

# LinksPlatform's Platform.RegularExpressions.Transformer Class Library

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

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

```

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

```

```

150     }
151 }
152
153 [MethodImpl(MethodImplOptions.AggressiveInlining)]
154 private static void EnsureSourceFileExists(string sourcePath)
155 {
156     if (!File.Exists(sourcePath))
157     {
158         throw new FileNotFoundException("Source file does not exists.", sourcePath);
159     }
160 }
161
162 [MethodImpl(MethodImplOptions.AggressiveInlining)]
163 private static string NormalizePath(string path) => Path.GetFullPath(path).TrimEnd(new[]
164     ↳ { Path.DirectorySeparatorChar, Path.AltDirectorySeparatorChar });
165
166 [MethodImpl(MethodImplOptions.AggressiveInlining)]
167 private static string GetRelativePath(string rootPath, string fullPath)
168 {
169     rootPath = NormalizePath(rootPath);
170     fullPath = NormalizePath(fullPath);
171     if (!fullPath.StartsWith(rootPath))
172     {
173         throw new Exception("Could not find rootPath in fullPath when calculating
174             ↳ relative path.");
175     }
176     return fullPath.Substring(rootPath.Length + 1);
177 }
178
179 [MethodImpl(MethodImplOptions.AggressiveInlining)]
180 private static void EnsureDirectoryIsCreated(string targetPath) =>
181     ↳ Directory.CreateDirectory(Path.GetDirectoryName(targetPath));
182
183 [MethodImpl(MethodImplOptions.AggressiveInlining)]
184 private static bool DirectoryExists(string path) => Directory.Exists(path) &&
185     ↳ File.GetAttributes(path).HasFlag(FileAttributes.Directory);
186
187 [MethodImpl(MethodImplOptions.AggressiveInlining)]
188 private static bool LooksLikeDirectoryPath(string path) =>
189     ↳ path.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
190     ↳ path.EndsWith(Path.AltDirectorySeparatorChar.ToString());
191 }
192 }

```

## 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.Collections.Generic;
2 using System.Linq;
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 static class ITextTransformerExtensions
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public static IList<ITextTransformer> GenerateTransformersForEachRule(this
13             ↪ ITextTransformer transformer)
14         {
15             var transformers = new List<ITextTransformer>();
16             for (int i = 1; i <= transformer.Rules.Count; i++)
17             {
18                 transformers.Add(new TextTransformer(transformer.Rules.Take(i).ToList()));
19             }
20             return transformers;
21         }
22
23         [MethodImpl(MethodImplOptions.AggressiveInlining)]
24         public static List<string> GetSteps(this ITextTransformer transformer, string
25             ↪ sourceText) =>
26             ↪ transformer.GenerateTransformersForEachRule().TransformWithAll(sourceText);
27
28         [MethodImpl(MethodImplOptions.AggressiveInlining)]
29         public static void WriteStepsToFiles(this ITextTransformer transformer, string
30             ↪ sourceText, string targetPath, bool skipFilesWithNoChanges) =>
31             ↪ transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
32             ↪ targetPath, skipFilesWithNoChanges);
33     }
34 }

```

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

```

1 using System.IO;
2 using System.Collections.Generic;
3 using System.Text;
4 using System.Runtime.CompilerServices;
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 List<string> TransformWithAll(this IList<ITextTransformer> transformers,
14             ↪ string source)
15         {
16             var strings = new List<string>();
17             if (transformers.Count > 0)
18             {
19                 for (int i = 0; i < transformers.Count; i++)
20                 {
21                     strings.Add(transformers[i].Transform(source));
22                 }
23             }
24             return strings;
25         }
26
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public static void TransformWithAllToFiles(this IList<ITextTransformer> transformers,
29             ↪ string sourceText, string targetPath, bool skipFilesWithNoChanges)
30         {
31             if (transformers.Count > 0)
32             {
33                 var directoryName = Path.GetDirectoryName(targetPath);
34                 var targetFilename = Path.GetFileNameWithoutExtension(targetPath);
35                 var targetExtension = Path.GetExtension(targetPath);
36                 var lastText = "";
37                 for (int i = 0; i < transformers.Count; i++)
38                 {
39                     var transformationOutput = transformers[i].Transform(sourceText);
40                     if (!(skipFilesWithNoChanges && string.Equals(lastText,
41                         ↪ transformationOutput)))
42                     {
43                         lastText = transformationOutput;
44                         File.WriteAllText(Path.Combine(directoryName,
45                             ↪ $"{targetFilename}.{i}{targetExtension}"), transformationOutput,
46                             ↪ Encoding.UTF8);
47                     }
48                 }
49             }
50         }
51     }
52 }

```

### 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 }

