

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 defaultPath = Path.GetFullPath(".");
52             if (string.IsNullOrEmpty(sourcePath))
53             {
54                 sourcePath = defaultPath;
55             }
56             if (string.IsNullOrEmpty(targetPath))
57             {
58                 targetPath = defaultPath;
59             }
60             var sourceDirectoryExists = DirectoryExists(sourcePath);
61             var sourceDirectoryPath = LooksLikeDirectoryPath(sourcePath);
62             var sourceIsDirectory = sourceDirectoryExists || sourceDirectoryPath;
63             var targetDirectoryExists = DirectoryExists(targetPath);
64             var targetDirectoryPath = LooksLikeDirectoryPath(targetPath);
65             var targetIsDirectory = targetDirectoryExists || targetDirectoryPath;
66             if (sourceIsDirectory && targetIsDirectory)
67             {
68                 // Folder -> Folder
69                 if (!sourceDirectoryExists)
70                 {
71                     return;
72                 }
73                 TransformFolder(sourcePath, targetPath);
74             }
75             else if (!(sourceIsDirectory || targetIsDirectory))
76             {
77                 // File -> File
78                 EnsureSourceFileExists(sourcePath);
79             }
80         }
81     }
82 }
```

```

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

```

```

150     var result = 0L;
151     for (var i = 0; i < directories.Length; i++)
152     {
153         result += CountFilesRecursively(directories[i], extension);
154     }
155     for (var i = 0; i < files.Length; i++)
156     {
157         if (FileExtensionMatches(files[i], extension))
158         {
159             result++;
160         }
161     }
162     return result;
163 }
164
165 [MethodImpl(MethodImplOptions.AggressiveInlining)]
166 private static bool FileExtensionMatches(string file, string extension) =>
167     file.EndsWith(extension, StringComparison.OrdinalIgnoreCase);
168
169 [MethodImpl(MethodImplOptions.AggressiveInlining)]
170 private static void EnsureTargetFileDirectoryExists(string targetPath)
171 {
172     if (!File.Exists(targetPath))
173     {
174         EnsureDirectoryIsCreated(targetPath);
175     }
176 }
177
178 [MethodImpl(MethodImplOptions.AggressiveInlining)]
179 private static void EnsureTargetDirectoryExists(string targetPath) =>
180     EnsureTargetDirectoryExists(targetPath, DirectoryExists(targetPath));
181
182 [MethodImpl(MethodImplOptions.AggressiveInlining)]
183 private static void EnsureTargetDirectoryExists(string targetPath, bool
184     targetDirectoryExists)
185 {
186     if (!targetDirectoryExists)
187     {
188         Directory.CreateDirectory(targetPath);
189     }
190 }
191
192 [MethodImpl(MethodImplOptions.AggressiveInlining)]
193 private static void EnsureSourceFileExists(string sourcePath)
194 {
195     if (!File.Exists(sourcePath))
196     {
197         throw new FileNotFoundException("Source file does not exists.", sourcePath);
198     }
199 }
200
201 [MethodImpl(MethodImplOptions.AggressiveInlining)]
202 private static void EnsureDirectoryIsCreated(string targetPath) =>
203     Directory.CreateDirectory(Path.GetDirectoryName(targetPath));
204
205 [MethodImpl(MethodImplOptions.AggressiveInlining)]
206 private static bool DirectoryExists(string path) => Directory.Exists(path) &&
207     File.GetAttributes(path).HasFlag(FileAttributes.Directory);
208
209 [MethodImpl(MethodImplOptions.AggressiveInlining)]
210 private static bool LooksLikeDirectoryPath(string path) =>
211     path.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
212     path.EndsWith(Path.AltDirectorySeparatorChar.ToString());
213 }

```

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 }

```

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         }
23     }
24 }

```

