for Int32 below.

Fields

- MaxValue - Represents the largest possible value of an Int32. This field is constant.
- MinValue - Represents the smallest possible value of Int32. This field is constant.

Methods

- CompareTo(Int32) - Compares this instance to a specified 32-bit signed integer and returns an indication of their relative values.
- CompareTo(Object) - Compares this instance to a specified object and returns an indication of their relative values.
- Equals(Int32) - Returns a value indicating whether this instance is equal to a specified Int32 value.
- Equals(Object) - Returns a value indicating whether this instance is equal to a specified object. (Overrides ValueType.Equals(Object).)
- Finalize - Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
- GetHashCode - Returns the hash code for this instance. (Overrides ValueType.GetHashCode().)
- GetType - Gets the Type of the current instance. (Inherited from Object.)
- GetTypeCode - Returns the TypeCode for value type Int32.
- MemberwiseClone - Creates a shallow copy of the current Object. (Inherited from Object.)
- Parse(String) - Converts the string representation of a number to its 32-bit signed integer equivalent.
- Parse(String, NumberStyles) - Converts the string representation of a number in a specified style to its 32-bit signed integer equivalent.
- Parse(String, IFormatProvider) - Converts the string representation of a number in a specified culture-specific format to its 32-bit signed integer equivalent.
- Parse(String, NumberStyles, IFormatProvider) - Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent.
- ToString() - Converts the numeric value of this instance to its equivalent string representation. (Overrides ValueType.ToString().)
- ToString(IFormatProvider) - Converts the numeric value of this instance to its equivalent string representation using the specified culture-specific format information.
- ToString(String) - Converts the numeric value of this instance to its equivalent string representation, using the specified format.
- ToString(String, IFormatProvider) - Converts the numeric value of this instance to its equivalent string representation using the specified format and culture-specific format information.
- TryParse(String, Int32) - Converts the string representation of a number to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.
- TryParse(String, NumberStyles, IFormatProvider, Int32) - Converts the string representation of a number in a specified style and culture-specific format to its 32-bit signed integer equivalent. A return value indicates whether the conversion succeeded.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Interfaces

Copyright ©2011 sheepsqueezers.com

Interfaces

The interfaces of the `System` namespace are listed below.

Interfaces

- _AppDomain - Exposes the public members of the System.AppDomain class to unmanaged code.
- IAppDomainSetup - Represents assembly binding information that can be added to an instance of AppDomain.
- IAsyncResult - Represents the status of an asynchronous operation.
- ICloneable - Supports cloning, which creates a new instance of a class with the same value as an existing instance.
- IComparable - Defines a generalized type-specific comparison method that a value type or class implements to order or sort its instances.
- IComparable<T> - Defines a generalized comparison method that a value type or class implements to create a type-specific comparison method for ordering instances.
- IConvertible - Defines methods that convert the value of the implementing reference or value type to a common language runtime type that has an equivalent value.
- ICustomFormatter - Defines a method that supports custom formatting of the value of an object.
- IDisposable - Defines a method to release allocated resources.
- IEquatable<T> - Defines a generalized method that a value type or class implements to create a type-specific method for determining equality of instances.
- IFormatProvider - Provides a mechanism for retrieving an object to control formatting.
- IFormattable - Provides functionality to format the value of an object into a string representation.
- IObservable<T> - Defines a provider for push-based notification.
- IObserver<T> - Provides a mechanism for receiving push-based notifications.
- IServiceProvider - Defines a mechanism for retrieving a service object; that is, an object that provides custom support to other objects.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Delegates

Copyright ©2011 sheepsqueezers.com

Delegates

The delegates of the `System` namespace are listed below.

Delegates

- Action - Encapsulates a method that has no parameters and does not return a value.
- Action<T> - Encapsulates a method that has a single parameter and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10> - Encapsulates a method that has 10 parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11> - Encapsulates a method that has 11 parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12> - Encapsulates a method that has 12 parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13> - Encapsulates a method that has 13 parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14> - Encapsulates a method that has 14 parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15> - Encapsulates a method that has 15 parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16> - Encapsulates a method that has 16 parameters and does not return a value.
- Action<T1, T2> - Encapsulates a method that has two parameters and does not return a value.
- Action<T1, T2, T3> - Encapsulates a method that has three parameters and does not return a value.
- Action<T1, T2, T3, T4> - Encapsulates a method that has four parameters and does not return a value.
- Action<T1, T2, T3, T4, T5> - Encapsulates a method that has five parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6> - Encapsulates a method that has six parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7> - Encapsulates a method that has seven parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8> - Encapsulates a method that has eight parameters and does not return a value.
- Action<T1, T2, T3, T4, T5, T6, T7, T8, T9> - Encapsulates a method that has nine parameters and does not return a value.
- AppDomainInitializer - Represents the callback method to invoke when the application domain is initialized.
- AssemblyLoadEventHandler - Represents the method that handles the AssemblyLoad event of an AppDomain.
- AsyncCallback - References a method to be called when a corresponding asynchronous operation completes.
- Comparison<T> - Represents the method that compares two objects of the same type.
- ConsoleCancelEventHandler - Represents the method that will handle the CancelKeyPress event of a System.Console.
- Converter<TInput, TOutput> - Represents a method that converts an object from one type to another type.
- CrossAppDomainDelegate - Used by DoCallBack for cross-application domain calls.
- EventHandler - Represents the method that will handle an event that has no event data.
- EventHandler<TEventArgs> - Represents the method that will handle an event.

