

# 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     /// <summary>
14     /// <para>
15     /// Represents the file transformer.
16     /// </para>
17     /// <para></para>
18     /// </summary>
19     /// <seealso cref="IFileTransformer"/>
20     public class FileTransformer : IFileTransformer
21     {
22         /// <summary>
23         /// <para>
24         /// The text transformer.
25         /// </para>
26         /// <para></para>
27         /// </summary>
28         protected readonly ITextTransformer _textTransformer;
29
30         /// <summary>
31         /// <para>
32         /// Gets or sets the source file extension value.
33         /// </para>
34         /// <para></para>
35         /// </summary>
36         public string SourceFileExtension
37         {
38             [MethodImpl(MethodImplOptions.AggressiveInlining)]
39             get;
40             [MethodImpl(MethodImplOptions.AggressiveInlining)]
41             private set;
42         }
43
44         /// <summary>
45         /// <para>
46         /// Gets or sets the target file extension value.
47         /// </para>
48         /// <para></para>
49         /// </summary>
50         public string TargetFileExtension
51         {
52             [MethodImpl(MethodImplOptions.AggressiveInlining)]
53             get;
54             [MethodImpl(MethodImplOptions.AggressiveInlining)]
55             private set;
56         }
57
58         /// <summary>
59         /// <para>
60         /// Gets the rules value.
61         /// </para>
62         /// <para></para>
63         /// </summary>
64         public IList<ISubstitutionRule> Rules
65         {
66             [MethodImpl(MethodImplOptions.AggressiveInlining)]
67             get => _textTransformer.Rules;
68         }
69
70         /// <summary>
71         /// <para>
72         /// Initializes a new <see cref="FileTransformer"/> instance.
73         /// </para>
74         /// <para></para>
75         /// </summary>
76         /// <param name="textTransformer">
77         /// <para>A text transformer.</para>
```

```

78     /// <para></para>
79     /// </param>
80     /// <param name="sourceFileExtension">
81     /// <para>A source file extension.</para>
82     /// <para></para>
83     /// </param>
84     /// <param name="targetFileExtension">
85     /// <para>A target file extension.</para>
86     /// <para></para>
87     /// </param>
88     [MethodImpl(MethodImplOptions.AggressiveInlining)]
89     public FileTransformer(ITextTransformer textTransformer, string sourceFileExtension,
90     ↪ string targetFileExtension)
91     {
92         _textTransformer = textTransformer;
93         SourceFileExtension = sourceFileExtension;
94         TargetFileExtension = targetFileExtension;
95     }
96     /// <summary>
97     /// <para>
98     /// Transforms the source path.
99     /// </para>
100    /// <para></para>
101    /// </summary>
102    /// <param name="sourcePath">
103    /// <para>The source path.</para>
104    /// <para></para>
105    /// </param>
106    /// <param name="targetPath">
107    /// <para>The target path.</para>
108    /// <para></para>
109    /// </param>
110    /// <exception cref="NotSupportedException">
111    /// <para></para>
112    /// <para></para>
113    /// </exception>
114    [MethodImpl(MethodImplOptions.AggressiveInlining)]
115    public void Transform(string sourcePath, string targetPath)
116    {
117        var defaultPath = Path.GetFullPath(".");
118        if (string.IsNullOrEmpty(sourcePath))
119        {
120            sourcePath = defaultPath;
121        }
122        if (string.IsNullOrEmpty(targetPath))
123        {
124            targetPath = defaultPath;
125        }
126        var sourceDirectoryExists = DirectoryExists(sourcePath);
127        var sourceDirectoryPath = LooksLikeDirectoryPath(sourcePath);
128        var sourceIsDirectory = sourceDirectoryExists || sourceDirectoryPath;
129        var targetDirectoryExists = DirectoryExists(targetPath);
130        var targetDirectoryPath = LooksLikeDirectoryPath(targetPath);
131        var targetIsDirectory = targetDirectoryExists || targetDirectoryPath;
132        if (sourceIsDirectory && targetIsDirectory)
133        {
134            // Folder -> Folder
135            if (!sourceDirectoryExists)
136            {
137                return;
138            }
139            TransformFolder(sourcePath, targetPath);
140        }
141        else if (!(sourceIsDirectory || targetIsDirectory))
142        {
143            // File -> File
144            EnsureSourceFileExists(sourcePath);
145            EnsureTargetFileDirectoryExists(targetPath);
146            TransformFile(sourcePath, targetPath);
147        }
148        else if (targetIsDirectory)
149        {
150            // File -> Folder
151            EnsureSourceFileExists(sourcePath);
152            EnsureTargetDirectoryExists(targetPath, targetDirectoryExists);
153            TransformFile(sourcePath, GetTargetFileName(sourcePath, targetPath));
154        }

```

```

155     else
156     {
157         // Folder -> File
158         throw new NotSupportedException();
159     }
160 }
161
162 /// <summary>
163 /// <para>
164 /// Transforms the folder using the specified source path.
165 /// </para>
166 /// <para></para>
167 /// </summary>
168 /// <param name="sourcePath">
169 /// <para>The source path.</para>
170 /// <para></para>
171 /// </param>
172 /// <param name="targetPath">
173 /// <para>The target path.</para>
174 /// <para></para>
175 /// </param>
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 protected virtual void TransformFolder(string sourcePath, string targetPath)
178 {
179     if (CountFilesRecursively(sourcePath, SourceFileExtension) == 0)
180     {
181         return;
182     }
183     EnsureTargetDirectoryExists(targetPath);
184     var directories = Directory.GetDirectories(sourcePath);
185     for (var i = 0; i < directories.Length; i++)
186     {
187 #if NETSTANDARD2_1
188         var relativePath = Path.GetRelativePath(sourcePath, directories[i]);
189 #else
190         var relativePath =
191             ↪ directories[i].Replace(sourcePath.TrimEnd(Path.DirectorySeparatorChar) +
192             ↪ Path.DirectorySeparatorChar, "");
193 #endif
194         var newTargetPath = Path.Combine(targetPath, relativePath);
195         TransformFolder(directories[i], newTargetPath);
196     }
197     var files = Directory.GetFiles(sourcePath);
198     Parallel.For(0, files.Length, i =>
199     {
200         var file = files[i];
201         if (FileExtensionMatches(file, SourceFileExtension))
202         {
203             TransformFile(file, GetTargetFileName(file, targetPath));
204         }
205     });
206 }
207
208 /// <summary>
209 /// <para>
210 /// Transforms the file using the specified source path.
211 /// </para>
212 /// <para></para>
213 /// </summary>
214 /// <param name="sourcePath">
215 /// <para>The source path.</para>
216 /// <para></para>
217 /// </param>
218 /// <param name="targetPath">
219 /// <para>The target path.</para>
220 /// <para></para>
221 /// </param>
222 [MethodImpl(MethodImplOptions.AggressiveInlining)]
223 protected virtual void TransformFile(string sourcePath, string targetPath)
224 {
225     if (File.Exists(targetPath))
226     {
227         var applicationPath = Process.GetCurrentProcess().MainModule.FileName;
228         var targetFileLastUpdateDateTime = new FileInfo(targetPath).LastWriteTimeUtc;
229         if (new FileInfo(sourcePath).LastWriteTimeUtc < targetFileLastUpdateDateTime &&
230             ↪ new FileInfo(applicationPath).LastWriteTimeUtc <
231             ↪ targetFileLastUpdateDateTime)
232         {

```

