

LinksPlatform's Platform.RegularExpressions.Transformer Class Library

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

```
1 using System;
2 using System.Collections.Generic;
3 using System.Diagnostics;
4 using System.IO;
5 using System.Runtime.CompilerServices;
6 using System.Text;
7 using System.Threading.Tasks;
8
9 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
10
11 namespace Platform.RegularExpressions.Transformer
12 {
13     public class FileTransformer : IFileTransformer
14     {
15         protected readonly ITextTransformer _textTransformer;
16
17         public string SourceFileExtension
18         {
19             [MethodImpl(MethodImplOptions.AggressiveInlining)]
20             get;
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             private set;
23         }
24
25         public string TargetFileExtension
26         {
27             [MethodImpl(MethodImplOptions.AggressiveInlining)]
28             get;
29             [MethodImpl(MethodImplOptions.AggressiveInlining)]
30             private set;
31         }
32
33         public IList<ISubstitutionRule> Rules
34         {
35             [MethodImpl(MethodImplOptions.AggressiveInlining)]
36             get => _textTransformer.Rules;
37         }
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public FileTransformer(ITextTransformer textTransformer, string sourceFileExtension,
41             → string targetFileExtension)
42         {
43             _textTransformer = textTransformer;
44             SourceFileExtension = sourceFileExtension;
45             TargetFileExtension = targetFileExtension;
46         }
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public void Transform(string sourcePath, string targetPath)
50         {
51             var sourceDirectoryExists = DirectoryExists(sourcePath);
52             var sourceDirectoryPath = LooksLikeDirectoryPath(sourcePath);
53             var sourceIsDirectory = sourceDirectoryExists || sourceDirectoryPath;
54             var targetDirectoryExists = DirectoryExists(targetPath);
55             var targetDirectoryPath = LooksLikeDirectoryPath(targetPath);
56             var targetIsDirectory = targetDirectoryExists || targetDirectoryPath;
57             if (sourceIsDirectory && targetIsDirectory)
58             {
59                 // Folder -> Folder
60                 if (!sourceDirectoryExists)
61                 {
62                     return;
63                 }
64                 TransformFolder(sourcePath, targetPath);
65             }
66             else if (!(sourceIsDirectory || targetIsDirectory))
67             {
68                 // File -> File
69                 EnsureSourceFileExists(sourcePath);
70                 EnsureTargetFileDirectoryExists(targetPath);
71                 TransformFile(sourcePath, targetPath);
72             }
73             else if (targetIsDirectory)
74             {
75                 // File -> Folder
76                 EnsureSourceFileExists(sourcePath);
77                 EnsureTargetDirectoryExists(targetPath, targetDirectoryExists);
78                 TransformFile(sourcePath, GetTargetFileName(sourcePath, targetPath));
79             }
80         }
81     }
82 }
```

```

78     }
79     else
80     {
81         // Folder -> File
82         throw new NotSupportedException();
83     }
84 }
85
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 protected virtual void TransformFolder(string sourcePath, string targetPath)
88 {
89     if (CountFilesRecursively(sourcePath, SourceFileExtension) == 0)
90     {
91         return;
92     }
93     EnsureTargetDirectoryExists(targetPath);
94     var directories = Directory.GetDirectories(sourcePath);
95     for (var i = 0; i < directories.Length; i++)
96     {
97         #if NETSTANDARD2_1
98             var relativePath = Path.GetRelativePath(sourcePath, directories[i]);
99         #else
100             var relativePath = directories[i].Replace(sourcePath.TrimEnd('\\') + "\\ ", "");
101         #endif
102         var newTargetPath = Path.Combine(targetPath, relativePath);
103         TransformFolder(directories[i], newTargetPath);
104     }
105     var files = Directory.GetFiles(sourcePath);
106     Parallel.For(0, files.Length, i =>
107     {
108         var file = files[i];
109         if (FileExtensionMatches(file, SourceFileExtension))
110         {
111             TransformFile(file, GetTargetFileName(file, targetPath));
112         }
113     });
114 }
115
116 [MethodImpl(MethodImplOptions.AggressiveInlining)]
117 protected virtual void TransformFile(string sourcePath, string targetPath)
118 {
119     if (File.Exists(targetPath))
120     {
121         var applicationPath = Process.GetCurrentProcess().MainModule.FileName;
122         var targetFileLastUpdateDateTime = new FileInfo(targetPath).LastWriteTimeUtc;
123         if (new FileInfo(sourcePath).LastWriteTimeUtc < targetFileLastUpdateDateTime &&
124             ↪ new FileInfo(applicationPath).LastWriteTimeUtc <
125             ↪ targetFileLastUpdateDateTime)
126         {
127             return;
128         }
129     }
130     var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
131     var targetText = _textTransformer.Transform(sourceText);
132     File.WriteAllText(targetPath, targetText, Encoding.UTF8);
133 }
134
135 [MethodImpl(MethodImplOptions.AggressiveInlining)]
136 protected string GetTargetFileName(string sourcePath, string targetDirectory) =>
137     ↪ Path.ChangeExtension(Path.Combine(targetDirectory, Path.GetFileName(sourcePath)),
138     ↪ TargetFileExtension);
139
140 [MethodImpl(MethodImplOptions.AggressiveInlining)]
141 private static long CountFilesRecursively(string path, string extension)
142 {
143     var files = Directory.GetFiles(path);
144     var directories = Directory.GetDirectories(path);
145     var result = 0L;
146     for (var i = 0; i < directories.Length; i++)
147     {
148         result += CountFilesRecursively(directories[i], extension);
149     }
150     for (var i = 0; i < files.Length; i++)
151     {
152         if (FileExtensionMatches(files[i], extension))
153         {
154             result++;
155         }
156     }
157 }

