

LinksPlatform's Platform.RegularExpressions.Transformer Class Library

1.1 ./csharp/Platform.RegularExpressions.Transformer/Context.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.RegularExpressions.Transformer
4  {
5      public class Context : IContext
6      {
7          public string Path { get; }
8
9          public Context(string path) => Path = path;
10     }
11 }
```

1.2 ./csharp/Platform.RegularExpressions.Transformer/IContext.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.RegularExpressions.Transformer
4  {
5      public interface IContext
6      {
7          public string Path { get; }
8      }
9  }
```

1.3 ./csharp/Platform.RegularExpressions.Transformer/ISubstitutionRule.cs

```
1  using System.Text.RegularExpressions;
2
3  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5  namespace Platform.RegularExpressions.Transformer
6  {
7      public interface ISubstitutionRule
8      {
9          Regex MatchPattern { get; }
10         string SubstitutionPattern { get; }
11         Regex PathPattern { get; }
12         int MaximumRepeatCount { get; }
13     }
14 }
```

1.4 ./csharp/Platform.RegularExpressions.Transformer/ITransformer.cs

```
1  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3  namespace Platform.RegularExpressions.Transformer
4  {
5      public interface ITransformer
6      {
7          string Transform(string source, IContext context);
8      }
9  }
```

1.5 ./csharp/Platform.RegularExpressions.Transformer/RegexExtensions.cs

```
1  using System;
2  using System.Text.RegularExpressions;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.RegularExpressions.Transformer
7  {
8      public static class RegexExtensions
9      {
10         public static Regex OverrideOptions(this Regex regex, RegexOptions options, TimeSpan
            ↳ matchTimeout)
11         {
12             if (regex == null)
13             {
14                 return null;
15             }
16             return new Regex(regex.ToString(), options, matchTimeout);
17         }
18     }
19 }
```

1.6 ./csharp/Platform.RegularExpressions.Transformer/SubstitutionRule.cs

```

1  using System;
2  using System.Text.RegularExpressions;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6  namespace Platform.RegularExpressions.Transformer
7  {
8      public class SubstitutionRule : ISubstitutionRule
9      {
10         public static readonly TimeSpan DefaultMatchTimeout = TimeSpan.FromMinutes(5);
11         public static readonly RegexOptions DefaultMatchPatternRegexOptions =
12             ↳ RegexOptions.Compiled | RegexOptions.Multiline;
13         public static readonly RegexOptions DefaultPathPatternRegexOptions =
14             ↳ RegexOptions.Compiled | RegexOptions.Singleline;
15
16         public Regex MatchPattern { get; set; }
17
18         public string SubstitutionPattern { get; set; }
19
20         public Regex PathPattern { get; set; }
21
22         public int MaximumRepeatCount { get; set; }
23
24         public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
25             ↳ pathPattern, int maximumRepeatCount, RegexOptions? matchPatternOptions,
26             ↳ RegexOptions? pathPatternOptions, TimeSpan? matchTimeout)
27         {
28             MatchPattern = matchPattern;
29             SubstitutionPattern = substitutionPattern;
30             PathPattern = pathPattern;
31             MaximumRepeatCount = maximumRepeatCount;
32             OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
33             ↳ matchTimeout ?? matchPattern.MatchTimeout);
34             OverridePathPatternOptions(pathPatternOptions ?? pathPattern.Options, matchTimeout
35             ↳ ?? pathPattern.MatchTimeout);
36         }
37
38         public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
39             ↳ pathPattern, int maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
40             ↳ substitutionPattern, pathPattern, maximumRepeatCount, useDefaultOptions ?
41             ↳ DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
42             ↳ DefaultPathPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
43             ↳ DefaultMatchTimeout : (TimeSpan?)null) { }
44
45         public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
46             ↳ pathPattern, int maximumRepeatCount) : this(matchPattern, substitutionPattern,
47             ↳ pathPattern, maximumRepeatCount, true) { }
48
49         public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
50             ↳ maximumRepeatCount) : this(matchPattern, substitutionPattern, null,
51             ↳ maximumRepeatCount) { }
52
53         public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
54             ↳ this(matchPattern, substitutionPattern, null, 0) { }
55
56         public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
57             ↳ SubstitutionRule(tuple.Item1, tuple.Item2);
58
59         public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
60             ↳ => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);
61
62         public static implicit operator SubstitutionRule(ValueTuple<Regex, string, Regex, int>
63             ↳ tuple) => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3, tuple.Item4);
64
65         public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
66             ↳ MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);
67
68         public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
69             ↳ PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
70     }
71 }

