

## 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.Runtime.CompilerServices;
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 interface ISubstitutionRule
9      {
10         Regex MatchPattern
11         {
12             [MethodImpl(MethodImplOptions.AggressiveInlining)]
13             get;
14         }
15
16         string SubstitutionPattern
17         {
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             get;
20         }
21
22         Regex PathPattern
23         {
24             [MethodImpl(MethodImplOptions.AggressiveInlining)]
25             get;
26         }
27
28         int MaximumRepeatCount
29         {
30             [MethodImpl(MethodImplOptions.AggressiveInlining)]
31             get;
32         }
33     }
34 }
```

### 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;
3 using System.Text.RegularExpressions;
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 SubstitutionRule : ISubstitutionRule
10    {
11        public static readonly TimeSpan DefaultMatchTimeout = TimeSpan.FromMinutes(5);
12        public static readonly RegexOptions DefaultMatchPatternRegexOptions =
            ↳ RegexOptions.Compiled | RegexOptions.Multiline;
13        public static readonly RegexOptions DefaultPathPatternRegexOptions =
            ↳ RegexOptions.Compiled | RegexOptions.Singleline;
14
15        public Regex MatchPattern { get; set; }
16
17        public string SubstitutionPattern { get; set; }
18
19        public Regex PathPattern { get; set; }
20
21        public int MaximumRepeatCount { get; set; }
22
23        public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
            ↳ pathPattern, int maximumRepeatCount, RegexOptions? matchPatternOptions,
            ↳ RegexOptions? pathPatternOptions, TimeSpan? matchTimeout)
24        {
25            MatchPattern = matchPattern;
26            SubstitutionPattern = substitutionPattern;
27            PathPattern = pathPattern;
28            MaximumRepeatCount = maximumRepeatCount;
29            OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
            ↳ matchTimeout ?? matchPattern.MatchTimeout);
30            OverridePathPatternOptions(pathPatternOptions ?? pathPattern.Options, matchTimeout
            ↳ ?? pathPattern.MatchTimeout);
31        }
32
33        public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
            ↳ pathPattern, int maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
            ↳ substitutionPattern, pathPattern, maximumRepeatCount, useDefaultOptions ?
            ↳ DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
            ↳ DefaultPathPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
            ↳ DefaultMatchTimeout : (TimeSpan?)null) { }
34
35        public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
            ↳ pathPattern, int maximumRepeatCount) : this(matchPattern, substitutionPattern,
            ↳ pathPattern, maximumRepeatCount, true) { }
36
37        public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
            ↳ maximumRepeatCount) : this(matchPattern, substitutionPattern, null,
            ↳ maximumRepeatCount) { }
38
39        public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
            ↳ this(matchPattern, substitutionPattern, null, 0) { }
40
41        public static implicit operator SubstitutionRule(ValueTuple<string, string> tuple) =>
            ↳ new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2);
42
43        public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
            ↳ SubstitutionRule(tuple.Item1, tuple.Item2);
44

```

```

45     public static implicit operator SubstitutionRule(ValueTuple<string, string, int> tuple)
46         => new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2, tuple.Item3);
47
48     public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
49         => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);
50
51     public static implicit operator SubstitutionRule(ValueTuple<string, string, Regex, int>
52         tuple) => new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2, tuple.Item3,
53         tuple.Item4);
54
55     public static implicit operator SubstitutionRule(ValueTuple<Regex, string, Regex, int>
56         tuple) => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3, tuple.Item4);
57
58     public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
59         MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);
60
61     public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
62         PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
63
64     public override string ToString()
65     {
66         var sb = new StringBuilder();
67         sb.Append(' ');
68         sb.Append(MatchPattern.ToString());
69         sb.Append(' ');
70         sb.Append(" -> ");
71         sb.Append(' ');
72         sb.Append(SubstitutionPattern);
73         sb.Append(' ');
74         if (PathPattern != null)
75         {
76             sb.Append(" on files ");
77             sb.Append(' ');
78             sb.Append(PathPattern.ToString());
79             sb.Append(' ');
80         }
81         if (MaximumRepeatCount > 0)
82         {
83             if (MaximumRepeatCount >= int.MaxValue)
84             {
85                 sb.Append(" repeated forever");
86             }
87             else
88             {
89                 sb.Append(" repeated up to ");
90                 sb.Append(MaximumRepeatCount);
91                 sb.Append(" times");
92             }
93         }
94         return sb.ToString();
95     }
96 }

```

## 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 =
13             substitutionRules;
14
15         public string Transform(string source, IContext context)
16         {
17             var current = source;
18             var currentFilePath = context?.Path ?? "";
19             for (var i = 0; i < _substitutionRules.Count; i++)
20             {
21                 var rule = _substitutionRules[i];
22                 var matchPattern = rule.MatchPattern;
23                 var substitutionPattern = rule.SubstitutionPattern;
24                 var pathPattern = rule.PathPattern;

```