```

```

152     }
153     return result;
154 }
155
156 [MethodImpl(MethodImplOptions.AggressiveInlining)]
157 private static bool FileExtensionMatches(string file, string extension) =>
158     ↪ file.EndsWith(extension, StringComparison.OrdinalIgnoreCase);
159
160 [MethodImpl(MethodImplOptions.AggressiveInlining)]
161 private static void EnsureTargetFileDirectoryExists(string targetPath)
162 {
163     if (!File.Exists(targetPath))
164     {
165         EnsureDirectoryIsCreated(targetPath);
166     }
167 }
168
169 [MethodImpl(MethodImplOptions.AggressiveInlining)]
170 private static void EnsureTargetDirectoryExists(string targetPath) =>
171     ↪ EnsureTargetDirectoryExists(targetPath, DirectoryExists(targetPath));
172
173 [MethodImpl(MethodImplOptions.AggressiveInlining)]
174 private static void EnsureTargetDirectoryExists(string targetPath, bool
175     ↪ targetDirectoryExists)
176 {
177     if (!targetDirectoryExists)
178     {
179         Directory.CreateDirectory(targetPath);
180     }
181 }
182
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 private static void EnsureSourceFileExists(string sourcePath)
185 {
186     if (!File.Exists(sourcePath))
187     {
188         throw new FileNotFoundException("Source file does not exists.", sourcePath);
189     }
190 }
191
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 private static void EnsureDirectoryIsCreated(string targetPath) =>
194     ↪ Directory.CreateDirectory(Path.GetDirectoryName(targetPath));
195
196 [MethodImpl(MethodImplOptions.AggressiveInlining)]
197 private static bool DirectoryExists(string path) => Directory.Exists(path) &&
198     ↪ File.GetAttributes(path).HasFlag(FileAttributes.Directory);
199
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 private static bool LooksLikeDirectoryPath(string path) =>
202     ↪ path.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
203     ↪ path.EndsWith(Path.AltDirectorySeparatorChar.ToString());
204 }
205 }

```

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

```

1 using System.Runtime.CompilerServices;
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 IFileTransformer : ITransformer
8     {
9         string SourceFileExtension
10         {
11             [MethodImpl(MethodImplOptions.AggressiveInlining)]
12             get;
13         }
14
15         string TargetFileExtension
16         {
17             [MethodImpl(MethodImplOptions.AggressiveInlining)]
18             get;
19         }
20
21         [MethodImpl(MethodImplOptions.AggressiveInlining)]
22         void Transform(string sourcePath, string targetPath);
23     }
24 }