```

21     return transformers;
22 }
23
24 [MethodImpl(MethodImplOptions.AggressiveInlining)]
25 public static IList<string> GetSteps(this ITextTransformer transformer, string
    ↪ sourceText)
26 {
27     if (transformer != null && !transformer.Rules.IsNullOrEmpty())
28     {
29         var steps = new List<string>();
30         var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
            ↪ sourceText);
31         while (steppedTransformer.Next())
32         {
33             steps.Add(steppedTransformer.Text);
34         }
35         return steps;
36     }
37     else
38     {
39         return Array.Empty<string>();
40     }
41 }
42
43 [MethodImpl(MethodImplOptions.AggressiveInlining)]
44 public static void WriteStepsToFiles(this ITextTransformer transformer, string
    ↪ sourceText, string targetPath, bool skipFilesWithNoChanges)
45 {
46     if (transformer != null && !transformer.Rules.IsNullOrEmpty())
47     {
48         targetPath.GetPathParts(out var directoryName, out var targetFilename, out var
            ↪ targetExtension);
49         Steps.DeleteAllSteps(directoryName, targetFilename, targetExtension);
50         var lastText = "";
51         var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
            ↪ sourceText);
52         while (steppedTransformer.Next())
53         {
54             var newText = steppedTransformer.Text;
55             Steps.WriteStep(transformer, directoryName, targetFilename, targetExtension,
                ↪ steppedTransformer.Current, ref lastText, newText,
                ↪ skipFilesWithNoChanges);
56         }
57     }
58 }
59 }
60 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Runtime.CompilerServices;
4 using Platform.Collections;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.RegularExpressions.Transformer
9 {
10     public static class ITextTransformersListExtensions
11     {
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public static IList<string> TransformWithAll(this IList<ITextTransformer> transformers,
            ↪ string source)
14         {
15             if (!transformers.IsNullOrEmpty())
16             {
17                 var steps = new List<string>();
18                 for (int i = 0; i < transformers.Count; i++)
19                 {
20                     steps.Add(transformers[i].Transform(source));
21                 }
22                 return steps;
23             }
24             else
25             {
26                 return Array.Empty<string>();
27             }
28         }
29     }
30 }

```

```

29     [MethodImpl(MethodImplOptions.AggressiveInlining)]
30     public static void TransformWithAllToFiles(this IList<ITextTransformer> transformers,
31     ↪     string sourceText, string targetPath, bool skipFilesWithNoChanges)
32     {
33         if (!transformers.IsNullOrEmpty())
34         {
35             targetPath.GetPathParts(out var directoryName, out var targetFilename, out var
36             ↪     targetExtension);
37             Steps.DeleteAllSteps(directoryName, targetFilename, targetExtension);
38             var lastText = "";
39             for (int i = 0; i < transformers.Count; i++)
40             {
41                 var transformer = transformers[i];
42                 var newText = transformer.Transform(sourceText);
43                 Steps.WriteStep(transformer, directoryName, targetFilename, targetExtension,
44                 ↪     i, ref lastText, newText, skipFilesWithNoChanges);
45             }
46         }
47     }

```

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/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
13         ↪     sourceFileExtension, string targetFileExtension) : base(textTransformer,
14         ↪     sourceFileExtension, targetFileExtension) { }
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         protected override void TransformFile(string sourcePath, string targetPath)
18         {
19             base.TransformFile(sourcePath, targetPath);
20             // Logging
21             var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
22             _textTransformer.WriteStepsToFiles(sourceText, targetPath, skipFilesWithNoChanges:
23             ↪     true);
24         }
25     }
26 }

```

1.9 ./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 }

```

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