```

1.7 ./csharp/Platform.RegularExpressions.Transformer/Transformer.cs

```

1  using System.Collections.Generic;
2  using System.Linq;
3
4  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5

```

```

6 namespace Platform.RegularExpressions.Transformer
7 {
8     public class Transformer : ITransformer
9     {
10         private readonly IList<ISubstitutionRule> _substitutionRules;
11
12         public Transformer(IList<ISubstitutionRule> substitutionRules) => _substitutionRules =
            ↳ substitutionRules;
13
14         public string Transform(string source, IContext context)
15         {
16             var current = source;
17             for (var i = 0; i < _substitutionRules.Count; i++)
18             {
19                 var rule = _substitutionRules[i];
20                 var matchPattern = rule.MatchPattern;
21                 var substitutionPattern = rule.SubstitutionPattern;
22                 var pathPattern = rule.PathPattern;
23                 var maximumRepeatCount = rule.MaximumRepeatCount;
24                 if (pathPattern == null || pathPattern.IsMatch(context.Path))
25                 {
26                     var replaceCount = 0;
27                     do
28                     {
29                         current = matchPattern.Replace(current, substitutionPattern);
30                         if (++replaceCount > maximumRepeatCount)
31                         {
32                             break;
33                         }
34                     }
35                     while (matchPattern.IsMatch(current));
36                 }
37             }
38             return current;
39         }
40
41         public IList<ITransformer> GenerateTransformersForEachRulesStep()
42         {
43             var transformers = new List<ITransformer>();
44             for (int i = 1; i <= _substitutionRules.Count; i++)
45             {
46                 transformers.Add(new Transformer(_substitutionRules.Take(i).ToList()));
47             }
48             return transformers;
49         }
50     }
51 }

```

1.8 ./csharp/Platform.RegularExpressions.Transformer/TransformerCLI.cs

```

1 using System.Diagnostics;
2 using System.IO;
3 using System.Text;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.RegularExpressions.Transformer
8 {
9     public class TransformerCLI
10     {
11         private readonly ITransformer _transformer;
12
13         public TransformerCLI(ITransformer transformer) => _transformer = transformer;
14
15         public bool Run(string[] args, out string message)
16         {
17             message = "";
18             var sourcePath = GetArgOrDefault(args, 0);
19             if (!File.Exists(sourcePath))
20             {
21                 message = $"{sourcePath} file does not exist.";
22                 return false;
23             }
24             var targetPath = GetArgOrDefault(args, 1);
25             if (string.IsNullOrEmpty(targetPath))
26             {
27                 targetPath = ChangeToTargetExtension(sourcePath);
28             }
29             else if (Directory.Exists(targetPath) &&
30                 ↳ File.GetAttributes(targetPath).HasFlag(FileAttributes.Directory))

```

```

31         targetPath = Path.Combine(targetPath, GetTargetFileName(sourcePath));
32     }
33     else if (LooksLikeDirectoryPath(targetPath))
34     {
35         Directory.CreateDirectory(targetPath);
36         targetPath = Path.Combine(targetPath, GetTargetFileName(sourcePath));
37     }
38     if (File.Exists(targetPath))
39     {
40         var applicationPath = Process.GetCurrentProcess().MainModule.FileName;
41         var targetFileLastUpdateDateTime = new FileInfo(targetPath).LastWriteTimeUtc;
42         if (new FileInfo(sourcePath).LastWriteTimeUtc < targetFileLastUpdateDateTime &&
43             ↪ new FileInfo(applicationPath).LastWriteTimeUtc <
44             ↪ targetFileLastUpdateDateTime)
45         {
46             return true;
47         }
48     }
49     File.WriteAllText(targetPath, _transformer.Transform(File.ReadAllText(sourcePath,
50         ↪ Encoding.UTF8), new Context(sourcePath)), Encoding.UTF8);
51     message = $"{targetPath} file written.";
52     return true;
53 }
54
55 private static string GetTargetFileName(string sourcePath) =>
56     ↪ ChangeToTargetExtension(Path.GetFileName(sourcePath));
57
58 private static string ChangeToTargetExtension(string path) => Path.ChangeExtension(path,
59     ↪ ".cpp");
60
61 private static bool LooksLikeDirectoryPath(string targetPath) =>
62     ↪ targetPath.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
63     ↪ targetPath.EndsWith(Path.AltDirectorySeparatorChar.ToString());
64
65 private static string GetArgOrDefault(string[] args, int index) => args.Length > index ?
66     ↪ args[index] : null;
67 }
68 }