```

```
24 }
```

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         int MaximumRepeatCount
23         {
24             [MethodImpl(MethodImplOptions.AggressiveInlining)]
25             get;
26         }
27     }
28 }
```

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

```
1 using System.Runtime.CompilerServices;
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 ITextTransformer : ITransformer
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10         string Transform(string sourceText);
11     }
12 }
```

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

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.CompilerServices;
5 using Platform.Collections;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.RegularExpressions.Transformer
10 {
11     public static class ITextTransformerExtensions
12     {
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public static IList<ITextTransformer> GenerateTransformersForEachRule(this
15             ↪ ITextTransformer transformer)
16         {
17             var transformers = new List<ITextTransformer>();
18             for (int i = 1; i <= transformer.Rules.Count; i++)
19             {
20                 transformers.Add(new TextTransformer(transformer.Rules.Take(i).ToList()));
21             }
22             return transformers;
23         }
24
25         [MethodImpl(MethodImplOptions.AggressiveInlining)]
26         public static IList<string> GetSteps(this ITextTransformer transformer, string
27             ↪ sourceText)
28         {
29             if (transformer != null && !transformer.Rules.IsNullOrEmpty())
30             {
31                 var steps = new List<string>();
32                 var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
33                     ↪ sourceText);
```



```

40         var newText = transformer.Transform(sourceText);
41         transformer.WriteStep(directoryName, targetFilename, targetExtension, i, ref
            ↪ lastText, newText, skipFilesWithNoChanges);
42     }
43 }
44 }
45 }
46 }

```

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

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
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 ITransformer
9     {
10         IList<ISubstitutionRule> Rules
11         {
12             [MethodImpl(MethodImplOptions.AggressiveInlining)]
13             get;
14         }
15     }
16 }

```

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

```

1 using System.Runtime.CompilerServices;
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 ITransformerExtensions
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10        public static void WriteStep(this ITransformer transformer, string directoryName, string
            ↪ targetFilename, string targetExtension, int currentStep, ref string lastText, string
            ↪ newText, bool skipFilesWithNoChanges)
11        {
12            if (!(skipFilesWithNoChanges && string.Equals(lastText, newText)))
13            {
14                lastText = newText;
15                newText.WriteToFile(directoryName,
                    ↪ $"{targetFilename}.{currentStep}{targetExtension}");
16                var ruleString = transformer.Rules[currentStep].ToString();
17                ruleString.WriteToFile(directoryName,
                    ↪ $"{targetFilename}.{currentStep}.rule.txt");
18            }
19        }
20    }
21 }

```

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

```

1 using System.IO;
2 using System.Runtime.CompilerServices;
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 LoggingFileTransformer : FileTransformer
10    {
11        [MethodImpl(MethodImplOptions.AggressiveInlining)]
12        public LoggingFileTransformer(ITextTransformer textTransformer, string
            ↪ sourceFileExtension, string targetFileExtension) : base(textTransformer,
            ↪ sourceFileExtension, targetFileExtension) { }
13
14        [MethodImpl(MethodImplOptions.AggressiveInlining)]
15        protected override void TransformFile(string sourcePath, string targetPath)
16        {
17            base.TransformFile(sourcePath, targetPath);
18            // Logging
19            var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
20            _textTransformer.WriteStepsToFiles(sourceText, targetPath, skipFilesWithNoChanges:
            ↪ true);

```

```

21     }
22 }
23 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
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 static class RegexExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static Regex OverrideOptions(this Regex regex, RegexOptions options, TimeSpan
            ↪ matchTimeout)
13         {
14             if (regex == null)
15             {
16                 return null;
17             }
18             return new Regex(regex.ToString(), options, matchTimeout);
19         }
20     }
21 }

```

1.11 ./csharp/Platform.RegularExpressions.Transformer/StringExtensions.cs