```

229         return;
230     }
231 }
232 var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
233 var targetText = _textTransformer.Transform(sourceText);
234 File.WriteAllText(targetPath, targetText, Encoding.UTF8);
235 }
236
237 /// <summary>
238 /// <para>
239 /// Gets the target file name using the specified source path.
240 /// </para>
241 /// <para></para>
242 /// </summary>
243 /// <param name="sourcePath">
244 /// <para>The source path.</para>
245 /// <para></para>
246 /// </param>
247 /// <param name="targetDirectory">
248 /// <para>The target directory.</para>
249 /// <para></para>
250 /// </param>
251 /// <returns>
252 /// <para>The string</para>
253 /// <para></para>
254 /// </returns>
255 [MethodImpl(MethodImplOptions.AggressiveInlining)]
256 protected string GetTargetFileName(string sourcePath, string targetDirectory) =>
    ↪ Path.ChangeExtension(Path.Combine(targetDirectory, Path.GetFileName(sourcePath)),
    ↪ TargetFileExtension);
257
258 /// <summary>
259 /// <para>
260 /// Counts the files recursively using the specified path.
261 /// </para>
262 /// <para></para>
263 /// </summary>
264 /// <param name="path">
265 /// <para>The path.</para>
266 /// <para></para>
267 /// </param>
268 /// <param name="extension">
269 /// <para>The extension.</para>
270 /// <para></para>
271 /// </param>
272 /// <returns>
273 /// <para>The result.</para>
274 /// <para></para>
275 /// </returns>
276 [MethodImpl(MethodImplOptions.AggressiveInlining)]
277 private static long CountFilesRecursively(string path, string extension)
278 {
279     var files = Directory.GetFiles(path);
280     var directories = Directory.GetDirectories(path);
281     var result = 0L;
282     for (var i = 0; i < directories.Length; i++)
283     {
284         result += CountFilesRecursively(directories[i], extension);
285     }
286     for (var i = 0; i < files.Length; i++)
287     {
288         if (FileExtensionMatches(files[i], extension))
289         {
290             result++;
291         }
292     }
293     return result;
294 }
295
296 /// <summary>
297 /// <para>
298 /// Determines whether file extension matches.
299 /// </para>
300 /// <para></para>
301 /// </summary>
302 /// <param name="file">
303 /// <para>The file.</para>
304 /// <para></para>

```

```

305     /// </param>
306     /// <param name="extension">
307     /// <para>The extension.</para>
308     /// <para></para>
309     /// </param>
310     /// <returns>
311     /// <para>The bool</para>
312     /// <para></para>
313     /// </returns>
314     [MethodImpl(MethodImplOptions.AggressiveInlining)]
315     private static bool FileExtensionMatches(string file, string extension) =>
316         ↪ file.EndsWith(extension, StringComparison.OrdinalIgnoreCase);
317
318     /// <summary>
319     /// <para>
320     /// Ensures the target file directory exists using the specified target path.
321     /// </para>
322     /// <para></para>
323     /// </summary>
324     /// <param name="targetPath">
325     /// <para>The target path.</para>
326     /// <para></para>
327     /// </param>
328     [MethodImpl(MethodImplOptions.AggressiveInlining)]
329     private static void EnsureTargetFileDirectoryExists(string targetPath)
330     {
331         if (!File.Exists(targetPath))
332         {
333             EnsureDirectoryIsCreated(targetPath);
334         }
335     }
336
337     /// <summary>
338     /// <para>
339     /// Ensures the target directory exists using the specified target path.
340     /// </para>
341     /// <para></para>
342     /// </summary>
343     /// <param name="targetPath">
344     /// <para>The target path.</para>
345     /// <para></para>
346     /// </param>
347     [MethodImpl(MethodImplOptions.AggressiveInlining)]
348     private static void EnsureTargetDirectoryExists(string targetPath) =>
349         ↪ EnsureTargetDirectoryExists(targetPath, Directory.Exists(targetPath));
350
351     /// <summary>
352     /// <para>
353     /// Ensures the target directory exists using the specified target path.
354     /// </para>
355     /// <para></para>
356     /// </summary>
357     /// <param name="targetPath">
358     /// <para>The target path.</para>
359     /// <para></para>
360     /// </param>
361     /// <param name="targetDirectoryExists">
362     /// <para>The target directory exists.</para>
363     /// <para></para>
364     /// </param>
365     [MethodImpl(MethodImplOptions.AggressiveInlining)]
366     private static void EnsureTargetDirectoryExists(string targetPath, bool
367         ↪ targetDirectoryExists)
368     {
369         if (!targetDirectoryExists)
370         {
371             Directory.CreateDirectory(targetPath);
372         }
373     }
374
375     /// <summary>
376     /// <para>
377     /// Ensures the source file exists using the specified source path.
378     /// </para>
379     /// <para></para>
380     /// </summary>
381     /// <param name="sourcePath">
382     /// <para>The source path.</para>

```

```

380     /// <para></para>
381     /// </param>
382     /// <exception cref="FileNotFoundException">
383     /// <para>Source file does not exists. </para>
384     /// <para></para>
385     /// </exception>
386     [MethodImpl(MethodImplOptions.AggressiveInlining)]
387     private static void EnsureSourceFileExists(string sourcePath)
388     {
389         if (!File.Exists(sourcePath))
390         {
391             throw new FileNotFoundException("Source file does not exists.", sourcePath);
392         }
393     }
394
395     /// <summary>
396     /// <para>
397     /// Ensures the directory is created using the specified target path.
398     /// </para>
399     /// <para></para>
400     /// </summary>
401     /// <param name="targetPath">
402     /// <para>The target path.</para>
403     /// <para></para>
404     /// </param>
405     [MethodImpl(MethodImplOptions.AggressiveInlining)]
406     private static void EnsureDirectoryIsCreated(string targetPath) =>
407         Directory.CreateDirectory(Path.GetDirectoryName(targetPath));
408
409     /// <summary>
410     /// <para>
411     /// Determines whether directory exists.
412     /// </para>
413     /// <para></para>
414     /// </summary>
415     /// <param name="path">
416     /// <para>The path.</para>
417     /// <para></para>
418     /// </param>
419     /// <returns>
420     /// <para>The bool</para>
421     /// <para></para>
422     /// </returns>
423     [MethodImpl(MethodImplOptions.AggressiveInlining)]
424     private static bool DirectoryExists(string path) => Directory.Exists(path) &&
425         File.GetAttributes(path).HasFlag(FileAttributes.Directory);
426
427     /// <summary>
428     /// <para>
429     /// Determines whether looks like directory path.
430     /// </para>
431     /// <para></para>
432     /// </summary>
433     /// <param name="path">
434     /// <para>The path.</para>
435     /// <para></para>
436     /// </param>
437     /// <returns>
438     /// <para>The bool</para>
439     /// <para></para>
440     /// </returns>
441     [MethodImpl(MethodImplOptions.AggressiveInlining)]
442     private static bool LooksLikeDirectoryPath(string path) =>
443         path.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
444         path.EndsWith(Path.AltDirectorySeparatorChar.ToString());
445 }

```

## 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     /// <summary>
8     /// <para>
9     /// Defines the file transformer.

```

```

10  /// </para>
11  /// <para></para>
12  /// </summary>
13  /// <seealso cref="ITransformer"/>
14  public interface IFileTransformer : ITransformer
15  {
16      /// <summary>
17      /// <para>
18      /// Gets the source file extension value.
19      /// </para>
20      /// <para></para>
21      /// </summary>
22      string SourceFileExtension
23      {
24          [MethodImpl(MethodImplOptions.AggressiveInlining)]
25          get;
26      }
27
28      /// <summary>
29      /// <para>
30      /// Gets the target file extension value.
31      /// </para>
32      /// <para></para>
33      /// </summary>
34      string TargetFileExtension
35      {
36          [MethodImpl(MethodImplOptions.AggressiveInlining)]
37          get;
38      }
39
40      /// <summary>
41      /// <para>
42      /// Transforms the source path.
43      /// </para>
44      /// <para></para>
45      /// </summary>
46      /// <param name="sourcePath">
47      /// <para>The source path.</para>
48      /// <para></para>
49      /// </param>
50      /// <param name="targetPath">
51      /// <para>The target path.</para>
52      /// <para></para>
53      /// </param>
54      [MethodImpl(MethodImplOptions.AggressiveInlining)]
55      void Transform(string sourcePath, string targetPath);
56  }
57

```

### 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      /// <summary>
9      /// <para>
10     /// Defines the substitution rule.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     public interface ISubstitutionRule
15     {
16         /// <summary>
17         /// <para>
18         /// Gets the match pattern value.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         Regex MatchPattern
23         {
24             [MethodImpl(MethodImplOptions.AggressiveInlining)]
25             get;
26         }
27
28         /// <summary>
29         /// <para>

```

