Programming in C#. Fundamentals

SIGMA Software

Lesson 6
Threads, strings and regular expressions

Threads, strings and regular expressions

Basics of threads
Operations with strings
Regular expression

Basics of threads

Programs can be split up into “threads” that can all run at the same time

1. The main thread is typically a User Interface thread
2. Without asynchronous programming, starting a long operation can freeze the main thread
3. With asynchronous programming the long thread can continue while the main thread is unaffected

```
public static void Main()
{
   Console.WriteLine("Application thread ID: {0}",
                     Thread.CurrentThread.ManagedThreadId);
   var t = Task.Run(() => {  Console.WriteLine("Task thread ID: {0}",
                                 Thread.CurrentThread.ManagedThreadId);
                          } );
   t.Wait();
}
```

Async/await

Wait for this to finish and then continue.

If you use await you must use async

```
public async void Work() {
    await SlowTask();
}

public void SlowTask() {
    int i;
    for (i = 0; i < 50; i++) {
        Console.WriteLine(i);
        for (int j = 0; j < 10000; j++) {
            var k = Math.Sqrt(j);
        }
    }
}
```

C# Strings

```
var name = "Frodo";
```

```
vendor.CompanyName = "ABC Corp";
```

```
var orderText = "Order from Acme, Inc" + Environment.NewLine +
                "Product: " + product.ProductCode + Environment.NewLine +
                "Quantity: " + quantity;
```

Strings Are Immutable

The contents of a string cannot be changed after the string is created

```
 var orderText = "Order from Acme, Inc" + Environment.NewLine +
                 "Product: " + product.ProductCode + Environment.NewLine +
                              "Quantity: " + quantity;
if (deliverBy.HasValue)
{
   orderText += System.Environment.NewLine +
                "Deliver By: " + deliverBy.Value.ToString("d");
}
```

A String Is a Reference Type …

```
var name = "Frodo";          //name = Frodo
var hobbit = name;           //name = Frodo  hobbit = Frodo
hobbit = "Bilbo";            //name = Frodo  hobbit = Bilbo
```

…That Acts Like a Value Type

.NET String Methods

```
var vendorInfo = "Vendor: ABC Corp.";

string result;
result = vendorInfo.ToLower();
result = vendorInfo.ToUpper();
result = vendorInfo.Replace("Vendor", "Supplier");

var length = vendorInfo.Length;
var index = vendorInfo.IndexOf(":");
var begins = vendorInfo.StartsWith("Vendor");
var ends = vendorInfo.EndsWith("Vendor");
```

Handling Nulls

```
string vendorInfo = null;

string result;
result = vendorInfo.ToLower();
```

```
if (!String.IsNullOrWhiteSpace(vendorInfo))
{
    result = vendorInfo.ToLower();
}
```

```
result = vendorInfo?.ToLower();
```

Verbatim String Literals

```
var orderText = "Product: Tools-1\r\nQuantity: 12\r\nInstructions: standard delivery";
```

```
Product: Tools-1
Quantity: 12
Instructions: standard delivery
```

```
var directions = "Insert \r\n to define a new line";
```

```
Insert
 to define a new line
```

```
var directions = @"Insert \r\n to define a new line";
```

```
Insert \r\n to define a new line
```

Verbatim String Literal Best Practices

Do:

Use verbatim string literals when the string contains special characters such as backslashes

Use verbatim string literals to hold folder or file names

```
@"c:\mydir\myfile.txt";
```

Use two quotes to include quotes in a verbatim string literal

```
@"Say it with a long ""a"" sound";
```

Avoid:

Using verbatim string literals when there is no reason

```
@"Frodo";
```

Formatting Strings

```
var p = product.Category + "-" + product.SequenceNumber;
```

```
var p = String.Format("{0}-{1}",
                      product.Category,
                      product.SequenceNumber);
```

String.Format Best Practices

Do:

Use String.Format to insert the value of an expression into a string

Better technique with C# 6

Include a formatting string as needed

```
String.Format("Deliver by: {0:d}",
              deliveryBy);
```

Avoid:

Using String.Format when concatenating string literals

```
String.Format("Hello {0}",
              "World");
```

String Interpolation

```
var pc = String.Format("{0}-{1}",
                       product.Category,
                       product.SequenceNumber);
```

```
var pc = $"{product.Category}-{product.SequenceNumber}";
```

Building Long Strings

```
var orderText = "Order from Acme, Inc" + Environment.NewLine +
       "Product: " + productIdentifier + Environment.NewLine +
       "Quantity: " + quantity;
if (deliverBy.HasValue)
{
   orderText += System.Environment.NewLine +
                "Deliver By: " + deliverBy.Value.ToString("d")
}
if (!String.IsNullOrWhiteSpace(instructions))
{
   orderText += System.Environment.NewLine +
                "Instructions: " + instructions;
}
```

StringBuilder

1. Conceptually a mutable string
2. Allows string operations, such as concatenation, without creating a new string
3. Provides methods for manipulating the mutable string
    - Append, Insert, Replace, etc
4. Use ToString to convert to a string
5. More efficient when working with strings that are
    - Built up with many separate concatenation operations
    - Changed a large number of times, such as within a loop

Long String Best Practices

Do:

Use StringBuilder when building up a string with numerous concatenation operations

Use StringBuilder when modifying a string numerous times
Such as in a loop

Consider readability

Avoid:

Using StringBuilder when only modifying a string a few times

What is Regular Expression?