```

```

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

```

### 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
13            ↪ matchTimeout)
14        {
15            if (regex == null)
16            {
17                return null;
18            }
19            return new Regex(regex.ToString(), options, matchTimeout);
20        }
21    }
22 }

```

### 1.10 ./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)]

```

```

42     get;
43     [MethodImpl(MethodImplOptions.AggressiveInlining)]
44     set;
45 }
46
47 [MethodImpl(MethodImplOptions.AggressiveInlining)]
48 public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
49     ↳ maximumRepeatCount, RegexOptions? matchPatternOptions, TimeSpan? matchTimeout)
50 {
51     MatchPattern = matchPattern;
52     SubstitutionPattern = substitutionPattern;
53     MaximumRepeatCount = maximumRepeatCount;
54     OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
55     ↳ matchTimeout ?? matchPattern.MatchTimeout);
56 }
57
58 [MethodImpl(MethodImplOptions.AggressiveInlining)]
59 public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
60     ↳ maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
61     ↳ substitutionPattern, maximumRepeatCount, useDefaultOptions ?
62     ↳ DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
63     ↳ DefaultMatchTimeout : (TimeSpan?)null) { }
64
65 [MethodImpl(MethodImplOptions.AggressiveInlining)]
66 public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
67     ↳ maximumRepeatCount) : this(matchPattern, substitutionPattern, maximumRepeatCount,
68     ↳ true) { }
69
70 [MethodImpl(MethodImplOptions.AggressiveInlining)]
71 public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
72     ↳ this(matchPattern, substitutionPattern, 0) { }
73
74 [MethodImpl(MethodImplOptions.AggressiveInlining)]
75 public static implicit operator SubstitutionRule(ValueTuple<string, string> tuple) =>
76     ↳ new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2);
77
78 [MethodImpl(MethodImplOptions.AggressiveInlining)]
79 public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
80     ↳ SubstitutionRule(tuple.Item1, tuple.Item2);
81
82 [MethodImpl(MethodImplOptions.AggressiveInlining)]
83 public static implicit operator SubstitutionRule(ValueTuple<string, string, int> tuple)
84     ↳ => new SubstitutionRule(new Regex(tuple.Item1), tuple.Item2, tuple.Item3);
85
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
88     ↳ => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);
89
90 [MethodImpl(MethodImplOptions.AggressiveInlining)]
91 public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
92     ↳ MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);
93
94 [MethodImpl(MethodImplOptions.AggressiveInlining)]
95 public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
96     ↳ PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
97
98 [MethodImpl(MethodImplOptions.AggressiveInlining)]
99 public override string ToString()
100 {
101     var sb = new StringBuilder();
102     sb.Append('');
103     sb.Append(MatchPattern.ToString());
104     sb.Append('');
105     sb.Append(" -> ");
106     sb.Append('');
107     sb.Append(SubstitutionPattern);
108     sb.Append('');
109     if (PathPattern != null)
110     {
111         sb.Append(" on files ");
112         sb.Append('');
113         sb.Append(PathPattern.ToString());
114         sb.Append('');
115     }
116     if (MaximumRepeatCount > 0)
117     {
118         if (MaximumRepeatCount >= int.MaxValue)
119         {