```

30     /// Gets the substitution pattern value.
31     /// </para>
32     /// <para></para>
33     /// </summary>
34     string SubstitutionPattern
35     {
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         get;
38     }
39
40     /// <summary>
41     /// <para>
42     /// Gets the maximum repeat count value.
43     /// </para>
44     /// <para></para>
45     /// </summary>
46     int MaximumRepeatCount
47     {
48         [MethodImpl(MethodImplOptions.AggressiveInlining)]
49         get;
50     }
51 }
52 }

```

#### 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      /// <summary>
8      /// <para>
9      /// Defines the text transformer.
10     /// </para>
11     /// <para></para>
12     /// </summary>
13     /// <seealso cref="ITransformer"/>
14     public interface ITextTransformer : ITransformer
15     {
16         /// <summary>
17         /// <para>
18         /// Transforms the source text.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         /// <param name="sourceText">
23         /// <para>The source text.</para>
24         /// <para></para>
25         /// </param>
26         /// <returns>
27         /// <para>The string</para>
28         /// <para></para>
29         /// </returns>
30         [MethodImpl(MethodImplOptions.AggressiveInlining)]
31         string Transform(string sourceText);
32     }
33 }

```

#### 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     /// <summary>
12     /// <para>
13     /// Represents the text transformer extensions.
14     /// </para>
15     /// <para></para>
16     /// </summary>
17     public static class ITextTransformerExtensions
18     {
19         /// <summary>

```



```

20    /// <para>
21    /// Generates the transformers for each rule using the specified transformer.
22    /// </para>
23    /// <para></para>
24    /// </summary>
25    /// <param name="transformer">
26    /// <para>The transformer.</para>
27    /// <para></para>
28    /// </param>
29    /// <returns>
30    /// <para>The transformers.</para>
31    /// <para></para>
32    /// </returns>
33    [MethodImpl(MethodImplOptions.AggressiveInlining)]
34    public static IList<ITextTransformer> GenerateTransformersForEachRule(this
    ↪ ITextTransformer transformer)
35    {
36        var transformers = new List<ITextTransformer>();
37        for (int i = 1; i <= transformer.Rules.Count; i++)
38        {
39            transformers.Add(new TextTransformer(transformer.Rules.Take(i).ToList()));
40        }
41        return transformers;
42    }
43
44    /// <summary>
45    /// <para>
46    /// Gets the steps using the specified transformer.
47    /// </para>
48    /// <para></para>
49    /// </summary>
50    /// <param name="transformer">
51    /// <para>The transformer.</para>
52    /// <para></para>
53    /// </param>
54    /// <param name="sourceText">
55    /// <para>The source text.</para>
56    /// <para></para>
57    /// </param>
58    /// <returns>
59    /// <para>A list of string</para>
60    /// <para></para>
61    /// </returns>
62    [MethodImpl(MethodImplOptions.AggressiveInlining)]
63    public static IList<string> GetSteps(this ITextTransformer transformer, string
    ↪ sourceText)
64    {
65        if (transformer != null && !transformer.Rules.IsNullOrEmpty())
66        {
67            var steps = new List<string>();
68            var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
    ↪ sourceText);
69            while (steppedTransformer.Next())
70            {
71                steps.Add(steppedTransformer.Text);
72            }
73            return steps;
74        }
75        else
76        {
77            return Array.Empty<string>();
78        }
79    }
80
81    /// <summary>
82    /// <para>
83    /// Writes the steps to files using the specified transformer.
84    /// </para>
85    /// <para></para>
86    /// </summary>
87    /// <param name="transformer">
88    /// <para>The transformer.</para>
89    /// <para></para>
90    /// </param>
91    /// <param name="sourceText">
92    /// <para>The source text.</para>
93    /// <para></para>
94    /// </param>

```

```

95     /// <param name="targetPath">
96     /// <para>The target path.</para>
97     /// <para></para>
98     /// </param>
99     /// <param name="skipFilesWithNoChanges">
100    /// <para>The skip files with no changes.</para>
101    /// <para></para>
102    /// </param>
103    [MethodImpl(MethodImplOptions.AggressiveInlining)]
104    public static void WriteStepsToFiles(this ITextTransformer transformer, string
    ↪ sourceText, string targetPath, bool skipFilesWithNoChanges)
105    {
106        if (transformer != null && !transformer.Rules.IsNullOrEmpty())
107        {
108            targetPath.GetPathParts(out var directoryName, out var targetFilename, out var
    ↪ targetExtension);
109            Steps.DeleteAllSteps(directoryName, targetFilename, targetExtension);
110            var lastText = "";
111            var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
    ↪ sourceText);
112            while (steppedTransformer.Next())
113            {
114                var newText = steppedTransformer.Text;
115                Steps.WriteStep(transformer, directoryName, targetFilename, targetExtension,
    ↪ steppedTransformer.Current, ref lastText, newText,
    ↪ skipFilesWithNoChanges);
116            }
117        }
118    }
119 }
120 }

```

## 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     /// <summary>
11     /// <para>
12     /// Represents the text transformers list extensions.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     public static class ITextTransformersListExtensions
17     {
18         /// <summary>
19         /// <para>
20         /// Transforms the with all using the specified transformers.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="transformers">
25         /// <para>The transformers.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="source">
29         /// <para>The source.</para>
30         /// <para></para>
31         /// </param>
32         /// <returns>
33         /// <para>A list of string</para>
34         /// <para></para>
35         /// </returns>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public static IList<string> TransformWithAll(this IList<ITextTransformer> transformers,
    ↪ string source)
38         {
39             if (!transformers.IsNullOrEmpty())
40             {
41                 var steps = new List<string>();
42                 for (int i = 0; i < transformers.Count; i++)
43                 {
44                     steps.Add(transformers[i].Transform(source));

```

```

45     }
46     return steps;
47 }
48 else
49 {
50     return Array.Empty<string>();
51 }
52 }
53
54 /// <summary>
55 /// <para>
56 /// Transforms the with all to files using the specified transformers.
57 /// </para>
58 /// <para></para>
59 /// </summary>
60 /// <param name="transformers">
61 /// <para>The transformers.</para>
62 /// <para></para>
63 /// </param>
64 /// <param name="sourceText">
65 /// <para>The source text.</para>
66 /// <para></para>
67 /// </param>
68 /// <param name="targetPath">
69 /// <para>The target path.</para>
70 /// <para></para>
71 /// </param>
72 /// <param name="skipFilesWithNoChanges">
73 /// <para>The skip files with no changes.</para>
74 /// <para></para>
75 /// </param>
76 [MethodImpl(MethodImplOptions.AggressiveInlining)]
77 public static void TransformWithAllToFiles(this IList<ITextTransformer> transformers,
78     ↪ string sourceText, string targetPath, bool skipFilesWithNoChanges)
79 {
80     if (!transformers.IsNullOrEmpty())
81     {
82         targetPath.GetPathParts(out var directoryName, out var targetFilename, out var
83             ↪ targetExtension);
84         Steps.DeleteAllSteps(directoryName, targetFilename, targetExtension);
85         var lastText = "";
86         for (int i = 0; i < transformers.Count; i++)
87         {
88             var transformer = transformers[i];
89             var newText = transformer.Transform(sourceText);
90             Steps.WriteStep(transformer, directoryName, targetFilename, targetExtension,
91                 ↪ i, ref lastText, newText, skipFilesWithNoChanges);
92         }
93     }
94 }

```

## 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     /// <summary>
9     /// <para>
10    /// Defines the transformer.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    public interface ITransformer
15    {
16        /// <summary>
17        /// <para>
18        /// Gets the rules value.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        IList<ISubstitutionRule> Rules
23        {
24            [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

25         get;
26     }
27 }
28 }

```

## 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      /// <summary>
10     /// <para>
11     /// Represents the logging file transformer.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="FileTransformer"/>
16     public class LoggingFileTransformer : FileTransformer
17     {
18         /// <summary>
19         /// <para>
20         /// Initializes a new <see cref="LoggingFileTransformer"/> instance.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         /// <param name="textTransformer">
25         /// <para>A text transformer.</para>
26         /// <para></para>
27         /// </param>
28         /// <param name="sourceFileExtension">
29         /// <para>A source file extension.</para>
30         /// <para></para>
31         /// </param>
32         /// <param name="targetFileExtension">
33         /// <para>A target file extension.</para>
34         /// <para></para>
35         /// </param>
36         [MethodImpl(MethodImplOptions.AggressiveInlining)]
37         public LoggingFileTransformer(ITextTransformer textTransformer, string
            ↪ sourceFileExtension, string targetFileExtension) : base(textTransformer,
            ↪ sourceFileExtension, targetFileExtension) { }
38
39         /// <summary>
40         /// <para>
41         /// Transforms the file using the specified source path.
42         /// </para>
43         /// <para></para>
44         /// </summary>
45         /// <param name="sourcePath">
46         /// <para>The source path.</para>
47         /// <para></para>
48         /// </param>
49         /// <param name="targetPath">
50         /// <para>The target path.</para>
51         /// <para></para>
52         /// </param>
53         [MethodImpl(MethodImplOptions.AggressiveInlining)]
54         protected override void TransformFile(string sourcePath, string targetPath)
55         {
56             base.TransformFile(sourcePath, targetPath);
57             // Logging
58             var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
59             _textTransformer.WriteStepsToFiles(sourceText, targetPath, skipFilesWithNoChanges:
            ↪ true);
60         }
61     }
62 }