```

1  using System.IO;
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 static class Steps
9      {
10         [MethodImpl(MethodImplOptions.AggressiveInlining)]
11         public static void DeleteAllSteps(string directoryName, string targetFilename, string
        ↳ targetExtension)
12         {
13             FileHelpers.DeleteAll(directoryName, $"{targetFilename}.*.rule.txt");
14             FileHelpers.DeleteAll(directoryName, $"{targetFilename}.*{targetExtension}");
15         }
16
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         public static void WriteStep(ITransformer transformer, string directoryName, string
        ↳ targetFilename, string targetExtension, int currentStep, ref string lastText, string
        ↳ newText, bool skipFilesWithNoChanges)
19         {
20             if (!(skipFilesWithNoChanges && string.Equals(lastText, newText)))
21             {
22                 lastText = newText;
23                 newText.WriteToFile(directoryName,
                ↳ $"{targetFilename}.{currentStep}{targetExtension}");
24                 var ruleString = transformer.Rules[currentStep].ToString();
25                 ruleString.WriteToFile(directoryName,
                ↳ $"{targetFilename}.{currentStep}.rule.txt");
26             }
27         }
28     }
29 }

```

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 =
14             ↳ RegexOptions.Compiled | RegexOptions.Multiline;
15
16         public Regex MatchPattern
17         {
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             get;
20             [MethodImpl(MethodImplOptions.AggressiveInlining)]
21             set;
22         }
23
24         public string SubstitutionPattern
25         {
26             [MethodImpl(MethodImplOptions.AggressiveInlining)]
27             get;
28             [MethodImpl(MethodImplOptions.AggressiveInlining)]
29             set;
30         }
31
32         public Regex PathPattern
33         {
34             [MethodImpl(MethodImplOptions.AggressiveInlining)]
35             get;
36             [MethodImpl(MethodImplOptions.AggressiveInlining)]
37             set;
38         }
39
40         public int MaximumRepeatCount
41         {
42             [MethodImpl(MethodImplOptions.AggressiveInlining)]
43             get;
44             [MethodImpl(MethodImplOptions.AggressiveInlining)]
45             set;
46         }
47
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
50             ↳ maximumRepeatCount, RegexOptions? matchPatternOptions, TimeSpan? matchTimeout)
51         {
52             MatchPattern = matchPattern;
53             SubstitutionPattern = substitutionPattern;
54             MaximumRepeatCount = maximumRepeatCount;
55             OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
56                 ↳ matchTimeout ?? matchPattern.MatchTimeout);
57         }
58
59         [MethodImpl(MethodImplOptions.AggressiveInlining)]
60         public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
61             ↳ maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
62                 ↳ substitutionPattern, maximumRepeatCount, useDefaultOptions ?
63                 ↳ DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
64                 ↳ DefaultMatchTimeout : (TimeSpan?)null) { }
65
66         [MethodImpl(MethodImplOptions.AggressiveInlining)]
67         public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
68             ↳ this(matchPattern, substitutionPattern, 0) { }
69
70         [MethodImpl(MethodImplOptions.AggressiveInlining)]
71         public static implicit operator SubstitutionRule(ValueTuple<string, string> tuple) =>
72             ↳ new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2);
73
74         [MethodImpl(MethodImplOptions.AggressiveInlining)]

```



```

69     public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
    ↳ SubstitutionRule(tuple.Item1, tuple.Item2);
70
71     [MethodImpl(MethodImplOptions.AggressiveInlining)]
72     public static implicit operator SubstitutionRule(ValueTuple<string, string, int> tuple)
    ↳ => new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2, tuple.Item3);
73
74     [MethodImpl(MethodImplOptions.AggressiveInlining)]
75     public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
    ↳ => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);
76
77     [MethodImpl(MethodImplOptions.AggressiveInlining)]
78     public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
    ↳ MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);
79
80     [MethodImpl(MethodImplOptions.AggressiveInlining)]
81     public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
    ↳ PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
82
83     [MethodImpl(MethodImplOptions.AggressiveInlining)]
84     public override string ToString()
85     {
86         var sb = new StringBuilder();
87         sb.Append('');
88         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(ICollection<ISubstitutionRule> rules, string text, int current)
37     {
38         => Reset(rules, text, current);
39     }
40
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public TextSteppedTransformer(ICollection<ISubstitutionRule> rules, string text) =>
43     {
44         Reset(rules, text);
45     }
46
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public TextSteppedTransformer() => Reset();
49
50     [MethodImpl(MethodImplOptions.AggressiveInlining)]
51     public void Reset(ICollection<ISubstitutionRule> rules, string text, int current)
52     {
53         Rules = rules;
54         Text = text;
55         Current = current;
56     }
57
58     [MethodImpl(MethodImplOptions.AggressiveInlining)]
59     public void Reset(ICollection<ISubstitutionRule> rules, string text) => Reset(rules, text, -1);
60
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public void Reset(ICollection<ISubstitutionRule> rules) => Reset(rules, "", -1);
63
64     [MethodImpl(MethodImplOptions.AggressiveInlining)]
65     public void Reset(string text) => Reset(Rules, text, -1);
66
67     [MethodImpl(MethodImplOptions.AggressiveInlining)]
68     public void Reset() => Reset(Array.Empty<ISubstitutionRule>(), "", -1);
69
70     [MethodImpl(MethodImplOptions.AggressiveInlining)]
71     public bool Next()
72     {
73         var current = Current + 1;
74         if (current >= Rules.Count)
75         {
76             return false;
77         }
78         var rule = Rules[current];
79         var matchPattern = rule.MatchPattern;
80         var substitutionPattern = rule.SubstitutionPattern;
81         var maximumRepeatCount = rule.MaximumRepeatCount;
82         var replaceCount = 0;
83         var text = Text;
84         do
85         {
86             text = matchPattern.Replace(text, substitutionPattern);
87             replaceCount++;
88         }
89         while ((maximumRepeatCount == int.MaxValue || replaceCount <= maximumRepeatCount) &&
90             matchPattern.IsMatch(text));
91         Text = text;
92         Current = current;
93         return true;
94     }
95 }
96 }