```

```

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 }

```

### 1.11 ./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) => Rules =
20             substitutionRules;
21
22         [MethodImpl(MethodImplOptions.AggressiveInlining)]
23         public string Transform(string source)
24         {
25             var current = source;
26             for (var i = 0; i < Rules.Count; i++)
27             {
28                 var rule = Rules[i];
29                 var matchPattern = rule.MatchPattern;
30                 var substitutionPattern = rule.SubstitutionPattern;
31                 var maximumRepeatCount = rule.MaximumRepeatCount;
32                 var replaceCount = 0;
33                 do
34                 {
35                     current = matchPattern.Replace(current, substitutionPattern);
36                     replaceCount++;
37                     if (maximumRepeatCount < int.MaxValue && replaceCount > maximumRepeatCount)
38                     {
39                         break;
40                     }
41                 } while (matchPattern.IsMatch(current));
42             }
43             return current;
44         }
45     }
46 }

```

### 1.12 ./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         }
19     }
20 }

```



```

17     {
18         var sourcePath = args.GetElementOrDefault(0);
19         var targetPath = args.GetElementOrDefault(1);
20         _transformer.Transform(sourcePath, targetPath);
21     }
22 }
23 }

```

### 1.13 ./csharp/Platform.RegularExpressions.Transformer.Tests/FileTransformerTests.cs

```

1  using System.IO;
2  using Xunit;
3
4  namespace Platform.RegularExpressions.Transformer.Tests
5  {
6      public class FileTransformerTests
7      {
8          [Fact]
9          public void FolderToFolderTransformationTest()
10         {
11             var tempPath = Path.GetTempPath();
12             var sourceFolderPath = Path.Combine(tempPath,
13                 ↪ "FileTransformerTestsFolderToFolderTransformationTestSourceFolder");
14             var targetFolderPath = Path.Combine(tempPath,
15                 ↪ "FileTransformerTestsFolderToFolderTransformationTestTargetFolder");
16
17             var baseTransformer = new TextTransformer(new SubstitutionRule[]
18             {
19                 ("a", "b"),
20                 ("b", "c")
21             });
22             var fileTransformer = new FileTransformer(baseTransformer, ".cs", ".cpp");
23
24             // Delete before creation (if previous test failed)
25             if (Directory.Exists(sourceFolderPath))
26             {
27                 Directory.Delete(sourceFolderPath, true);
28             }
29             if (Directory.Exists(targetFolderPath))
30             {
31                 Directory.Delete(targetFolderPath, true);
32             }
33
34             Directory.CreateDirectory(sourceFolderPath);
35             Directory.CreateDirectory(targetFolderPath);
36
37             File.WriteAllText(Path.Combine(sourceFolderPath, "a.cs"), "a a a");
38             var aFolderPath = Path.Combine(sourceFolderPath, "A");
39             Directory.CreateDirectory(aFolderPath);
40             File.WriteAllText(Path.Combine(aFolderPath, "b.cs"), "b b b");
41             File.WriteAllText(Path.Combine(sourceFolderPath, "x.txt"), "should not be
42                 ↪ translated");
43
44             fileTransformer.Transform(sourceFolderPath,
45                 ↪ $"{targetFolderPath}{Path.DirectorySeparatorChar}");
46
47             var aCppFile = Path.Combine(targetFolderPath, "a.cpp");
48             Assert.True(File.Exists(aCppFile));
49             Assert.Equal("c c c", File.ReadAllText(aCppFile));
50             Assert.True(Directory.Exists(Path.Combine(targetFolderPath, "A")));
51             var bCppFile = Path.Combine(targetFolderPath, "A", "b.cpp");
52             Assert.True(File.Exists(bCppFile));
53             Assert.Equal("c c c", File.ReadAllText(bCppFile));
54             Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.txt")));
55             Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.cpp")));
56
57             Directory.Delete(sourceFolderPath, true);
58             Directory.Delete(targetFolderPath, true);
59         }
60     }
61 }

```

### 1.14 ./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", "", int.MaxValue), // "0" -> "" repeated forever
20        };
21        var transformer = new TextTransformer(rules);
22        var input = "101";
23        var expectedOutput = "||||";
24        var output = transformer.Transform(input);
25        Assert.Equal(expectedOutput, output);
26    }
27 }
28 }

```

#### 1.15 ./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs

```

1 using System.Text.RegularExpressions;
2 using Xunit;
3
4 namespace Platform.RegularExpressions.Transformer.Tests
5 {
6     public class SubstitutionRuleTests
7     {
8         [Fact]
9         public void OptionsOverrideTest()
10        {
11            SubstitutionRule rule = (new Regex(@"^s*?\#pragma[\sa-zA-Z0-9\/]+$"), "", 0);
12            Assert.Equal(RegexOptions.Compiled | RegexOptions.Multiline,
13                rule.MatchPattern.Options);
14        }
15    }
16 }

```

#### 1.16 ./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     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
47     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
48
49     Assert.True(File.Exists(firstStepReferenceFilename));
50     Assert.True(File.Exists(secondStepReferenceFilename));
51
52     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
53         ↪ Encoding.UTF8));
54     Assert.Equal(secondStepReferenceText, File.ReadAllText(secondStepReferenceFilename,
55         ↪ Encoding.UTF8));
56
57     File.Delete(firstStepReferenceFilename);
58     File.Delete(secondStepReferenceFilename);
59 }
60
61 [Fact]
62 public void FilesWithNoChangesSkippedTest()
63 {
64     var sourceText = "aaaa";
65     var firstStepReferenceText = "bbbb";
66     var thirdStepReferenceText = "cccc";
67
68     var transformer = new TextTransformer(new SubstitutionRule[] {
69         (new Regex("a"), "b"),
70         (new Regex("x"), "y"),
71         (new Regex("b"), "c")
72     });
73
74     var targetFilename = Path.GetTempFileName();
75
76     transformer.WriteStepsToFiles(sourceText, $"{targetFilename}.txt",
77         ↪ skipFilesWithNoChanges: true);
78
79     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
80     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
81     var thirdStepReferenceFilename = $"{targetFilename}.2.txt";
82
83     Assert.True(File.Exists(firstStepReferenceFilename));
84     Assert.False(File.Exists(secondStepReferenceFilename));
85     Assert.True(File.Exists(thirdStepReferenceFilename));
86
87     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
88         ↪ Encoding.UTF8));
89     Assert.Equal(thirdStepReferenceText, File.ReadAllText(thirdStepReferenceFilename,
90         ↪ Encoding.UTF8));
91
92     File.Delete(firstStepReferenceFilename);
93     File.Delete(secondStepReferenceFilename);
94     File.Delete(thirdStepReferenceFilename);
95 }
96 }
97
98 }
```

## Index

./csharp/Platform.RegularExpressions.Transformer.Tests/FileTransformerTests.cs, 9  
./csharp/Platform.RegularExpressions.Transformer.Tests/MarkovAlgorithmsTests.cs, 9  
./csharp/Platform.RegularExpressions.Transformer.Tests/SubstitutionRuleTests.cs, 10  
./csharp/Platform.RegularExpressions.Transformer.Tests/TextTransformerTests.cs, 10  
./csharp/Platform.RegularExpressions.Transformer/FileTransformer.cs, 1  
./csharp/Platform.RegularExpressions.Transformer/IFileTransformer.cs, 3  
./csharp/Platform.RegularExpressions.Transformer/ISubstitutionRule.cs, 3  
./csharp/Platform.RegularExpressions.Transformer/ITextTransformer.cs, 4  
./csharp/Platform.RegularExpressions.Transformer/ITextTransformerExtensions.cs, 4  
./csharp/Platform.RegularExpressions.Transformer/ITextTransformersListExtensions.cs, 4  
./csharp/Platform.RegularExpressions.Transformer/ITransformer.cs, 5  
./csharp/Platform.RegularExpressions.Transformer/LoggingFileTransformer.cs, 5  
./csharp/Platform.RegularExpressions.Transformer/RegexExtensions.cs, 6  
./csharp/Platform.RegularExpressions.Transformer/SubstitutionRule.cs, 6  
./csharp/Platform.RegularExpressions.Transformer/TextTransformer.cs, 8  
./csharp/Platform.RegularExpressions.Transformer/TransformerCLI.cs, 8