

LinksPlatform's Platform.RegularExpressions.Transformer Class Library

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

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

```

78     }
79     else
80     {
81         // Folder -> File
82         throw new NotSupportedException();
83     }
84 }
85
86 [MethodImpl(MethodImplOptions.AggressiveInlining)]
87 protected virtual void TransformFolder(string sourcePath, string targetPath)
88 {
89     if (CountFilesRecursively(sourcePath, SourceFileExtension) == 0)
90     {
91         return;
92     }
93     EnsureTargetDirectoryExists(targetPath);
94     var directories = Directory.GetDirectories(sourcePath);
95     for (var i = 0; i < directories.Length; i++)
96     {
97         var relativePath = GetRelativePath(sourcePath, directories[i]);
98         var newTargetPath = Path.Combine(targetPath, relativePath);
99         TransformFolder(directories[i], newTargetPath);
100     }
101     var files = Directory.GetFiles(sourcePath);
102     Parallel.For(0, files.Length, i =>
103     {
104         var file = files[i];
105         if (FileExtensionMatches(file, SourceFileExtension))
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 long CountFilesRecursively(string path, string extension)
138 {
139     var files = Directory.GetFiles(path);
140     var directories = Directory.GetDirectories(path);
141     var result = 0L;
142     for (var i = 0; i < directories.Length; i++)
143     {
144         result += CountFilesRecursively(directories[i], extension);
145     }
146     for (var i = 0; i < files.Length; i++)
147     {
148         if (FileExtensionMatches(files[i], extension))
149         {
150             result++;
151         }
152     }
153     return result;
154 }

```

```

152 [MethodImpl(MethodImplOptions.AggressiveInlining)]
153 private static bool FileExtensionMatches(string file, string extension) =>
    ↳ file.EndsWith(extension, StringComparison.OrdinalIgnoreCase);
154
155 [MethodImpl(MethodImplOptions.AggressiveInlining)]
156 private static void EnsureTargetFileDirectoryExists(string targetPath)
157 {
158     if (!File.Exists(targetPath))
159     {
160         EnsureDirectoryIsCreated(targetPath);
161     }
162 }
163
164 [MethodImpl(MethodImplOptions.AggressiveInlining)]
165 private static void EnsureTargetDirectoryExists(string targetPath) =>
    ↳ EnsureTargetDirectoryExists(targetPath, DirectoryExists(targetPath));
166
167 [MethodImpl(MethodImplOptions.AggressiveInlining)]
168 private static void EnsureTargetDirectoryExists(string targetPath, bool
    ↳ targetDirectoryExists)
169 {
170     if (!targetDirectoryExists)
171     {
172         Directory.CreateDirectory(targetPath);
173     }
174 }
175
176 [MethodImpl(MethodImplOptions.AggressiveInlining)]
177 private static void EnsureSourceFileExists(string sourcePath)
178 {
179     if (!File.Exists(sourcePath))
180     {
181         throw new FileNotFoundException("Source file does not exists.", sourcePath);
182     }
183 }
184
185 [MethodImpl(MethodImplOptions.AggressiveInlining)]
186 private static string NormalizePath(string path) => Path.GetFullPath(path).TrimEnd(new[]
    ↳ { Path.DirectorySeparatorChar, Path.AltDirectorySeparatorChar });
187
188 [MethodImpl(MethodImplOptions.AggressiveInlining)]
189 private static string GetRelativePath(string rootPath, string fullPath)
190 {
191     rootPath = NormalizePath(rootPath);
192     fullPath = NormalizePath(fullPath);
193     if (!fullPath.StartsWith(rootPath))
194     {
195         throw new Exception("Could not find rootPath in fullPath when calculating
            ↳ relative path.");
196     }
197     return fullPath.Substring(rootPath.Length + 1);
198 }
199
200 [MethodImpl(MethodImplOptions.AggressiveInlining)]
201 private static void EnsureDirectoryIsCreated(string targetPath) =>
    ↳ Directory.CreateDirectory(Path.GetDirectoryName(targetPath));
202
203 [MethodImpl(MethodImplOptions.AggressiveInlining)]
204 private static bool DirectoryExists(string path) => Directory.Exists(path) &&
    ↳ File.GetAttributes(path).HasFlag(FileAttributes.Directory);
205
206 [MethodImpl(MethodImplOptions.AggressiveInlining)]
207 private static bool LooksLikeDirectoryPath(string path) =>
    ↳ path.EndsWith(Path.DirectorySeparatorChar.ToString()) ||
    ↳ path.EndsWith(Path.AltDirectorySeparatorChar.ToString());
208 }
209 }

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.RegularExpressions.Transformer
6 {
7     public interface IFileTransformer : ITransformer
8     {
9         string SourceFileExtension