```

24     var maximumRepeatCount = rule.MaximumRepeatCount;
25     if (pathPattern == null || pathPattern.IsMatch(currentFilePath))
26     {
27         var replaceCount = 0;
28         do
29         {
30             current = matchPattern.Replace(current, substitutionPattern);
31             replaceCount++;
32             if (maximumRepeatCount < int.MaxValue && replaceCount >
33                 → maximumRepeatCount)
34             {
35                 break;
36             }
37             while (matchPattern.IsMatch(current));
38         }
39     }
40     return current;
41 }
42
43 public IList<ITransformer> GenerateTransformersForEachRulesStep()
44 {
45     var transformers = new List<ITransformer>();
46     for (int i = 1; i <= _substitutionRules.Count; i++)
47     {
48         transformers.Add(new Transformer(_substitutionRules.Take(i).ToList()));
49     }
50     return transformers;
51 }
52 }
53 }

```

## 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             {
32                 targetPath = Path.Combine(targetPath, GetTargetFileName(sourcePath));
33             }
34             else if (LooksLikeDirectoryPath(targetPath))
35             {
36                 Directory.CreateDirectory(targetPath);
37                 targetPath = Path.Combine(targetPath, GetTargetFileName(sourcePath));
38             }
39             if (File.Exists(targetPath))
40             {
41                 var applicationPath = Process.GetCurrentProcess().MainModule.FileName;
42                 var targetFileLastUpdateDateTime = new FileInfo(targetPath).LastWriteTimeUtc;
43                 if (new FileInfo(sourcePath).LastWriteTimeUtc < targetFileLastUpdateDateTime &&
44                     → new FileInfo(applicationPath).LastWriteTimeUtc <
45                     → targetFileLastUpdateDateTime)
46                 {

```

```

44         return true;
45     }
46 }
47 File.WriteAllText(targetPath, _transformer.Transform(File.ReadAllText(sourcePath,
48     ↪ Encoding.UTF8), new Context(sourcePath), Encoding.UTF8));
49 message = $"{targetPath} file written.";
50 return true;
51 }
52 private static string GetTargetFileName(string sourcePath) =>
53     ↪ ChangeToTargetExtension(Path.GetFileName(sourcePath));
54 private static string ChangeToTargetExtension(string path) => Path.ChangeExtension(path,
55     ↪ ".cpp");
56 private static bool LooksLikeDirectoryPath(string targetPath) =>
57     ↪ targetPath.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
58     ↪ targetPath.EndsWith(Path.AltDirectorySeparatorChar.ToString());
59 private static string GetArgOrDefault(string[] args, int index) => args.Length > index ?
60     ↪ args[index] : null;
61 }

```

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

```

1 using System.Collections.Generic;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.RegularExpressions.Transformer
6 {
7     public static class TransformerExtensions
8     {
9         public static List<string> GetSteps(this Transformer transformer, string source) =>
10             ↪ transformer.GenerateTransformersForEachRulesStep().TransformWithAll(source);
11
12         public static void WriteStepsToFiles(this Transformer transformer, string sourcePath,
13             ↪ string targetFilename, string targetExtension, bool skipFilesWithNoChanges) => trans
14             ↪ former.GenerateTransformersForEachRulesStep().TransformWithAllToFiles(sourcePath,
15             ↪ targetFilename, targetExtension, skipFilesWithNoChanges);
16     }
17 }

```

## 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 List<string> TransformWithAll(this IList<ITransformer> transformers,
12             ↪ string source)
13         {
14             var strings = new List<string>();
15             if (transformers.Count > 0)
16             {
17                 for (int i = 0; i < transformers.Count; i++)
18                 {
19                     strings.Add(transformers[i].Transform(source, null));
20                 }
21             }
22             return strings;
23         }
24
25         public static void TransformWithAllToFiles(this IList<ITransformer> transformers, string
26             ↪ sourcePath, string targetFilename, string targetExtension, bool
27             ↪ skipFilesWithNoChanges)
28         {
29             if (transformers.Count > 0)
30             {
31                 var lastText = "";
32                 var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
33                 var transformerContext = new Context(sourcePath);
34                 for (int i = 0; i < transformers.Count; i++)

```

```

32     {
33         var transformationOutput = transformers[i].Transform(sourceText,
34             ↪ transformerContext);
35         if (!skipFilesWithNoChanges && string.Equals(lastText,
36             ↪ transformationOutput))
37         {
38             lastText = transformationOutput;
39             File.WriteAllText($"{targetFilename}.{i}-{targetExtension}",
40                 ↪ transformationOutput, Encoding.UTF8);
41         }
42     }
43 }

```

### 1.11 ./csharp/Platform.RegularExpressions.Transformer.Tests/MarkovAlgorithmsTests.cs