```

1.9 ./csharp/Platform.RegularExpressions.Transformer/TransformerExtensions.cs

```

1 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
2
3 namespace Platform.RegularExpressions.Transformer
4 {
5     public static class TransformerExtensions
6     {
7         public static void DebugOutput(this Transformer transformer, string sourcePath, string
8             ↪ targetFilename, string targetExtension) =>
9             ↪ transformer.GenerateTransformersForEachRulesStep().TransformAllToFiles(sourcePath,
10             ↪ targetFilename, targetExtension);
11     }
12 }

```

1.10 ./csharp/Platform.RegularExpressions.Transformer/TransformersListExtensions.cs

```

1 using System.IO;
2 using System.Collections.Generic;
3 using System.Text;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.RegularExpressions.Transformer
8 {
9     public static class TransformersListExtensions
10     {
11         public static void TransformAllToFiles(this IList<ITransformer> transformers, string
12             ↪ sourcePath, string targetFilename, string targetExtension)
13         {
14             if (transformers.Count > 0)
15             {
16                 var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
17                 var transformerContext = new Context(sourcePath);
18                 for (int i = 0; i < transformers.Count; i++)
19                 {
20                     var transformationOutput = transformers[i].Transform(sourceText,
21                         ↪ transformerContext);
22                     File.WriteAllText($"{targetFilename}.{i}{targetExtension}",
23                         ↪ transformationOutput, Encoding.UTF8);
24                 }
25             }
26         }
27     }
28 }

```

```

21     }
22 }
23 }
24 }
25 }

```

1.11 ./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs

```

1 using System.Text.RegularExpressions;
2 using Xunit;
3
4 namespace Platform.RegularExpressions.Transformer.Tests
5 {
6     public class SubstitutionRuleTests
7     {
8         [Fact]
9         public void OptionsOverrideTest()
10        {
11            SubstitutionRule rule = (new Regex(@"^\.s*?\#pragma[\.sa-zA-Z0-9\./]+\$"), "", null, 0);
12            Assert.Equal(RegexOptions.Compiled | RegexOptions.Multiline,
13                ↪ rule.MatchPattern.Options);
14        }
15    }

```

1.12 ./csharp/Platform.RegularExpressions.Transformer.Tests/TransformersTests.cs

```

1 using System.IO;
2 using System.Text;
3 using System.Text.RegularExpressions;
4 using Xunit;
5
6 namespace Platform.RegularExpressions.Transformer.Tests
7 {
8     public class TransformersTests
9     {
10        [Fact]
11        public void DebugOutputTest()
12        {
13            var rule1 = (new Regex("a"), "b");
14            var rule2 = (new Regex("b"), "c");
15
16            var sourceText = "aaaa";
17            var firstStepReferenceText = "bbbb";
18            var secondStepReferenceText = "cccc";
19
20            var sourceFilename = Path.GetTempFileName();
21            File.WriteAllText(sourceFilename, sourceText, Encoding.UTF8);
22
23            var transformer = new Transformer(new SubstitutionRule[] { rule1, rule2 });
24
25            var targetFilename = Path.GetTempFileName();
26
27            transformer.DebugOutput(sourceFilename, targetFilename, ".txt");
28
29            var firstStepReferenceFilename = $"{targetFilename}.0.txt";
30            var secondStepReferenceFilename = $"{targetFilename}.1.txt";
31
32            Assert.True(File.Exists(firstStepReferenceFilename));
33            Assert.True(File.Exists(secondStepReferenceFilename));
34
35            Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
36                ↪ Encoding.UTF8));
37            Assert.Equal(secondStepReferenceText, File.ReadAllText(secondStepReferenceFilename,
38                ↪ Encoding.UTF8));
39
40            File.Delete(sourceFilename);
41            File.Delete(firstStepReferenceFilename);
42            File.Delete(secondStepReferenceFilename);
43        }
44    }

```

Index

- ./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs, 5
- ./csharp/Platform.RegularExpressions.Transformer.Tests/TransformersTests.cs, 5
- ./csharp/Platform.RegularExpressions.Transformer/Context.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/IContext.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/ISubstitutionRule.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/ITransformer.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/RegexExtensions.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/SubstitutionRule.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/Transformer.cs, 2
- ./csharp/Platform.RegularExpressions.Transformer/TransformerCLI.cs, 3
- ./csharp/Platform.RegularExpressions.Transformer/TransformerExtensions.cs, 4
- ./csharp/Platform.RegularExpressions.Transformer/TransformersListExtensions.cs, 4