```

## 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     /// <summary>
10    /// <para>
11    /// Represents the regex extensions.
12    /// </para>
13    /// <para></para>
14    /// </summary>
15    public static class RegexExtensions
16    {
17        /// <summary>
18        /// <para>
19        /// Overrides the options using the specified regex.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        /// <param name="regex">
24        /// <para>The regex.</para>
25        /// <para></para>
26        /// </param>
27        /// <param name="options">
28        /// <para>The options.</para>
29        /// <para></para>
30        /// </param>
31        /// <param name="matchTimeout">
32        /// <para>The match timeout.</para>
33        /// <para></para>
34        /// </param>
35        /// <returns>
36        /// <para>The regex</para>
37        /// <para></para>
38        /// </returns>
39        [MethodImpl(MethodImplOptions.AggressiveInlining)]
40        public static Regex OverrideOptions(this Regex regex, RegexOptions options, TimeSpan
41        ↪ matchTimeout)
42        {
43            if (regex == null)
44            {
45                return null;
46            }
47            return new Regex(regex.ToString(), options, matchTimeout);
48        }
49    }

```

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

```

1 using Platform.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     /// <summary>
9     /// <para>
10    /// Represents the steps.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    public static class Steps
15    {
16        /// <summary>
17        /// <para>
18        /// Deletes the all steps using the specified directory name.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        /// <param name="directoryName">
23        /// <para>The directory name.</para>
24        /// <para></para>
25        /// </param>
26        /// <param name="targetFilename">
27        /// <para>The target filename.</para>
28        /// <para></para>
29        /// </param>
30        /// <param name="targetExtension">
31        /// <para>The target extension.</para>

```

```

32     /// <para></para>
33     /// </param>
34     [MethodImpl(MethodImplOptions.AggressiveInlining)]
35     public static void DeleteAllSteps(string directoryName, string targetFilename, string
    ↪ targetExtension)
36     {
37         FileHelpers.DeleteAll(directoryName, $"{targetFilename}.rule.txt");
38         FileHelpers.DeleteAll(directoryName, $"{targetFilename}.{targetExtension}");
39     }
40
41     /// <summary>
42     /// <para>
43     /// Writes the step using the specified transformer.
44     /// </para>
45     /// <para></para>
46     /// </summary>
47     /// <param name="transformer">
48     /// <para>The transformer.</para>
49     /// <para></para>
50     /// </param>
51     /// <param name="directoryName">
52     /// <para>The directory name.</para>
53     /// <para></para>
54     /// </param>
55     /// <param name="targetFilename">
56     /// <para>The target filename.</para>
57     /// <para></para>
58     /// </param>
59     /// <param name="targetExtension">
60     /// <para>The target extension.</para>
61     /// <para></para>
62     /// </param>
63     /// <param name="currentStep">
64     /// <para>The current step.</para>
65     /// <para></para>
66     /// </param>
67     /// <param name="lastText">
68     /// <para>The last text.</para>
69     /// <para></para>
70     /// </param>
71     /// <param name="newText">
72     /// <para>The new text.</para>
73     /// <para></para>
74     /// </param>
75     /// <param name="skipFilesWithNoChanges">
76     /// <para>The skip files with no changes.</para>
77     /// <para></para>
78     /// </param>
79     [MethodImpl(MethodImplOptions.AggressiveInlining)]
80     public static void WriteStep(ITransformer transformer, string directoryName, string
    ↪ targetFilename, string targetExtension, int currentStep, ref string lastText, string
    ↪ newText, bool skipFilesWithNoChanges)
81     {
82         if (!(skipFilesWithNoChanges && string.Equals(lastText, newText)))
83         {
84             lastText = newText;
85             newText.WriteToFile(directoryName,
    ↪ $"{targetFilename}.{currentStep}{targetExtension}");
86             var ruleString = transformer.Rules[currentStep].ToString();
87             ruleString.WriteToFile(directoryName,
    ↪ $"{targetFilename}.{currentStep}.rule.txt");
88         }
89     }
90 }
91 }

```

## 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      /// <summary>
10     /// <para>
11     /// Represents the string extensions.

```

```

12  /// </para>
13  /// <para></para>
14  /// </summary>
15  internal static class StringExtensions
16  {
17      /// <summary>
18      /// <para>
19      /// Gets the path parts using the specified path.
20      /// </para>
21      /// <para></para>
22      /// </summary>
23      /// <param name="path">
24      /// <para>The path.</para>
25      /// <para></para>
26      /// </param>
27      /// <param name="directoryName">
28      /// <para>The directory name.</para>
29      /// <para></para>
30      /// </param>
31      /// <param name="targetFilename">
32      /// <para>The target filename.</para>
33      /// <para></para>
34      /// </param>
35      /// <param name="targetExtension">
36      /// <para>The target extension.</para>
37      /// <para></para>
38      /// </param>
39      [MethodImpl(MethodImplOptions.AggressiveInlining)]
40      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));
41
42      /// <summary>
43      /// <para>
44      /// Writes the to file using the specified text.
45      /// </para>
46      /// <para></para>
47      /// </summary>
48      /// <param name="text">
49      /// <para>The text.</para>
50      /// <para></para>
51      /// </param>
52      /// <param name="directoryName">
53      /// <para>The directory name.</para>
54      /// <para></para>
55      /// </param>
56      /// <param name="targetFilename">
57      /// <para>The target filename.</para>
58      /// <para></para>
59      /// </param>
60      [MethodImpl(MethodImplOptions.AggressiveInlining)]
61      public static void WriteToFile(this string text, string directoryName, string
        ↳ targetFilename) => File.WriteAllText(Path.Combine(directoryName, targetFilename),
        ↳ text, Encoding.UTF8);
62  }
63  }

```

## 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     /// <summary>
11     /// <para>
12     /// Represents the substitution rule.
13     /// </para>
14     /// <para></para>
15     /// </summary>
16     /// <seealso cref="ISubstitutionRule"/>
17     public class SubstitutionRule : ISubstitutionRule
18     {
19         /// <summary>

```

```

20     /// <para>
21     /// The from minutes.
22     /// </para>
23     /// <para></para>
24     /// </summary>
25     public static readonly TimeSpan DefaultMatchTimeout = TimeSpan.FromMinutes(5);
26     /// <summary>
27     /// <para>
28     /// The multiline.
29     /// </para>
30     /// <para></para>
31     /// </summary>
32     public static readonly RegexOptions DefaultMatchPatternRegexOptions =
33     ↪ RegexOptions.Compiled | RegexOptions.Multiline;
34     /// <summary>
35     /// <para>
36     /// Gets or sets the match pattern value.
37     /// </para>
38     /// <para></para>
39     /// </summary>
40     public Regex MatchPattern
41     {
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         get;
44         [MethodImpl(MethodImplOptions.AggressiveInlining)]
45         set;
46     }
47
48     /// <summary>
49     /// <para>
50     /// Gets or sets the substitution pattern value.
51     /// </para>
52     /// <para></para>
53     /// </summary>
54     public string SubstitutionPattern
55     {
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         get;
58         [MethodImpl(MethodImplOptions.AggressiveInlining)]
59         set;
60     }
61
62     /// <summary>
63     /// <para>
64     /// Gets or sets the path pattern value.
65     /// </para>
66     /// <para></para>
67     /// </summary>
68     public Regex PathPattern
69     {
70         [MethodImpl(MethodImplOptions.AggressiveInlining)]
71         get;
72         [MethodImpl(MethodImplOptions.AggressiveInlining)]
73         set;
74     }
75
76     /// <summary>
77     /// <para>
78     /// Gets or sets the maximum repeat count value.
79     /// </para>
80     /// <para></para>
81     /// </summary>
82     public int MaximumRepeatCount
83     {
84         [MethodImpl(MethodImplOptions.AggressiveInlining)]
85         get;
86         [MethodImpl(MethodImplOptions.AggressiveInlining)]
87         set;
88     }
89
90     /// <summary>
91     /// <para>
92     /// Initializes a new <see cref="SubstitutionRule"/> instance.
93     /// </para>
94     /// <para></para>
95     /// </summary>
96     /// <param name="matchPattern">
97     /// <para>A match pattern.</para>

```