```

1 using System.Text.RegularExpressions;
2 using Xunit;
3
4 namespace Platform.RegularExpressions.Transformer.Tests
5 {
6     public class MarkovAlgorithmsTests
7     {
8         /// <remarks>
9         /// Example is from https://en.wikipedia.org/wiki/Markov_algorithm.
10        /// </remarks>
11        [Fact]
12        public void BinaryToUnaryNumbersTest()
13        {
14            var rules = new SubstitutionRule[]
15            {
16                ("1", "0|", int.MaxValue), // "1" -> "0|" repeated forever
17                // | symbol should be escaped for regular expression pattern, but not in the
18                ↪ substitution pattern
19                ("0", "0||", int.MaxValue), // "0" -> "0||" repeated forever
20                ("0", "", int.MaxValue), // "0" -> "" repeated forever
21            };
22            var transformer = new Transformer(rules);
23            var input = "101";
24            var expectedOutput = "|||||";
25            var output = transformer.Transform(input, null);
26            Assert.Equal(expectedOutput, output);
27        }
28    }

```

### 1.12 ./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    }

```

### 1.13 ./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 sourceText = "aaaa";
14     var firstStepReferenceText = "bbbb";
15     var secondStepReferenceText = "cccc";
16
17     var transformer = new Transformer(new SubstitutionRule[] {
18         (new Regex("a"), "b"),
19         (new Regex("b"), "c")
20     });
21
22     var steps = transformer.GetSteps(sourceText);
23
24     Assert.Equal(2, steps.Count);
25     Assert.Equal(firstStepReferenceText, steps[0]);
26     Assert.Equal(secondStepReferenceText, steps[1]);
27 }
28
29 [Fact]
30 public void DebugFilesOutputTest()
31 {
32     var sourceText = "aaaa";
33     var firstStepReferenceText = "bbbb";
34     var secondStepReferenceText = "cccc";
35
36     var sourceFilename = Path.GetTempFileName();
37     File.WriteAllText(sourceFilename, sourceText, Encoding.UTF8);
38
39     var transformer = new Transformer(new SubstitutionRule[] {
40         (new Regex("a"), "b"),
41         (new Regex("b"), "c")
42     });
43
44     var targetFilename = Path.GetTempFileName();
45
46     transformer.WriteStepsToFiles(sourceFilename, targetFilename, ".txt",
47         ↪ skipFilesWithNoChanges: false);
48
49     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
50     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
51
52     Assert.True(File.Exists(firstStepReferenceFilename));
53     Assert.True(File.Exists(secondStepReferenceFilename));
54
55     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
56         ↪ Encoding.UTF8));
57     Assert.Equal(secondStepReferenceText, File.ReadAllText(secondStepReferenceFilename,
58         ↪ Encoding.UTF8));
59
60     File.Delete(sourceFilename);
61     File.Delete(firstStepReferenceFilename);
62     File.Delete(secondStepReferenceFilename);
63 }
64
65 [Fact]
66 public void FilesWithNoChangesSkippedTest()
67 {
68     var sourceText = "aaaa";
69     var firstStepReferenceText = "bbbb";
70     var thirdStepReferenceText = "cccc";
71
72     var sourceFilename = Path.GetTempFileName();
73     File.WriteAllText(sourceFilename, sourceText, Encoding.UTF8);
74
75     var transformer = new Transformer(new SubstitutionRule[] {
76         (new Regex("a"), "b"),
77         (new Regex("x"), "y"),
78         (new Regex("b"), "c")
79     });
80
81     var targetFilename = Path.GetTempFileName();
82
83     transformer.WriteStepsToFiles(sourceFilename, targetFilename, ".txt",
84         ↪ skipFilesWithNoChanges: true);
85
86     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
87     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
88     var thirdStepReferenceFilename = $"{targetFilename}.2.txt";
89
90     Assert.True(File.Exists(firstStepReferenceFilename));
91     Assert.False(File.Exists(secondStepReferenceFilename));

```

```
88     Assert.True(File.Exists(thirdStepReferenceFilename));
89
90     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
91         ↪ Encoding.UTF8));
92     Assert.Equal(thirdStepReferenceText, File.ReadAllText(thirdStepReferenceFilename,
93         ↪ Encoding.UTF8));
94
95     File.Delete(sourceFilename);
96     File.Delete(firstStepReferenceFilename);
97     File.Delete(secondStepReferenceFilename);
98     File.Delete(thirdStepReferenceFilename);
99 }
```



## Index

./csharp/Platform.RegularExpressions.Transformer.Tests/MarkovAlgorithmsTests.cs, 6  
./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs, 6  
./csharp/Platform.RegularExpressions.Transformer.Tests/TransformersTests.cs, 6  
./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, 2  
./csharp/Platform.RegularExpressions.Transformer/Transformer.cs, 3  
./csharp/Platform.RegularExpressions.Transformer/TransformerCLI.cs, 4  
./csharp/Platform.RegularExpressions.Transformer/TransformerExtensions.cs, 5  
./csharp/Platform.RegularExpressions.Transformer/TransformersListExtensions.cs, 5