```

1  using System.IO;
2  using System.Runtime.CompilerServices;
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      internal static class StringExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static void GetPathParts(this string path, out string directoryName, out string
            ↪ targetFilename, out string targetExtension) => (directoryName, targetFilename,
            ↪ targetExtension) = (Path.GetDirectoryName(path),
            ↪ Path.GetFileNameWithoutExtension(path), Path.GetExtension(path));
13
14         [MethodImpl(MethodImplOptions.AggressiveInlining)]
15         public static void WriteToFile(this string text, string directoryName, string
            ↪ targetFilename) => File.WriteAllText(Path.Combine(directoryName, targetFilename),
            ↪ text, Encoding.UTF8);
16     }
17 }

```

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

```

1  using System;
2  using System.Runtime.CompilerServices;
3  using System.Text;
4  using System.Text.RegularExpressions;
5
6  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8  namespace Platform.RegularExpressions.Transformer
9  {
10     public class SubstitutionRule : ISubstitutionRule
11     {
12         public static readonly TimeSpan DefaultMatchTimeout = TimeSpan.FromMinutes(5);
13         public static readonly RegexOptions DefaultMatchPatternRegexOptions =
            ↪ RegexOptions.Compiled | RegexOptions.Multiline;
14
15         public Regex MatchPattern
16         {
17             [MethodImpl(MethodImplOptions.AggressiveInlining)]
18             get;
19             [MethodImpl(MethodImplOptions.AggressiveInlining)]
20             set;
21         }
22
23         public string SubstitutionPattern
24         {
25             [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

26         get;
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         set;
29     }
30
31     public Regex PathPattern
32     {
33         [MethodImpl(MethodImplOptions.AggressiveInlining)]
34         get;
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         set;
37     }
38
39     public int MaximumRepeatCount
40     {
41         [MethodImpl(MethodImplOptions.AggressiveInlining)]
42         get;
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         set;
45     }
46
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
49     ↪ maximumRepeatCount, RegexOptions? matchPatternOptions, TimeSpan? matchTimeout)
50     {
51         MatchPattern = matchPattern;
52         SubstitutionPattern = substitutionPattern;
53         MaximumRepeatCount = maximumRepeatCount;
54         OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
55         ↪ matchTimeout ?? matchPattern.MatchTimeout);
56     }
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
60     ↪ maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
61     ↪ substitutionPattern, maximumRepeatCount, useDefaultOptions ?
62     ↪ DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
63     ↪ DefaultMatchTimeout : (TimeSpan?)null) { }
64
65     [MethodImpl(MethodImplOptions.AggressiveInlining)]
66     public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
67     ↪ maximumRepeatCount) : this(matchPattern, substitutionPattern, maximumRepeatCount,
68     ↪ true) { }
69
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
72     ↪ this(matchPattern, substitutionPattern, 0) { }
73
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public static implicit operator SubstitutionRule(ValueTuple<string, string> tuple) =>
76     ↪ new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2);
77
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
80     ↪ SubstitutionRule(tuple.Item1, tuple.Item2);
81
82     [MethodImpl(MethodImplOptions.AggressiveInlining)]
83     public static implicit operator SubstitutionRule(ValueTuple<string, string, int> tuple)
84     ↪ => new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2, tuple.Item3);
85
86     [MethodImpl(MethodImplOptions.AggressiveInlining)]
87     public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
88     ↪ => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);
89
90     [MethodImpl(MethodImplOptions.AggressiveInlining)]
91     public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
92     ↪ MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);
93
94     [MethodImpl(MethodImplOptions.AggressiveInlining)]
95     public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
96     ↪ PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
97
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public override string ToString()
100     {
101         var sb = new StringBuilder();
102         sb.Append(' ');
103         sb.Append(MatchPattern.ToString());

```