```

98     /// <para></para>
99     /// </param>
100    /// <param name="substitutionPattern">
101    /// <para>A substitution pattern.</para>
102    /// <para></para>
103    /// </param>
104    /// <param name="maximumRepeatCount">
105    /// <para>A maximum repeat count.</para>
106    /// <para></para>
107    /// </param>
108    /// <param name="matchPatternOptions">
109    /// <para>A match pattern options.</para>
110    /// <para></para>
111    /// </param>
112    /// <param name="matchTimeout">
113    /// <para>A match timeout.</para>
114    /// <para></para>
115    /// </param>
116    [MethodImpl(MethodImplOptions.AggressiveInlining)]
117    public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
        ↳ maximumRepeatCount, RegexOptions? matchPatternOptions, TimeSpan? matchTimeout)
118    {
119        MatchPattern = matchPattern;
120        SubstitutionPattern = substitutionPattern;
121        MaximumRepeatCount = maximumRepeatCount;
122        OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
        ↳ matchTimeout ?? matchPattern.MatchTimeout);
123    }
124
125    /// <summary>
126    /// <para>
127    /// Initializes a new <see cref="SubstitutionRule"/> instance.
128    /// </para>
129    /// <para></para>
130    /// </summary>
131    /// <param name="matchPattern">
132    /// <para>A match pattern.</para>
133    /// <para></para>
134    /// </param>
135    /// <param name="substitutionPattern">
136    /// <para>A substitution pattern.</para>
137    /// <para></para>
138    /// </param>
139    /// <param name="maximumRepeatCount">
140    /// <para>A maximum repeat count.</para>
141    /// <para></para>
142    /// </param>
143    /// <param name="useDefaultOptions">
144    /// <para>A use default options.</para>
145    /// <para></para>
146    /// </param>
147    [MethodImpl(MethodImplOptions.AggressiveInlining)]
148    public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
        ↳ maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
        ↳ substitutionPattern, maximumRepeatCount, useDefaultOptions ?
        ↳ DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
        ↳ DefaultMatchTimeout : (TimeSpan?)null) { }
149
150    /// <summary>
151    /// <para>
152    /// Initializes a new <see cref="SubstitutionRule"/> instance.
153    /// </para>
154    /// <para></para>
155    /// </summary>
156    /// <param name="matchPattern">
157    /// <para>A match pattern.</para>
158    /// <para></para>
159    /// </param>
160    /// <param name="substitutionPattern">
161    /// <para>A substitution pattern.</para>
162    /// <para></para>
163    /// </param>
164    /// <param name="maximumRepeatCount">
165    /// <para>A maximum repeat count.</para>
166    /// <para></para>
167    /// </param>
168    [MethodImpl(MethodImplOptions.AggressiveInlining)]

```

```

169 public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
    ↳ maximumRepeatCount) : this(matchPattern, substitutionPattern, maximumRepeatCount,
    ↳ true) { }
170
171 /// <summary>
172 /// <para>
173 /// Initializes a new <see cref="SubstitutionRule"/> instance.
174 /// </para>
175 /// <para></para>
176 /// </summary>
177 /// <param name="matchPattern">
178 /// <para>A match pattern.</para>
179 /// <para></para>
180 /// </param>
181 /// <param name="substitutionPattern">
182 /// <para>A substitution pattern.</para>
183 /// <para></para>
184 /// </param>
185 [MethodImpl(MethodImplOptions.AggressiveInlining)]
186 public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
    ↳ this(matchPattern, substitutionPattern, 0) { }
187
188 [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 public static implicit operator SubstitutionRule(ValueTuple<string, string> tuple) =>
    ↳ new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2);
190
191 [MethodImpl(MethodImplOptions.AggressiveInlining)]
192 public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
    ↳ SubstitutionRule(tuple.Item1, tuple.Item2);
193
194 [MethodImpl(MethodImplOptions.AggressiveInlining)]
195 public static implicit operator SubstitutionRule(ValueTuple<string, string, int> tuple)
    ↳ => new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2, tuple.Item3);
196
197 [MethodImpl(MethodImplOptions.AggressiveInlining)]
198 public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
    ↳ => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);
199
200 /// <summary>
201 /// <para>
202 /// Overrides the match pattern options using the specified options.
203 /// </para>
204 /// <para></para>
205 /// </summary>
206 /// <param name="options">
207 /// <para>The options.</para>
208 /// <para></para>
209 /// </param>
210 /// <param name="matchTimeout">
211 /// <para>The match timeout.</para>
212 /// <para></para>
213 /// </param>
214 [MethodImpl(MethodImplOptions.AggressiveInlining)]
215 public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
    ↳ MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);
216
217 /// <summary>
218 /// <para>
219 /// Overrides the path pattern options using the specified options.
220 /// </para>
221 /// <para></para>
222 /// </summary>
223 /// <param name="options">
224 /// <para>The options.</para>
225 /// <para></para>
226 /// </param>
227 /// <param name="matchTimeout">
228 /// <para>The match timeout.</para>
229 /// <para></para>
230 /// </param>
231 [MethodImpl(MethodImplOptions.AggressiveInlining)]
232 public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
    ↳ PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
233
234 /// <summary>
235 /// <para>
236 /// Returns the string.

```

```

237     /// </para>
238     /// <para></para>
239     /// </summary>
240     /// <returns>
241     /// <para>The string</para>
242     /// <para></para>
243     /// </returns>
244     [MethodImpl(MethodImplOptions.AggressiveInlining)]
245     public override string ToString()
246     {
247         var sb = new StringBuilder();
248         sb.Append('');
249         sb.Append(MatchPattern.ToString());
250         sb.Append('');
251         sb.Append(" -> ");
252         sb.Append('');
253         sb.Append(SubstitutionPattern);
254         sb.Append('');
255         if (PathPattern != null)
256         {
257             sb.Append(" on files ");
258             sb.Append('');
259             sb.Append(PathPattern.ToString());
260             sb.Append('');
261         }
262         if (MaximumRepeatCount > 0)
263         {
264             if (MaximumRepeatCount >= int.MaxValue)
265             {
266                 sb.Append(" repeated forever");
267             }
268             else
269             {
270                 sb.Append(" repeated up to ");
271                 sb.Append(MaximumRepeatCount);
272                 sb.Append(" times");
273             }
274         }
275         return sb.ToString();
276     }
277 }
278 }

```

### 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      /// <summary>
10     /// <para>
11     /// Represents the text stepped transformer.
12     /// </para>
13     /// <para></para>
14     /// </summary>
15     /// <seealso cref="ITransformer"/>
16     public class TextSteppedTransformer : ITransformer
17     {
18         /// <summary>
19         /// <para>
20         /// Gets or sets the rules value.
21         /// </para>
22         /// <para></para>
23         /// </summary>
24         public IList<ISubstitutionRule> Rules
25         {
26             [MethodImpl(MethodImplOptions.AggressiveInlining)]
27             get;
28             [MethodImpl(MethodImplOptions.AggressiveInlining)]
29             set;
30         }
31
32         /// <summary>
33         /// <para>
34         /// Gets or sets the text value.
35         /// </para>