Copyright ©2011 sheepsqueezers.com

Delegates

Delegates (continued)

- Func<TResult> - Encapsulates a method that has no parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, TResult> - Encapsulates a method that has nine parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, TResult> - Encapsulates a method that has 10 parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, TResult> - Encapsulates a method that has 11 parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, TResult> - Encapsulates a method that has 12 parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, TResult> - Encapsulates a method that has 13 parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, TResult> - Encapsulates a method that has 14 parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, TResult> - Encapsulates a method that has 15 parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11, T12, T13, T14, T15, T16, TResult> - Encapsulates a method that has 16 parameters and returns a value of the type specified by the TResult parameter.
- Func<T, TResult> - Encapsulates a method that has one parameter and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, TResult> - Encapsulates a method that has two parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, TResult> - Encapsulates a method that has three parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, TResult> - Encapsulates a method that has four parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, TResult> - Encapsulates a method that has five parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, TResult> - Encapsulates a method that has six parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, TResult> - Encapsulates a method that has seven parameters and returns a value of the type specified by the TResult parameter.
- Func<T1, T2, T3, T4, T5, T6, T7, T8, TResult> - Encapsulates a method that has eight parameters and returns a value of the type specified by the TResult parameter.
- Predicate<T> - Represents the method that defines a set of criteria and determines whether the specified object meets those criteria.
- ResolveEventHandler - Represents a method that handles the AppDomain.TypeResolve, AppDomain.ResourceResolve, or AssemblyResolve event of an AppDomain.
- UnhandledExceptionEventHandler - Represents the method that will handle the event raised by an exception that is not handled by the application domain.

Copyright ©2011 sheepsqueezers.com

The `System` Namespace

→ Enumerations

Copyright ©2011 sheepsqueezers.com

Enumerations

The Enumerations of the `System` namespace are listed below.

Enumerations

- ActivationContext.ContextForm - Indicates the context for a manifest-activated application.
- AppDomainManagerInitializationOptions - Specifies the action that a custom application domain manager takes when initializing a new domain.
- AttributeTargets - Specifies the application elements on which it is valid to apply an attribute.
- Base64FormattingOptions - Specifies whether relevant Convert.ToBase64CharArray and Convert.ToBase64String methods insert line breaks in their output.
- ConsoleColor - Specifies constants that define foreground and background colors for the console.
- ConsoleKey - Specifies the standard keys on a console.
- ConsoleModifiers - Represents the SHIFT, ALT, and CTRL modifier keys on a keyboard.
- ConsoleSpecialKey - Specifies combinations of modifier and console keys that can interrupt the current process.
- DateTimeKind - Specifies whether a DateTime object represents a local time, a Coordinated Universal Time (UTC), or is not specified as either local time or UTC.
- DayOfWeek - Specifies the day of the week.
- Environment.SpecialFolder - Specifies enumerated constants used to retrieve directory paths to system special folders.
- Environment.SpecialFolderOption - Specifies options to use for getting the path to a special folder.
- EnvironmentVariableTarget - Specifies the location where an environment variable is stored or retrieved in a set or get operation.
- GCCollectionMode - Specifies the behavior for a forced garbage collection.
- GCNotificationStatus - Provides information about the current registration for notification of the next full garbage collection.
- GenericUriParserOptions - Specifies options for a UriParser.
- LoaderOptimization - An enumeration used with the LoaderOptimizationAttribute class to specify loader optimizations for an executable.
- MidpointRounding - Specifies how mathematical rounding methods should process a number that is midway between two numbers.
- PlatformID - Identifies the operating system, or platform, supported by an assembly.
- StringComparison - Specifies the culture, case, and sort rules to be used by certain overloads of the String.Compare and String.Equals methods.
- StringSplitOptions - Specifies whether applicable String.Split method overloads include or omit empty substrings from the return value.
- TypeCode - Specifies the type of an object.
- UriComponents - Specifies the parts of a Uri.
- UriFormat - Controls how URI information is escaped.
- UriHostNameType - Defines host name types for the Uri.CheckHostName method.
- UriIdnScope - Provides the possible values for the configuration setting of the System.Configuration.IdnElement in the System.Configuration namespace.
- UriKind - Defines the kinds of Uris for the Uri.IsWellFormedUriString(String, UriKind) and several Uri.Uri methods.
- UriPartial - Defines the parts of a URI for the Uri.GetLeftPart method.

Copyright ©2011 sheepsqueezers.com

What Next?

In *C# Programming IV-#*, we look at specific classes within specific namespaces such as the System namespace, the System.Data namespace, etc.

Copyright ©2011 sheepsqueezers.com

References

Click the book titles below to read more about these books on Amazon.com's website.

- Introducing Microsoft LINQ, Paolo Pialorsi and Marco Russo, Microsoft Press, ISBN:9780735623910
- LINQ Pocket Reference, Joseph Albahari and Ben Albahari, O'Reilly Press, ISBN:9780596519247
- Inside C#, Tom Archer and Andrew Whitechapel, Microsoft Press, ISBN:0735616485
- C# 4.0 In a Nutshell, O'Reilly Press, Joseph Albahari and Ben Albahari, ISBN:9780596800956
- The Object Primer, Scott W. Ambler, Cambridge Press, ISBN:0521540186
- CLR via C#, Jeffrey Richter, Microsoft Press, ISBN:9780735621633

Copyright ©2011 sheepsqueezers.com

Support sheepsqueezers.com

If you found this information helpful, please consider supporting sheepsqueezers.com. There are several ways to support our site:

- ☐ Buy me a cup of coffee by clicking on the following link and donate to my PayPal account: Buy Me A Cup Of Coffee?.
- ☐ Visit my Amazon.com Wish list at the following link and purchase an item: http://amzn.com/w/3OBK1K4EIWIR6

Please let me know if this document was useful by e-mailing me at `comments@sheepsqueezers.com`.

Copyright ©2011 sheepsqueezers.com