```

89         sb.Append('');
90         sb.Append(" -> ");
91         sb.Append('');
92         sb.Append(SubstitutionPattern);
93         sb.Append('');
94         if (PathPattern != null)
95         {
96             sb.Append(" on files ");
97             sb.Append('');
98             sb.Append(PathPattern.ToString());
99             sb.Append('');
100        }
101        if (MaximumRepeatCount > 0)
102        {
103            if (MaximumRepeatCount >= int.MaxValue)
104            {
105                sb.Append(" repeated forever");
106            }
107            else
108            {
109                sb.Append(" repeated up to ");
110                sb.Append(MaximumRepeatCount);
111                sb.Append(" times");
112            }
113        }
114        return sb.ToString();
115    }
116 }
117 }

```

1.13 ./csharp/Platform.RegularExpressions.Transformer/TextSteppedTransformer.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
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 TextSteppedTransformer : ITransformer
10     {
11         public IList<ISubstitutionRule> Rules
12         {
13             [MethodImpl(MethodImplOptions.AggressiveInlining)]
14             get;
15             [MethodImpl(MethodImplOptions.AggressiveInlining)]
16             set;
17         }
18
19         public string Text
20         {
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             get;
23             [MethodImpl(MethodImplOptions.AggressiveInlining)]
24             set;
25         }
26
27         public int Current
28         {
29             [MethodImpl(MethodImplOptions.AggressiveInlining)]
30             get;
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             set;
33         }
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public TextSteppedTransformer(IList<ISubstitutionRule> rules, string text, int current)
37             => Reset(rules, text, current);
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public TextSteppedTransformer(IList<ISubstitutionRule> rules, string text) =>
41             Reset(rules, text);
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public TextSteppedTransformer(IList<ISubstitutionRule> rules) => Reset(rules);
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public TextSteppedTransformer() => Reset();
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

48     public void Reset(IList<ISubstitutionRule> rules, string text, int current)
49     {
50         Rules = rules;
51         Text = text;
52         Current = current;
53     }
54
55     [MethodImpl(MethodImplOptions.AggressiveInlining)]
56     public void Reset(IList<ISubstitutionRule> rules, string text) => Reset(rules, text, -1);
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public void Reset(IList<ISubstitutionRule> rules) => Reset(rules, "", -1);
60
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public void Reset(string text) => Reset(Rules, text, -1);
63
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public void Reset() => Reset(Array.Empty<ISubstitutionRule>(), "", -1);
66
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public bool Next()
69     {
70         var current = Current + 1;
71         if (current >= Rules.Count)
72         {
73             return false;
74         }
75         var rule = Rules[current];
76         var matchPattern = rule.MatchPattern;
77         var substitutionPattern = rule.SubstitutionPattern;
78         var maximumRepeatCount = rule.MaximumRepeatCount;
79         var replaceCount = 0;
80         var text = Text;
81         do
82         {
83             text = matchPattern.Replace(text, substitutionPattern);
84             replaceCount++;
85         }
86         while ((maximumRepeatCount == int.MaxValue || replaceCount <= maximumRepeatCount) &&
87             ↪ matchPattern.IsMatch(text));
88         Text = text;
89         Current = current;
90         return true;
91     }
92 }

```

1.14 ./csharp/Platform.RegularExpressions.Transformer/TextTransformer.cs

```

1  using System.Collections.Generic;
2  using System.Runtime.CompilerServices;
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 TextTransformer : ITextTransformer
9      {
10         public IList<ISubstitutionRule> Rules
11         {
12             [MethodImpl(MethodImplOptions.AggressiveInlining)]
13             get;
14             [MethodImpl(MethodImplOptions.AggressiveInlining)]
15             private set;
16         }
17
18         [MethodImpl(MethodImplOptions.AggressiveInlining)]
19         public TextTransformer(IList<ISubstitutionRule> substitutionRules)
20         {
21             Rules = substitutionRules;
22         }
23
24         [MethodImpl(MethodImplOptions.AggressiveInlining)]
25         public string Transform(string source)
26         {
27             var baseTrasformer = new TextSteppedTransformer(Rules);
28             baseTrasformer.Reset(source);
29             while (baseTrasformer.Next());
30             return baseTrasformer.Text;
31         }
32     }
33 }

```

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

```

1 using System.Runtime.CompilerServices;
2 using Platform.Collections.Arrays;
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 TransformerCLI
9     {
10         private readonly IFileTransformer _transformer;
11
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public TransformerCLI(IFileTransformer transformer) => _transformer = transformer;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public void Run(string[] args)
17         {
18             var sourcePath = args.GetElementOrDefault(0);
19             var targetPath = args.GetElementOrDefault(1);
20             _transformer.Transform(sourcePath, targetPath);
21         }
22     }
23 }