```

```

36     /// <para></para>
37     /// </summary>
38     public string Text
39     {
40         [MethodImpl(MethodImplOptions.AggressiveInlining)]
41         get;
42         [MethodImpl(MethodImplOptions.AggressiveInlining)]
43         set;
44     }
45
46     /// <summary>
47     /// <para>
48     /// Gets or sets the current value.
49     /// </para>
50     /// <para></para>
51     /// </summary>
52     public int Current
53     {
54         [MethodImpl(MethodImplOptions.AggressiveInlining)]
55         get;
56         [MethodImpl(MethodImplOptions.AggressiveInlining)]
57         set;
58     }
59
60     /// <summary>
61     /// <para>
62     /// Initializes a new <see cref="TextSteppedTransformer"/> instance.
63     /// </para>
64     /// <para></para>
65     /// </summary>
66     /// <param name="rules">
67     /// <para>A rules.</para>
68     /// <para></para>
69     /// </param>
70     /// <param name="text">
71     /// <para>A text.</para>
72     /// <para></para>
73     /// </param>
74     /// <param name="current">
75     /// <para>A current.</para>
76     /// <para></para>
77     /// </param>
78     [MethodImpl(MethodImplOptions.AggressiveInlining)]
79     public TextSteppedTransformer(ICollection<ISubstitutionRule> rules, string text, int current)
80     {
81         => Reset(rules, text, current);
82     }
83
84     /// <summary>
85     /// <para>
86     /// Initializes a new <see cref="TextSteppedTransformer"/> instance.
87     /// </para>
88     /// <para></para>
89     /// </summary>
90     /// <param name="rules">
91     /// <para>A rules.</para>
92     /// <para></para>
93     /// </param>
94     /// <param name="text">
95     /// <para>A text.</para>
96     /// <para></para>
97     /// </param>
98     [MethodImpl(MethodImplOptions.AggressiveInlining)]
99     public TextSteppedTransformer(ICollection<ISubstitutionRule> rules, string text) =>
100     {
101         => Reset(rules, text);
102     }
103
104     /// <summary>
105     /// <para>
106     /// Initializes a new <see cref="TextSteppedTransformer"/> instance.
107     /// </para>
108     /// <para></para>
109     /// </summary>
110     /// <param name="rules">
111     /// <para>A rules.</para>
112     /// <para></para>
113     /// </param>
114     [MethodImpl(MethodImplOptions.AggressiveInlining)]
115     public TextSteppedTransformer(ICollection<ISubstitutionRule> rules) => Reset(rules);
116
117     /// <summary>

```

```

112     /// <para>
113     /// Initializes a new <see cref="TextSteppedTransformer"/> instance.
114     /// </para>
115     /// <para></para>
116     /// </summary>
117     [MethodImpl(MethodImplOptions.AggressiveInlining)]
118     public TextSteppedTransformer() => Reset();
119
120     /// <summary>
121     /// <para>
122     /// Resets the rules.
123     /// </para>
124     /// <para></para>
125     /// </summary>
126     /// <param name="rules">
127     /// <para>The rules.</para>
128     /// <para></para>
129     /// </param>
130     /// <param name="text">
131     /// <para>The text.</para>
132     /// <para></para>
133     /// </param>
134     /// <param name="current">
135     /// <para>The current.</para>
136     /// <para></para>
137     /// </param>
138     [MethodImpl(MethodImplOptions.AggressiveInlining)]
139     public void Reset(IList<ISubstitutionRule> rules, string text, int current)
140     {
141         Rules = rules;
142         Text = text;
143         Current = current;
144     }
145
146     /// <summary>
147     /// <para>
148     /// Resets the rules.
149     /// </para>
150     /// <para></para>
151     /// </summary>
152     /// <param name="rules">
153     /// <para>The rules.</para>
154     /// <para></para>
155     /// </param>
156     /// <param name="text">
157     /// <para>The text.</para>
158     /// <para></para>
159     /// </param>
160     [MethodImpl(MethodImplOptions.AggressiveInlining)]
161     public void Reset(IList<ISubstitutionRule> rules, string text) => Reset(rules, text, -1);
162
163     /// <summary>
164     /// <para>
165     /// Resets the rules.
166     /// </para>
167     /// <para></para>
168     /// </summary>
169     /// <param name="rules">
170     /// <para>The rules.</para>
171     /// <para></para>
172     /// </param>
173     [MethodImpl(MethodImplOptions.AggressiveInlining)]
174     public void Reset(IList<ISubstitutionRule> rules) => Reset(rules, "", -1);
175
176     /// <summary>
177     /// <para>
178     /// Resets the text.
179     /// </para>
180     /// <para></para>
181     /// </summary>
182     /// <param name="text">
183     /// <para>The text.</para>
184     /// <para></para>
185     /// </param>
186     [MethodImpl(MethodImplOptions.AggressiveInlining)]
187     public void Reset(string text) => Reset(Rules, text, -1);
188
189     /// <summary>

```

```

190     /// <para>
191     /// Resets this instance.
192     /// </para>
193     /// <para></para>
194     /// </summary>
195     [MethodImpl(MethodImplOptions.AggressiveInlining)]
196     public void Reset() => Reset(Array.Empty<ISubstitutionRule>(), "", -1);
197
198     /// <summary>
199     /// <para>
200     /// Determines whether this instance next.
201     /// </para>
202     /// <para></para>
203     /// </summary>
204     /// <returns>
205     /// <para>The bool</para>
206     /// <para></para>
207     /// </returns>
208     [MethodImpl(MethodImplOptions.AggressiveInlining)]
209     public bool Next()
210     {
211         var current = Current + 1;
212         if (current >= Rules.Count)
213         {
214             return false;
215         }
216         var rule = Rules[current];
217         var matchPattern = rule.MatchPattern;
218         var substitutionPattern = rule.SubstitutionPattern;
219         var maximumRepeatCount = rule.MaximumRepeatCount;
220         var replaceCount = 0;
221         var text = Text;
222         do
223         {
224             text = matchPattern.Replace(text, substitutionPattern);
225             replaceCount++;
226         }
227         while ((maximumRepeatCount == int.MaxValue || replaceCount <= maximumRepeatCount) &&
228             ↪ matchPattern.IsMatch(text));
229         Text = text;
230         Current = current;
231         return true;
232     }
233 }

```

#### 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     /// <summary>
9     /// <para>
10    /// Represents the text transformer.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    /// <seealso cref="ITextTransformer"/>
15    public class TextTransformer : ITextTransformer
16    {
17        /// <summary>
18        /// <para>
19        /// Gets or sets the rules value.
20        /// </para>
21        /// <para></para>
22        /// </summary>
23        public IList<ISubstitutionRule> Rules
24        {
25            [MethodImpl(MethodImplOptions.AggressiveInlining)]
26            get;
27            [MethodImpl(MethodImplOptions.AggressiveInlining)]
28            private set;
29        }
30
31        /// <summary>
32        /// <para>
33        /// Initializes a new <see cref="TextTransformer"/> instance.

```

```

34     /// </para>
35     /// <para></para>
36     /// </summary>
37     /// <param name="substitutionRules">
38     /// <para>A substitution rules.</para>
39     /// <para></para>
40     /// </param>
41     [MethodImpl(MethodImplOptions.AggressiveInlining)]
42     public TextTransformer(ICollection<ISubstitutionRule> substitutionRules)
43     {
44         Rules = substitutionRules;
45     }
46
47     /// <summary>
48     /// <para>
49     /// Transforms the source.
50     /// </para>
51     /// <para></para>
52     /// </summary>
53     /// <param name="source">
54     /// <para>The source.</para>
55     /// <para></para>
56     /// </param>
57     /// <returns>
58     /// <para>The string</para>
59     /// <para></para>
60     /// </returns>
61     [MethodImpl(MethodImplOptions.AggressiveInlining)]
62     public string Transform(string source)
63     {
64         var baseTrasformer = new TextSteppedTransformer(Rules);
65         baseTrasformer.Reset(source);
66         while (baseTrasformer.Next());
67         return baseTrasformer.Text;
68     }
69 }
70 }

```

## 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     /// <summary>
9     /// <para>
10    /// Represents the transformer cli.
11    /// </para>
12    /// <para></para>
13    /// </summary>
14    public class TransformerCLI
15    {
16        /// <summary>
17        /// <para>
18        /// The transformer.
19        /// </para>
20        /// <para></para>
21        /// </summary>
22        private readonly IFileTransformer _transformer;
23
24        /// <summary>
25        /// <para>
26        /// Initializes a new <see cref="TransformerCLI"/> instance.
27        /// </para>
28        /// <para></para>
29        /// </summary>
30        /// <param name="transformer">
31        /// <para>A transformer.</para>
32        /// <para></para>
33        /// </param>
34        [MethodImpl(MethodImplOptions.AggressiveInlining)]
35        public TransformerCLI(IFileTransformer transformer) => _transformer = transformer;
36
37        /// <summary>
38        /// <para>
39        /// Runs the args.
40        /// </para>

```

```

41     /// <para></para>
42     /// </summary>
43     /// <param name="args">
44     /// <para>The args.</para>
45     /// <para></para>
46     /// </param>
47     [MethodImpl(MethodImplOptions.AggressiveInlining)]
48     public void Run(string[] args)
49     {
50         var sourcePath = args.GetElementOrDefault(0);
51         var targetPath = args.GetElementOrDefault(1);
52         _transformer.Transform(sourcePath, targetPath);
53     }
54 }
55 }