```

```

10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         get;
13     }
14
15     string TargetFileExtension
16     {
17         [MethodImpl(MethodImplOptions.AggressiveInlining)]
18         get;
19     }
20
21     [MethodImpl(MethodImplOptions.AggressiveInlining)]
22     void Transform(string sourcePath, string targetPath);
23 }
24 }

```

1.3 ./csharp/Platform.RegularExpressions.Transformer/ISubstitutionRule.cs

```

1 using System.Runtime.CompilerServices;
2 using System.Text.RegularExpressions;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.RegularExpressions.Transformer
7 {
8     public interface ISubstitutionRule
9     {
10         Regex MatchPattern
11         {
12             [MethodImpl(MethodImplOptions.AggressiveInlining)]
13             get;
14         }
15
16         string SubstitutionPattern
17         {
18             [MethodImpl(MethodImplOptions.AggressiveInlining)]
19             get;
20         }
21
22         int MaximumRepeatCount
23         {
24             [MethodImpl(MethodImplOptions.AggressiveInlining)]
25             get;
26         }
27     }
28 }

```

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

```

1 using System.Runtime.CompilerServices;
2
3 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
4
5 namespace Platform.RegularExpressions.Transformer
6 {
7     public interface ITextTransformer : ITransformer
8     {
9         [MethodImpl(MethodImplOptions.AggressiveInlining)]
10         string Transform(string sourceText);
11     }
12 }

```

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

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Runtime.CompilerServices;
5 using Platform.Collections;
6
7 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
8
9 namespace Platform.RegularExpressions.Transformer
10 {
11     public static class ITextTransformerExtensions
12     {
13         [MethodImpl(MethodImplOptions.AggressiveInlining)]
14         public static IList<ITextTransformer> GenerateTransformersForEachRule(this
15             ↪ ITextTransformer transformer)
16         {
17             var transformers = new List<ITextTransformer>();
18             for (int i = 1; i <= transformer.Rules.Count; i++)
19             {

```

```

19         transformers.Add(new TextTransformer(transformer.Rules.Take(i).ToList()));
20     }
21     return transformers;
22 }
23
24 [MethodImpl(MethodImplOptions.AggressiveInlining)]
25 public static IList<string> GetSteps(this ITextTransformer transformer, string
    ↪ sourceText)
26 {
27     if (transformer != null && !transformer.Rules.IsNullOrEmpty())
28     {
29         var steps = new List<string>();
30         var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
    ↪ sourceText);
31         while (steppedTransformer.Next())
32         {
33             steps.Add(steppedTransformer.Text);
34         }
35         return steps;
36     }
37     else
38     {
39         return Array.Empty<string>();
40     }
41 }
42
43 [MethodImpl(MethodImplOptions.AggressiveInlining)]
44 public static void WriteStepsToFiles(this ITextTransformer transformer, string
    ↪ sourceText, string targetPath, bool skipFilesWithNoChanges)
45 {
46     if (transformer != null && !transformer.Rules.IsNullOrEmpty())
47     {
48         targetPath.GetPathParts(out var directoryName, out var targetFilename, out var
    ↪ targetExtension);
49         var lastText = "";
50         var steppedTransformer = new TextSteppedTransformer(transformer.Rules,
    ↪ sourceText);
51         while (steppedTransformer.Next())
52         {
53             var newText = steppedTransformer.Text;
54             if (!(skipFilesWithNoChanges && string.Equals(lastText, newText)))
55             {
56                 lastText = newText;
57                 newText.WriteStepToFile(directoryName, targetFilename, targetExtension,
    ↪ steppedTransformer.Current);
58             }
59         }
60     }
61 }
62 }
63 }

```

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