```

1.16 ./csharp/Platform.RegularExpressions.Transformer.Tests/FileTransformerTests.cs

```

1 using System.IO;
2 using Xunit;
3
4 namespace Platform.RegularExpressions.Transformer.Tests
5 {
6     public class FileTransformerTests
7     {
8         [Fact]
9         public void FolderToFolderTransformationTest()
10        {
11            var tempPath = Path.GetTempPath();
12            var sourceFolderPath = Path.Combine(tempPath,
13                ↪ "FileTransformerTestsFolderToFolderTransformationTestSourceFolder");
14            var targetFolderPath = Path.Combine(tempPath,
15                ↪ "FileTransformerTestsFolderToFolderTransformationTestTargetFolder");
16
17            var baseTransformer = new TextTransformer(new SubstitutionRule[]
18            {
19                ("a", "b"),
20                ("b", "c")
21            });
22            var fileTransformer = new FileTransformer(baseTransformer, ".cs", ".cpp");
23
24            // Delete before creation (if previous test failed)
25            if (Directory.Exists(sourceFolderPath))
26            {
27                Directory.Delete(sourceFolderPath, true);
28            }
29            if (Directory.Exists(targetFolderPath))
30            {
31                Directory.Delete(targetFolderPath, true);
32            }
33
34            Directory.CreateDirectory(sourceFolderPath);
35            Directory.CreateDirectory(targetFolderPath);
36
37            File.WriteAllText(Path.Combine(sourceFolderPath, "a.cs"), "a a a");
38            var aFolderPath = Path.Combine(sourceFolderPath, "A");
39            Directory.CreateDirectory(aFolderPath);
40            Directory.CreateDirectory(Path.Combine(sourceFolderPath, "B"));
41            File.WriteAllText(Path.Combine(aFolderPath, "b.cs"), "b b b");
42            File.WriteAllText(Path.Combine(sourceFolderPath, "x.txt"), "should not be
43                ↪ translated");
44
45            fileTransformer.Transform(sourceFolderPath,
46                ↪ $"{targetFolderPath}{Path.DirectorySeparatorChar}");
47
48            var aCppFile = Path.Combine(targetFolderPath, "a.cpp");
49            Assert.True(File.Exists(aCppFile));
50            Assert.Equal("c c c", File.ReadAllText(aCppFile));
51            Assert.True(Directory.Exists(Path.Combine(targetFolderPath, "A")));
52            Assert.False(Directory.Exists(Path.Combine(targetFolderPath, "B")));
53            var bCppFile = Path.Combine(targetFolderPath, "A", "b.cpp");

```

```

50     Assert.True(File.Exists(bCppFile));
51     Assert.Equal("c c c", File.ReadAllText(bCppFile));
52     Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.txt")));
53     Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.cpp")));
54
55     Directory.Delete(sourceFolderPath, true);
56     Directory.Delete(targetFolderPath, true);
57 }
58 }
59 }

```

1.17 ./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 TextTransformer(rules);
23             var input = "101";
24             var expectedOutput = "||||";
25             var output = transformer.Transform(input);
26             Assert.Equal(expectedOutput, output);
27         }
28     }
29 }

```

1.18 ./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\./]+\$"), "", 0);
12             Assert.Equal(RegexOptions.Compiled | RegexOptions.Multiline,
13                 ↳ rule.MatchPattern.Options);
14         }
15     }
16 }