```

## 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      /// <summary>
7      /// <para>
8      /// Represents the file transformer tests.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public class FileTransformerTests
13     {
14         /// <summary>
15         /// <para>
16         /// Tests that folder to folder transformation test.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         [Fact]
21         public void FolderToFolderTransformationTest()
22         {
23             var tempPath = Path.GetTempPath();
24             var sourceFolderPath = Path.Combine(tempPath,
25                 ↪ "FileTransformerTestsFolderToFolderTransformationTestSourceFolder");
26             var targetFolderPath = Path.Combine(tempPath,
27                 ↪ "FileTransformerTestsFolderToFolderTransformationTestTargetFolder");
28
29             var baseTransformer = new TextTransformer(new SubstitutionRule[]
30             {
31                 ("a", "b"),
32                 ("b", "c")
33             });
34             var fileTransformer = new FileTransformer(baseTransformer, ".cs", ".cpp");
35
36             // Delete before creation (if previous test failed)
37             if (Directory.Exists(sourceFolderPath))
38             {
39                 Directory.Delete(sourceFolderPath, true);
40             }
41             if (Directory.Exists(targetFolderPath))
42             {
43                 Directory.Delete(targetFolderPath, true);
44             }
45
46             Directory.CreateDirectory(sourceFolderPath);
47             Directory.CreateDirectory(targetFolderPath);
48
49             File.WriteAllText(Path.Combine(sourceFolderPath, "a.cs"), "a a a");
50             var aFolderPath = Path.Combine(sourceFolderPath, "A");
51             Directory.CreateDirectory(aFolderPath);
52             Directory.CreateDirectory(Path.Combine(sourceFolderPath, "B"));
53             File.WriteAllText(Path.Combine(aFolderPath, "b.cs"), "b b b");
54             File.WriteAllText(Path.Combine(sourceFolderPath, "x.txt"), "should not be
55                 ↪ translated");
56
57             fileTransformer.Transform(sourceFolderPath,
58                 ↪ $"{targetFolderPath}{Path.DirectorySeparatorChar}");
59
60             var aCppFile = Path.Combine(targetFolderPath, "a.cpp");
61             Assert.True(File.Exists(aCppFile));

```



```

58     Assert.Equal("c c c", File.ReadAllText(aCppFile));
59     Assert.True(Directory.Exists(Path.Combine(targetFolderPath, "A")));
60     Assert.False(Directory.Exists(Path.Combine(targetFolderPath, "B")));
61     var bCppFile = Path.Combine(targetFolderPath, "A", "b.cpp");
62     Assert.True(File.Exists(bCppFile));
63     Assert.Equal("c c c", File.ReadAllText(bCppFile));
64     Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.txt")));
65     Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.cpp")));
66
67     Directory.Delete(sourceFolderPath, true);
68     Directory.Delete(targetFolderPath, true);
69 }
70 }
71 }

```

### 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      /// <summary>
7      /// <para>
8      /// Represents the markov algorithms tests.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public class MarkovAlgorithmsTests
13     {
14         /// <remarks>
15         /// Example is from https://en.wikipedia.org/wiki/Markov_algorithm.
16         /// </remarks>
17         [Fact]
18         public void BinaryToUnaryNumbersTest()
19         {
20             var rules = new SubstitutionRule[]
21             {
22                 ("1", "0|", int.MaxValue), // "1" -> "0|" repeated forever
23                 // | symbol should be escaped for regular expression pattern, but not in the
24                 // ↪ substitution pattern
25                 ("0", "0||", int.MaxValue), // "0" -> "0||" repeated forever
26                 ("0", "", int.MaxValue), // "0" -> "" repeated forever
27             };
28             var transformer = new TextTransformer(rules);
29             var input = "101";
30             var expectedOutput = "||||";
31             var output = transformer.Transform(input);
32             Assert.Equal(expectedOutput, output);
33         }
34     }
35 }

```

### 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      /// <summary>
7      /// <para>
8      /// Represents the substitution rule tests.
9      /// </para>
10     /// <para></para>
11     /// </summary>
12     public class SubstitutionRuleTests
13     {
14         /// <summary>
15         /// <para>
16         /// Tests that options override test.
17         /// </para>
18         /// <para></para>
19         /// </summary>
20         [Fact]
21         public void OptionsOverrideTest()
22         {
23             SubstitutionRule rule = (new Regex(@"^s*?#\pragma[\sa-zA-Z0-9\./]+$"), "", 0);
24             Assert.Equal(RegexOptions.Compiled | RegexOptions.Multiline,
25                 ↪ rule.MatchPattern.Options);
26         }
27     }
28 }

```

```

25     }
26 }
27 }

```

## 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      /// <summary>
9      /// <para>
10     /// Represents the text transformer tests.
11     /// </para>
12     /// <para></para>
13     /// </summary>
14     public class TextTransformerTests
15     {
16         /// <summary>
17         /// <para>
18         /// Tests that debug output test.
19         /// </para>
20         /// <para></para>
21         /// </summary>
22         [Fact]
23         public void DebugOutputTest()
24         {
25             var sourceText = "aaaa";
26             var firstStepReferenceText = "bbbb";
27             var secondStepReferenceText = "cccc";
28
29             var transformer = new TextTransformer(new SubstitutionRule[] {
30                 (new Regex("a"), "b"),
31                 (new Regex("b"), "c")
32             });
33
34             var steps = transformer.GetSteps(sourceText);
35
36             Assert.Equal(2, steps.Count);
37             Assert.Equal(firstStepReferenceText, steps[0]);
38             Assert.Equal(secondStepReferenceText, steps[1]);
39         }
40
41         /// <summary>
42         /// <para>
43         /// Tests that debug files output test.
44         /// </para>
45         /// <para></para>
46         /// </summary>
47         [Fact]
48         public void DebugFilesOutputTest()
49         {
50             var sourceText = "aaaa";
51             var firstStepReferenceText = "bbbb";
52             var secondStepReferenceText = "cccc";
53
54             var transformer = new TextTransformer(new SubstitutionRule[] {
55                 (new Regex("a"), "b"),
56                 (new Regex("b"), "c")
57             });
58
59             var targetFilename = Path.GetTempFileName();
60
61             transformer.WriteStepsToFiles(sourceText, $"{targetFilename}.txt",
62                 ↪ skipFilesWithNoChanges: false);
63
64             CheckAndCleanUpTwoRulesFiles(firstStepReferenceText, secondStepReferenceText,
65                 ↪ transformer, targetFilename);
66         }
67
68         /// <summary>
69         /// <para>
70         /// Checks the and clean up two rules files using the specified first step reference
71         ↪ text.
72         /// </para>
73         /// <para></para>
74         /// </summary>

```

```

72     /// <param name="firstStepReferenceText">
73     /// <para>The first step reference text.</para>
74     /// </para>
75     /// </param>
76     /// <param name="secondStepReferenceText">
77     /// <para>The second step reference text.</para>
78     /// </para>
79     /// </param>
80     /// <param name="transformer">
81     /// <para>The transformer.</para>
82     /// </para>
83     /// </param>
84     /// <param name="targetFilename">
85     /// <para>The target filename.</para>
86     /// </para>
87     /// </param>
88     private static void CheckAndCleanUpTwoRulesFiles(string firstStepReferenceText, string
    ↪ secondStepReferenceText, TextTransformer transformer, string targetFilename)
89     {
90         var firstStepReferenceFilename = $"{targetFilename}.0.txt";
91         var firstStepRuleFilename = $"{targetFilename}.0.rule.txt";
92         var secondStepReferenceFilename = $"{targetFilename}.1.txt";
93         var secondStepRuleFilename = $"{targetFilename}.1.rule.txt";
94
95         Assert.True(File.Exists(firstStepReferenceFilename));
96         Assert.True(File.Exists(firstStepRuleFilename));
97         Assert.True(File.Exists(secondStepReferenceFilename));
98         Assert.True(File.Exists(secondStepRuleFilename));
99
100        Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
    ↪ Encoding.UTF8));
101        Assert.Equal(transformer.Rules[0].ToString(),
    ↪ File.ReadAllText(firstStepRuleFilename, Encoding.UTF8));
102        Assert.Equal(secondStepReferenceText, File.ReadAllText(secondStepReferenceFilename,
    ↪ Encoding.UTF8));
103        Assert.Equal(transformer.Rules[1].ToString(),
    ↪ File.ReadAllText(secondStepRuleFilename, Encoding.UTF8));
104
105        File.Delete(firstStepReferenceFilename);
106        File.Delete(firstStepRuleFilename);
107        File.Delete(secondStepReferenceFilename);
108        File.Delete(secondStepRuleFilename);
109    }
110
111    /// <summary>
112    /// <para>
113    /// Tests that files with no changes skipped test.
114    /// </para>
115    /// </para>
116    /// </summary>
117    [Fact]
118    public void FilesWithNoChangesSkippedTest()
119    {
120        var sourceText = "aaaa";
121        var firstStepReferenceText = "bbbb";
122        var thirdStepReferenceText = "cccc";
123
124        var transformer = new TextTransformer(new SubstitutionRule[] {
125            (new Regex("a"), "b"),
126            (new Regex("x"), "y"),
127            (new Regex("b"), "c")
128        });
129
130        var targetFilename = Path.GetTempFileName();
131
132        transformer.WriteStepsToFiles(sourceText, $"{targetFilename}.txt",
    ↪ skipFilesWithNoChanges: true);
133
134        CheckAndCleanUpThreeRulesFiles(firstStepReferenceText, thirdStepReferenceText,
    ↪ transformer, targetFilename);
135    }
136
137    /// <summary>
138    /// <para>
139    /// Checks the and clean up three rules files using the specified first step reference
    ↪ text.
140    /// </para>
141    /// </para>