```

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

```

```

26         return Array.Empty<string>();
27     }
28 }
29
30 [MethodImpl(MethodImplOptions.AggressiveInlining)]
31 public static void TransformWithAllToFiles(this IList<ITextTransformer> transformers,
32     ↪ string sourceText, string targetPath, bool skipFilesWithNoChanges)
33 {
34     if (!transformers.IsNullOrEmpty())
35     {
36         targetPath.GetPathParts(out var directoryName, out var targetFilename, out var
37             ↪ targetExtension);
38         var lastText = "";
39         for (int i = 0; i < transformers.Count; i++)
40         {
41             var transformationOutput = transformers[i].Transform(sourceText);
42             if (!(skipFilesWithNoChanges && string.Equals(lastText,
43                 ↪ transformationOutput)))
44             {
45                 lastText = transformationOutput;
46                 transformationOutput.WriteStepToFile(directoryName, targetFilename,
47                     ↪ targetExtension, i);
48             }
49         }
50     }
51 }
52
53 }
54
55 }
56
57 }
58
59 }
60
61 }
62
63 }
64
65 }
66
67 }
68
69 }
70
71 }
72
73 }
74
75 }
76
77 }
78
79 }
80
81 }
82
83 }
84
85 }
86
87 }
88
89 }
90
91 }
92
93 }
94
95 }
96
97 }
98
99 }
100
101 }
102
103 }
104
105 }
106
107 }
108
109 }
110
111 }
112
113 }
114
115 }
116
117 }
118
119 }
120
121 }
122
123 }
124
125 }
126
127 }
128
129 }
130
131 }
132
133 }
134
135 }
136
137 }
138
139 }
140
141 }
142
143 }
144
145 }
146
147 }
148
149 }
150
151 }
152
153 }
154
155 }
156
157 }
158
159 }
160
161 }
162
163 }
164
165 }
166
167 }
168
169 }
170
171 }
172
173 }
174
175 }
176
177 }
178
179 }
180
181 }
182
183 }
184
185 }
186
187 }
188
189 }
190
191 }
192
193 }
194
195 }
196
197 }
198
199 }
200
201 }
202
203 }
204
205 }
206
207 }
208
209 }
210
211 }
212
213 }
214
215 }
216
217 }
218
219 }
220
221 }
222
223 }
224
225 }
226
227 }
228
229 }
230
231 }
232
233 }
234
235 }
236
237 }
238
239 }
240
241 }
242
243 }
244
245 }
246
247 }
248
249 }
250
251 }
252
253 }
254
255 }
256
257 }
258
259 }
260
261 }
262
263 }
264
265 }
266
267 }
268
269 }
270
271 }
272
273 }
274
275 }
276
277 }
278
279 }
280
281 }
282
283 }
284
285 }
286
287 }
288
289 }
290
291 }
292
293 }
294
295 }
296
297 }
298
299 }
300
301 }
302
303 }
304
305 }
306
307 }
308
309 }
310
311 }
312
313 }
314
315 }
316
317 }
318
319 }
320
321 }
322
323 }
324
325 }
326
327 }
328
329 }
330
331 }
332
333 }
334
335 }
336
337 }
338
339 }
340
341 }
342
343 }
344
345 }
346
347 }
348
349 }
350
351 }
352
353 }
354
355 }
356
357 }
358
359 }
360
361 }
362
363 }
364
365 }
366
367 }
368
369 }
370
371 }
372
373 }
374
375 }
376
377 }
378
379 }
380
381 }
382
383 }
384
385 }
386
387 }
388
389 }
390
391 }
392
393 }
394
395 }
396
397 }
398
399 }
400
401 }
402
403 }
404
405 }
406
407 }
408
409 }
410
411 }
412
413 }
414
415 }
416
417 }
418
419 }
420
421 }
422
423 }
424
425 }
426