```

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))

```

```

24     {
25         Directory.Delete(sourceFolderPath, true);
26     }
27     if (Directory.Exists(targetFolderPath))
28     {
29         Directory.Delete(targetFolderPath, true);
30     }
31
32     Directory.CreateDirectory(sourceFolderPath);
33     Directory.CreateDirectory(targetFolderPath);
34
35     File.WriteAllText(Path.Combine(sourceFolderPath, "a.cs"), "a a a");
36     var aFolderPath = Path.Combine(sourceFolderPath, "A");
37     Directory.CreateDirectory(aFolderPath);
38     Directory.CreateDirectory(Path.Combine(sourceFolderPath, "B"));
39     File.WriteAllText(Path.Combine(aFolderPath, "b.cs"), "b b b");
40     File.WriteAllText(Path.Combine(sourceFolderPath, "x.txt"), "should not be
    ↪ translated");
41
42     fileTransformer.Transform(sourceFolderPath,
    ↪ $"{targetFolderPath}{Path.DirectorySeparatorChar}");
43
44     var aCppFile = Path.Combine(targetFolderPath, "a.cpp");
45     Assert.True(File.Exists(aCppFile));
46     Assert.Equal("c c c", File.ReadAllText(aCppFile));
47     Assert.True(Directory.Exists(Path.Combine(targetFolderPath, "A")));
48     Assert.False(Directory.Exists(Path.Combine(targetFolderPath, "B")));
49     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 }

```

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));