```

1.19 ./csharp/Platform.RegularExpressions.Transformer.Tests/TextTransformerTests.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 TextTransformerTests
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 TextTransformer(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 transformer = new TextTransformer(new SubstitutionRule[] {
37         (new Regex("a"), "b"),
38         (new Regex("b"), "c")
39     });
40
41     var targetFilename = Path.GetTempFileName();
42
43     transformer.WriteStepsToFiles(sourceText, $"{targetFilename}.txt",
44         ↪ skipFilesWithNoChanges: false);
45
46     CheckAndCleanUpTwoRulesFiles(firstStepReferenceText, secondStepReferenceText,
47         ↪ transformer, targetFilename);
48 }
49
50 private static void CheckAndCleanUpTwoRulesFiles(string firstStepReferenceText, string
51     ↪ secondStepReferenceText, TextTransformer transformer, string targetFilename)
52 {
53     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
54     var firstStepRuleFilename = $"{targetFilename}.0.rule.txt";
55     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
56     var secondStepRuleFilename = $"{targetFilename}.1.rule.txt";
57
58     Assert.True(File.Exists(firstStepReferenceFilename));
59     Assert.True(File.Exists(firstStepRuleFilename));
60     Assert.True(File.Exists(secondStepReferenceFilename));
61     Assert.True(File.Exists(secondStepRuleFilename));
62
63     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
64         ↪ Encoding.UTF8));
65     Assert.Equal(transformer.Rules[0].ToString(),
66         ↪ File.ReadAllText(firstStepRuleFilename, Encoding.UTF8));
67     Assert.Equal(secondStepReferenceText, File.ReadAllText(secondStepReferenceFilename,
68         ↪ Encoding.UTF8));
69     Assert.Equal(transformer.Rules[1].ToString(),
70         ↪ File.ReadAllText(secondStepRuleFilename, Encoding.UTF8));
71
72     File.Delete(firstStepReferenceFilename);
73     File.Delete(firstStepRuleFilename);
74     File.Delete(secondStepReferenceFilename);
75     File.Delete(secondStepRuleFilename);
76 }
77
78 [Fact]
79 public void FilesWithNoChangesSkippedTest()
80 {
81     var sourceText = "aaaa";
82     var firstStepReferenceText = "bbbb";
83     var thirdStepReferenceText = "cccc";
84
85     var transformer = new TextTransformer(new SubstitutionRule[] {
86         (new Regex("a"), "b"),
87         (new Regex("x"), "y"),
88         (new Regex("b"), "c")
89     });
90
91     var targetFilename = Path.GetTempFileName();
92
93     transformer.WriteStepsToFiles(sourceText, $"{targetFilename}.txt",
94         ↪ skipFilesWithNoChanges: true);
95 }

```

```

88         CheckAndCleanUpThreeRulesFiles(firstStepReferenceText, thirdStepReferenceText,
89         ↪ transformer, targetFilename);
90     }
91     private static void CheckAndCleanUpThreeRulesFiles(string firstStepReferenceText, string
92     ↪ thirdStepReferenceText, TextTransformer transformer, string targetFilename)
93     {
94         var firstStepReferenceFilename = $"{targetFilename}.0.txt";
95         var firstStepRuleFilename = $"{targetFilename}.0.rule.txt";
96         var secondStepReferenceFilename = $"{targetFilename}.1.txt";
97         var secondStepRuleFilename = $"{targetFilename}.1.rule.txt";
98         var thirdStepReferenceFilename = $"{targetFilename}.2.txt";
99         var thirdStepRuleFilename = $"{targetFilename}.2.rule.txt";
100
101         Assert.True(File.Exists(firstStepReferenceFilename));
102         Assert.True(File.Exists(firstStepRuleFilename));
103         Assert.False(File.Exists(secondStepReferenceFilename));
104         Assert.False(File.Exists(secondStepRuleFilename));
105         Assert.True(File.Exists(thirdStepReferenceFilename));
106         Assert.True(File.Exists(thirdStepRuleFilename));
107
108         Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
109         ↪ Encoding.UTF8));
110         Assert.Equal(transformer.Rules[0].ToString(),
111         ↪ File.ReadAllText(firstStepRuleFilename, Encoding.UTF8));
112         Assert.Equal(thirdStepReferenceText, File.ReadAllText(thirdStepReferenceFilename,
113         ↪ Encoding.UTF8));
114         Assert.Equal(transformer.Rules[2].ToString(),
115         ↪ File.ReadAllText(thirdStepRuleFilename, Encoding.UTF8));
116
117         File.Delete(firstStepReferenceFilename);
118         File.Delete(firstStepRuleFilename);
119         File.Delete(secondStepReferenceFilename);
120         File.Delete(secondStepRuleFilename);
121         File.Delete(thirdStepReferenceFilename);
122         File.Delete(thirdStepRuleFilename);
123     }
124
125     [Fact]
126     public void DebugOutputUsingTransformersGenerationTest()
127     {
128         var sourceText = "aaaa";
129         var firstStepReferenceText = "bbbb";
130         var secondStepReferenceText = "cccc";
131
132         var transformer = new TextTransformer(new SubstitutionRule[] {
133             (new Regex("a"), "b"),
134             (new Regex("b"), "c")
135         });
136
137         var steps =
138         ↪ transformer.GenerateTransformersForEachRule().TransformWithAll(sourceText);
139
140         Assert.Equal(2, steps.Count);
141         Assert.Equal(firstStepReferenceText, steps[0]);
142         Assert.Equal(secondStepReferenceText, steps[1]);
143     }
144
145     [Fact]
146     public void DebugFilesOutputUsingTransformersGenerationTest()
147     {
148         var sourceText = "aaaa";
149         var firstStepReferenceText = "bbbb";
150         var secondStepReferenceText = "cccc";
151
152         var transformer = new TextTransformer(new SubstitutionRule[] {
153             (new Regex("a"), "b"),
154             (new Regex("b"), "c")
155         });
156
157         var targetFilename = Path.GetTempFileName();
158
159         transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
160         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: false);
161
162         CheckAndCleanUpTwoRulesFiles(firstStepReferenceText, secondStepReferenceText,
163         ↪ transformer, targetFilename);
164     }

```