- A regular expression (regex or regexp for short) is a special text string for describing a search pattern.
- A regular expression is a set of pattern matching rules encoded in a string according to certain syntax rules.
- The syntax (language format) described is compliant with extended regular expressions (EREs) defined in IEEE POSIX 1003.2

Sample Example for Email Id:

^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}$

Regex in .NET

1. Validation in ASP.NET, we use the **RegularExpressionValidator** control to validate that input Fields.

   Name, Email Id, Urls, Date

2. Also when certain patterns need to be replaced by a String.

   Regex.Replace(intput, @“Pattern1", “Pattern2”}

3. To reformatting an input string by re-arranging the order and placement of the elements within the input string

Regex Language Elements

Metacharacters

- The constructs within regular expressions that have special meaning are referred to as metacharacters
- Characters other than . $ ^ { [(|)] } * + ? \ match themselves.

Character Classes

- Character classes are a mini-language within regular expressions, defined by the enclosing hard braces **[ ]**.

  Examples: [a-z A-Z 0-9],

Regex Language Elements

MeataCharacters and their description followed by an example:

^ Start of a string. ^abc matches are abc, abcdefg, abc123,

$ End of a string. abc$ matches with abc, endsinabc, 123abc

. Any character (except \n newline)

| Alternation.

{...} Explicit quantifier notation.

[...] Explicit set of characters to match.

(...) Logical grouping of part of an expression.

* 0 or more of previous expression.

+ 1 or more of previous expression.

? 0 or 1 of previous expression; also forces minimal matching when an expression might match several strings within a search string.

\ Preceding one of the above, it makes it a literal instead of a special character. Preceding a special matching character, see below.

Regex Language Elements

\w Matches any word character. equivalent to [a-zA-Z_0-9].

\W Matches any nonword character. Equivalent to the Unicode categories equivalent to [^a-zA-Z_0-9].

\s Matches any white-space character. equivalent to [\f\n\r\t\v].

\S Matches any non-white-space character. equivalent to [^ \f\n\r\t\v].

\d Matches any decimal digit.

\D Matches any nondigit.

Examples

Pattern	Description
^\d{5}$	5 numeric digits, such as a US ZIP code.
^\d{3}-\d{2}-\d{4}$	Validates the format such as 111-11-1111 (Social Security Number)
^[01]?[- .]?(\([2-9]\d{2}\)|[2-9]\d{2})[- .]?\d{3}[- .]?\d{4}$	Validates a U.S. phone number.

Regular Expression API

Regular Expression classes found in the System.Text.RegularExpressions namespace.
Main classes we'll want to use are **Regex**, **Match**, and **MatchCollection.**

Regex: Methods and their description

IsMatch -	Returns true if the regex finds a match in the input string.
Match -	Returns a Match object if a match is found in the input string.
Matches -	Returns a MatchCollection object containing any and all
Replace -	Replaces matches in the input string with a given replacemen.

How to Test regex in .NET

```
Regex r = new Regex(pattern, RegexOptions.IgnoreCase | RegexOptions.IgnorePatternWhitespace);
Match m = r.Match(inputtext);
if (m.Success)
{
    Console.WriteLine("Matched String " + m.Group());
}
else
{
    Console.WriteLine("Not Matched ");
}
```

How to Test regex in .NET

```
string text = "One fish two fish red fish blue fish";
string pat = @"(?<1>\w+)\s+(?<2>fish)\s*";
// Compile the regular expression.
Regex r = new Regex(pat, RegexOptions.IgnoreCase);
// Match the regular expression pattern against a text string
Match m = r.Match(text);
while (m.Success)
{
    // Display the first match and its capture set.
    System.Console.WriteLine("Match=[" + m + "]");
    CaptureCollection cc = m.Captures;
    foreach (Capture c in cc)
    {
        System.Console.WriteLine("Capture=[" + c + "]");
    }
}
```

Regular Expression

- Regular expressions provide a very powerful way to describe patterns in text, making them an excellent resource for string validation and manipulation.
- The .NET Framework provides first-rate support for regular expressions in its System.Text.RegularExpressions namespace and specifically the Regex class found there.

References:

- http://msdn.microsoft.com/en-us/library/system.text.regularexpressions.groupcollection.aspx
- http://www.regular-expressions.info/dotnet.html
- http://regexlib.com/CheatSheet.aspx

Q & A

Practice
Lesson 6

Home work