```

```

63     Assert.Equal(transformer.Rules[1].ToString(),
64         ↪ File.ReadAllText(secondStepRuleFilename, Encoding.UTF8));
65
66     File.Delete(firstStepReferenceFilename);
67     File.Delete(firstStepRuleFilename);
68     File.Delete(secondStepReferenceFilename);
69     File.Delete(secondStepRuleFilename);
70 }
71
72 [Fact]
73 public void FilesWithNoChangesSkippedTest()
74 {
75     var sourceText = "aaaa";
76     var firstStepReferenceText = "bbbb";
77     var thirdStepReferenceText = "cccc";
78
79     var transformer = new TextTransformer(new SubstitutionRule[] {
80         (new Regex("a"), "b"),
81         (new Regex("x"), "y"),
82         (new Regex("b"), "c")
83     });
84
85     var targetFilename = Path.GetTempFileName();
86
87     transformer.WriteStepsToFiles(sourceText, $"{targetFilename}.txt",
88         ↪ skipFilesWithNoChanges: true);
89
90     CheckAndCleanUpThreeRulesFiles(firstStepReferenceText, thirdStepReferenceText,
91         ↪ transformer, targetFilename);
92 }
93
94 private static void CheckAndCleanUpThreeRulesFiles(string firstStepReferenceText, string
95     ↪ thirdStepReferenceText, TextTransformer transformer, string targetFilename)
96 {
97     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
98     var firstStepRuleFilename = $"{targetFilename}.0.rule.txt";
99     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
100    var secondStepRuleFilename = $"{targetFilename}.1.rule.txt";
101    var thirdStepReferenceFilename = $"{targetFilename}.2.txt";
102    var thirdStepRuleFilename = $"{targetFilename}.2.rule.txt";
103
104    Assert.True(File.Exists(firstStepReferenceFilename));
105    Assert.True(File.Exists(firstStepRuleFilename));
106    Assert.False(File.Exists(secondStepReferenceFilename));
107    Assert.False(File.Exists(secondStepRuleFilename));
108    Assert.True(File.Exists(thirdStepReferenceFilename));
109    Assert.True(File.Exists(thirdStepRuleFilename));
110
111    Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
112        ↪ Encoding.UTF8));
113    Assert.Equal(transformer.Rules[0].ToString(),
114        ↪ File.ReadAllText(firstStepRuleFilename, Encoding.UTF8));
115    Assert.Equal(thirdStepReferenceText, File.ReadAllText(thirdStepReferenceFilename,
116        ↪ Encoding.UTF8));
117    Assert.Equal(transformer.Rules[2].ToString(),
118        ↪ File.ReadAllText(thirdStepRuleFilename, Encoding.UTF8));
119
120    File.Delete(firstStepReferenceFilename);
121    File.Delete(firstStepRuleFilename);
122    File.Delete(secondStepReferenceFilename);
123    File.Delete(secondStepRuleFilename);
124    File.Delete(thirdStepReferenceFilename);
125    File.Delete(thirdStepRuleFilename);
126 }
127
128 [Fact]
129 public void DebugOutputUsingTransformersGenerationTest()
130 {
131     var sourceText = "aaaa";
132     var firstStepReferenceText = "bbbb";
133     var secondStepReferenceText = "cccc";
134
135     var transformer = new TextTransformer(new SubstitutionRule[] {
136         (new Regex("a"), "b"),
137         (new Regex("b"), "c")
138     });

```

```

132     var steps =
133         ↪ transformer.GenerateTransformersForEachRule().TransformWithAll(sourceText);
134
135     Assert.Equal(2, steps.Count);
136     Assert.Equal(firstStepReferenceText, steps[0]);
137     Assert.Equal(secondStepReferenceText, steps[1]);
138 }
139
140 [Fact]
141 public void DebugFilesOutputUsingTransformersGenerationTest()
142 {
143     var sourceText = "aaaa";
144     var firstStepReferenceText = "bbbb";
145     var secondStepReferenceText = "cccc";
146
147     var transformer = new TextTransformer(new SubstitutionRule[] {
148         (new Regex("a"), "b"),
149         (new Regex("b"), "c")
150     });
151
152     var targetFilename = Path.GetTempFileName();
153
154     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
155         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: false);
156
157     CheckAndCleanUpTwoRulesFiles(firstStepReferenceText, secondStepReferenceText,
158         ↪ transformer, targetFilename);
159 }
160
161 [Fact]
162 public void FilesWithNoChangesSkippedWhenUsingTransformersGenerationTest()
163 {
164     var sourceText = "aaaa";
165     var firstStepReferenceText = "bbbb";
166     var thirdStepReferenceText = "cccc";
167
168     var transformer = new TextTransformer(new SubstitutionRule[] {
169         (new Regex("a"), "b"),
170         (new Regex("x"), "y"),
171         (new Regex("b"), "c")
172     });
173
174     var targetFilename = Path.GetTempFileName();
175
176     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
177         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: true);
178
179     CheckAndCleanUpThreeRulesFiles(firstStepReferenceText, thirdStepReferenceText,
180         ↪ transformer, targetFilename);
181 }
182 }

```

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, 13
- ./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/LoggingFileTransformer.cs, 6
- ./csharp/Platform.RegularExpressions.Transformer/RegexExtensions.cs, 6
- ./csharp/Platform.RegularExpressions.Transformer/Steps.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