```

157 [Fact]
158 public void FilesWithNoChangesSkippedWhenUsingTransformersGenerationTest()
159 {
160     var sourceText = "aaaa";
161     var firstStepReferenceText = "bbbb";
162     var thirdStepReferenceText = "cccc";
163
164     var transformer = new TextTransformer(new SubstitutionRule[] {
165         (new Regex("a"), "b"),
166         (new Regex("x"), "y"),
167         (new Regex("b"), "c")
168     });
169
170     var targetFilename = Path.GetTempFileName();
171
172     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
173         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: true);
174
175     CheckAndCleanUpThreeRulesFiles(firstStepReferenceText, thirdStepReferenceText,
176         ↪ transformer, targetFilename);
177 }
178 }

```

Index

./csharp/Platform.RegularExpressions.Transformer.Tests/FileTransformerTests.cs, 11
./csharp/Platform.RegularExpressions.Transformer.Tests/MarkovAlgorithmsTests.cs, 12
./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs, 12
./csharp/Platform.RegularExpressions.Transformer.Tests/TextTransformerTests.cs, 12
./csharp/Platform.RegularExpressions.Transformer/FileTransformer.cs, 1
./csharp/Platform.RegularExpressions.Transformer/IFileTransformer.cs, 3
./csharp/Platform.RegularExpressions.Transformer/ISubstitutionRule.cs, 4
./csharp/Platform.RegularExpressions.Transformer/ITextTransformer.cs, 4
./csharp/Platform.RegularExpressions.Transformer/ITextTransformerExtensions.cs, 4
./csharp/Platform.RegularExpressions.Transformer/ITextTransformersListExtensions.cs, 5
./csharp/Platform.RegularExpressions.Transformer/ITransformer.cs, 6
./csharp/Platform.RegularExpressions.Transformer/ITransformerExtensions.cs, 6
./csharp/Platform.RegularExpressions.Transformer/LoggingFileTransformer.cs, 6
./csharp/Platform.RegularExpressions.Transformer/RegexExtensions.cs, 7
./csharp/Platform.RegularExpressions.Transformer/StringExtensions.cs, 7
./csharp/Platform.RegularExpressions.Transformer/SubstitutionRule.cs, 7
./csharp/Platform.RegularExpressions.Transformer/TextSteppedTransformer.cs, 9
./csharp/Platform.RegularExpressions.Transformer/TextTransformer.cs, 10
./csharp/Platform.RegularExpressions.Transformer/TransformerCLI.cs, 11