427 }
428
429 }
430
431 }
432
433 }
434
435 }
436
437 }
438
439 }
440
441 }
442
443 }
444
445 }
446
447 }
448
449 }
450
451 }
452
453 }
454
455 }
456
457 }
458
459 }
460
461 }
462
463 }
464
465 }
466
467 }
468
469 }
470
471 }
472
473 }
474
475 }
476
477 }
478
479 }
480
481 }
482
483 }
484
485 }
486
487 }
488
489 }
490
491 }
492
493 }
494
495 }
496
497 }
498
499 }
500
501 }
502
503 }
504
505 }
506
507 }
508
509 }
510
511 }
512
513 }
514
515 }
516
517 }
518
519 }
520
521 }
522
523 }
524
525 }
526
527 }
528
529 }
530
531 }
532
533 }
534
535 }
536
537 }
538
539 }
540
541 }
542
543 }
544
545 }
546
547 }
548
549 }
550
551 }
552
553 }
554
555 }
556
557 }
558
559 }
560
561 }
562
563 }
564
565 }
566
567 }
568
569 }
570
571 }
572
573 }
574
575 }
576
577 }
578
579 }
580
581 }
582
583 }
584
585 }
586
587 }
588
589 }
590
591 }
592
593 }
594
595 }
596
597 }
598
599 }
600
601 }
602
603 }
604
605 }
606
607 }
608
609 }
610
611 }
612
613 }
614
615 }
616
617 }
618
619 }
620
621 }
622
623 }
624
625 }
626
627 }
628
629 }
630
631 }
632
633 }
634
635 }
636
637 }
638
639 }
640
641 }
642
643 }
644
645 }
646
647 }
648
649 }
650
651 }
652
653 }
654
655 }
656
657 }
658
659 }
660
661 }
662
663 }
664
665 }
666
667 }
668
669 }
670
671 }
672
673 }
674
675 }
676
677 }
678
679 }
680
681 }
682
683 }
684
685 }
686
687 }
688
689 }
690
691 }
692
693 }
694
695 }
696
697 }
698
699 }
700
701 }
702
703 }
704
705 }
706
707 }
708
709 }
710
711 }
712
713 }
714
715 }
716
717 }
718
719 }
720
721 }
722
723 }
724
725 }
726
727 }
728
729 }
730
731 }
732
733 }
734
735 }
736
737 }
738
739 }
740
741 }
742
743 }
744
745 }
746
747 }
748
749 }
750
751 }
752
753 }
754
755 }
756
757 }
758
759 }
760
761 }
762
763 }
764
765 }
766
767 }
768
769 }
770
771 }
772
773 }
774
775 }
776
777 }
778
779 }
780
781 }
782
783 }
784
785 }
786
787 }
788
789 }
790
791 }
792
793 }
794
795 }
796
797 }
798
799 }
800
801 }
802
803 }
804
805 }
806
807 }
808
809 }
810
811 }
812
813 }
814
815 }
816
817 }
818
819 }
820
821 }
822
823 }
824
825 }
826
827 }
828
829 }
830
831 }
832
833 }
834
835 }
836
837 }
838
839 }
840
841 }
842
843 }
844
845 }
846
847 }
848
849 }
850
851 }
852
853 }
854
855 }
856
857 }
858
859 }
860
861 }
862
863 }
864
865 }
866
867 }
868
869 }
870
871 }
872
873 }
874
875 }
876
877 }
878
879 }
880
881 }
882
883 }
884
885 }
886
887 }
888
889 }
890
891 }
892
893 }
894
895 }
896
897 }
898
899 }
900
901 }
902
903 }
904
905 }
906
907 }
908
909 }
910
911 }
912
913 }
914
915 }
916
917 }
918
919 }
920
921 }
922
923 }
924
925 }
926
927 }
928
929 }
930
931 }
932
933 }
934
935 }
936
937 }
938
939 }
940
941 }
942
943 }
944
945 }
946
947 }
948
949 }
950
951 }
952
953 }
954
955 }
956
957 }
958
959 }
960
961 }
962
963 }
964
965 }
966
967 }
968
969 }
970
971 }
972
973 }
974
975 }
976
977 }
978
979 }
980
981 }
982
983 }
984
985 }
986
987 }
988
989 }
990
991 }
992
993 }
994
995 }
996
997 }
998
999 }
1000
1001 }
1002
1003 }
1004
1005 }