```

```

142     /// </summary>
143     /// <param name="firstStepReferenceText">
144     /// <para>The first step reference text.</para>
145     /// <para></para>
146     /// </param>
147     /// <param name="thirdStepReferenceText">
148     /// <para>The third step reference text.</para>
149     /// <para></para>
150     /// </param>
151     /// <param name="transformer">
152     /// <para>The transformer.</para>
153     /// <para></para>
154     /// </param>
155     /// <param name="targetFilename">
156     /// <para>The target filename.</para>
157     /// <para></para>
158     /// </param>
159     private static void CheckAndCleanUpThreeRulesFiles(string firstStepReferenceText, string
    ↪ thirdStepReferenceText, TextTransformer transformer, string targetFilename)
160     {
161         var firstStepReferenceFilename = $"{targetFilename}.0.txt";
162         var firstStepRuleFilename = $"{targetFilename}.0.rule.txt";
163         var secondStepReferenceFilename = $"{targetFilename}.1.txt";
164         var secondStepRuleFilename = $"{targetFilename}.1.rule.txt";
165         var thirdStepReferenceFilename = $"{targetFilename}.2.txt";
166         var thirdStepRuleFilename = $"{targetFilename}.2.rule.txt";
167
168         Assert.True(File.Exists(firstStepReferenceFilename));
169         Assert.True(File.Exists(firstStepRuleFilename));
170         Assert.False(File.Exists(secondStepReferenceFilename));
171         Assert.False(File.Exists(secondStepRuleFilename));
172         Assert.True(File.Exists(thirdStepReferenceFilename));
173         Assert.True(File.Exists(thirdStepRuleFilename));
174
175         Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
    ↪ Encoding.UTF8));
176         Assert.Equal(transformer.Rules[0].ToString(),
    ↪ File.ReadAllText(firstStepRuleFilename, Encoding.UTF8));
177         Assert.Equal(thirdStepReferenceText, File.ReadAllText(thirdStepReferenceFilename,
    ↪ Encoding.UTF8));
178         Assert.Equal(transformer.Rules[2].ToString(),
    ↪ File.ReadAllText(thirdStepRuleFilename, Encoding.UTF8));
179
180         File.Delete(firstStepReferenceFilename);
181         File.Delete(firstStepRuleFilename);
182         File.Delete(secondStepReferenceFilename);
183         File.Delete(secondStepRuleFilename);
184         File.Delete(thirdStepReferenceFilename);
185         File.Delete(thirdStepRuleFilename);
186     }
187
188     /// <summary>
189     /// <para>
190     /// Tests that debug output using transformers generation test.
191     /// </para>
192     /// <para></para>
193     /// </summary>
194     [Fact]
195     public void DebugOutputUsingTransformersGenerationTest()
196     {
197         var sourceText = "aaaa";
198         var firstStepReferenceText = "bbbb";
199         var secondStepReferenceText = "cccc";
200
201         var transformer = new TextTransformer(new SubstitutionRule[] {
202             (new Regex("a"), "b"),
203             (new Regex("b"), "c")
204         });
205
206         var steps =
    ↪ transformer.GenerateTransformersForEachRule().TransformWithAll(sourceText);
207
208         Assert.Equal(2, steps.Count);
209         Assert.Equal(firstStepReferenceText, steps[0]);
210         Assert.Equal(secondStepReferenceText, steps[1]);
211     }
212
213     /// <summary>

```

```

214 /// <para>
215 /// Tests that debug files output using transformers generation test.
216 /// </para>
217 /// <para></para>
218 /// </summary>
219 [Fact]
220 public void DebugFilesOutputUsingTransformersGenerationTest()
221 {
222     var sourceText = "aaaa";
223     var firstStepReferenceText = "bbbb";
224     var secondStepReferenceText = "cccc";
225
226     var transformer = new TextTransformer(new SubstitutionRule[] {
227         (new Regex("a"), "b"),
228         (new Regex("b"), "c")
229     });
230
231     var targetFilename = Path.GetTempFileName();
232
233     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
234         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: false);
235
236     CheckAndCleanUpTwoRulesFiles(firstStepReferenceText, secondStepReferenceText,
237         ↪ transformer, targetFilename);
238 }
239
240 /// <summary>
241 /// <para>
242 /// Tests that files with no changes skipped when using transformers generation test.
243 /// </para>
244 /// <para></para>
245 /// </summary>
246 [Fact]
247 public void FilesWithNoChangesSkippedWhenUsingTransformersGenerationTest()
248 {
249     var sourceText = "aaaa";
250     var firstStepReferenceText = "bbbb";
251     var thirdStepReferenceText = "cccc";
252
253     var transformer = new TextTransformer(new SubstitutionRule[] {
254         (new Regex("a"), "b"),
255         (new Regex("x"), "y"),
256         (new Regex("b"), "c")
257     });
258
259     var targetFilename = Path.GetTempFileName();
260
261     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
262         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: true);
263
264     CheckAndCleanUpThreeRulesFiles(firstStepReferenceText, thirdStepReferenceText,
265         ↪ transformer, targetFilename);
266 }
267 }

```

## Index

- ./csharp/Platform.RegularExpressions.Transformer.Tests/FileTransformerTests.cs, 24
- ./csharp/Platform.RegularExpressions.Transformer.Tests/MarkovAlgorithmsTests.cs, 25
- ./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs, 25
- ./csharp/Platform.RegularExpressions.Transformer.Tests/TextTransformerTests.cs, 26
- ./csharp/Platform.RegularExpressions.Transformer/FileTransformer.cs, 1
- ./csharp/Platform.RegularExpressions.Transformer/IFileTransformer.cs, 6
- ./csharp/Platform.RegularExpressions.Transformer/ISubstitutionRule.cs, 7
- ./csharp/Platform.RegularExpressions.Transformer/ITextTransformer.cs, 8
- ./csharp/Platform.RegularExpressions.Transformer/ITextTransformerExtensions.cs, 8
- ./csharp/Platform.RegularExpressions.Transformer/ITextTransformersListExtensions.cs, 10
- ./csharp/Platform.RegularExpressions.Transformer/ITransformer.cs, 11
- ./csharp/Platform.RegularExpressions.Transformer/LoggingFileTransformer.cs, 12
- ./csharp/Platform.RegularExpressions.Transformer/RegexExtensions.cs, 12
- ./csharp/Platform.RegularExpressions.Transformer/Steps.cs, 13
- ./csharp/Platform.RegularExpressions.Transformer/StringExtensions.cs, 14
- ./csharp/Platform.RegularExpressions.Transformer/SubstitutionRule.cs, 15
- ./csharp/Platform.RegularExpressions.Transformer/TextSteppedTransformer.cs, 19
- ./csharp/Platform.RegularExpressions.Transformer/TextTransformer.cs, 22
- ./csharp/Platform.RegularExpressions.Transformer/TransformerCLI.cs, 23