1006
1007 }
1008
1009 }
1010
1011 }
1012
1013 }
1014
1015 }
1016
1017 }
1018
1019 }
1020
1021 }
1022
1023 }
1024
1025 }
1026
1027 }
1028
1029 }
1030
1031 }
1032
1033 }
1034
1035 }
1036
1037 }
1038
1039 }
1040
1041 }
1042
1043 }
1044
1045 }
1046
1047 }
1048
1049 }
1050
1051 }
1052
1053 }
1054
1055 }
1056
1057 }
1058
1059 }
1060
1061 }
1062
1063 }
1064
1065 }
1066
1067 }
1068
1069 }
1070
1071 }
1072
1073 }
1074
1075 }
1076
1077 }
1078
1079 }
1080
1081 }
1082
1083 }
1084
1085 }
1086
1087 }
1088
1089 }
1090
1091 }
1092
1093 }
1094
1095 }
1096
1097 }
1098
1099 }
1100
1101 }
1102
1103 }
1104
1105 }
1106
1107 }
1108
1109 }
1110
1111 }
1112
1113 }
1114
1115 }
1116
1117 }
1118
1119 }
1120
1121 }
1122
1123 }
1124
1125 }
1126
1127 }
1128
1129 }
1130
1131 }
1132
1133 }
1134
1135 }
1136
1137 }
1138
1139 }
1140
1141 }
1142
1143 }
1144
1145 }
1146
1147 }
1148
1149 }
1150
1151 }
1152
1153 }
1154
1155 }
1156
1157 }
1158
1159 }
1160
1161 }
1162
1163 }
1164
1165 }
1166
1167 }
1168
1169 }
1170
1171 }
1172
1173 }
1174
1175 }
1176
1177 }
1178
1179 }
1180
1181 }
1182
1183 }
1184
1185 }
1186
1187 }
1188
1189 }
1190
1191 }
1192
1193 }
1194
1195 }
1196
1197 }
1198
1199 }
1200
1201 }
1202
1203 }
1204
1205 }
1206
1207 }
1208
1209 }
1210
1211 }
1212
1213 }
1214
1215 }
1216
1217 }
1218
1219 }
1220
1221 }
1222
1223 }
1224
1225 }
1226
1227 }
1228
1229 }
1230
1231 }
1232
1233 }
1234
1235 }
1236
1237 }
1238
1239 }
1240
1241 }
1242
1243 }
1244
1245 }
1246
1247 }
1248
1249 }
1250
1251 }
1252
1253 }
1254
1255 }
1256
1257 }
1258
1259 }
1260
1261 }
1262
1263 }
1264
1265 }
1266
1267 }
1268
1269 }
1270
1271 }
1272
1273 }
1274
1275 }
1276
1277 }
1278
1279 }
1280
1281 }
1282
1283 }
1284
1285 }
1286
1287 }
1288
1289 }
1290
1291 }
1292
1293 }
1294
1295 }
1296
1297 }
1298
1299 }
1300
1301 }
1302
1303 }
1304
1305 }
1306
1307 }
1308
1309 }
1310
1311 }
1312
1313 }
1314
1315 }
1316
1317 }
1318
1319 }
1320
1321 }
1322
1323 }
1324
1325 }
1326
1327 }
1328
1329 }
1330
1331 }
1332
1333 }
1334
1335 }
1336
1337 }
1338
1339 }
1340
1341 }
1342
1343 }
1344
1345 }
1346
1347 }
1348
1349 }
1350
1351 }
1352
1353 }
1354
1355 }
1356
1357 }
1358
1359 }
1360
1361 }
1362
1363 }
1364
1365 }
1366
1367 }
1368
1369 }
1370
1371 }
1372
1373 }
1374
1375 }
1376
1377 }
1378
1379 }
1380
1381 }
1382
1383 }
1384
1385 }
1386
1387 }
1388
1389 }
1390
1391 }
1392
1393 }
1394
1395 }
1396
1397 }
1398
1399 }
1400
1401 }
1402
1403 }
1404
1405 }
1406
1407 }
1408
1409 }
1410
1411 }
1412
1413 }
1414
1415 }
1416
1417 }
1418
1419 }
1420
1421 }
1422
1423 }
1424
1425 }
1426
1427 }
1428
1429 }
1429

```

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

```

1 using System.Collections.Generic;
2 using System.Runtime.CompilerServices;
3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.RegularExpressions.Transformer
7 {
8     public interface ITransformer
9     {
10         IList<ISubstitutionRule> Rules
11         {
12             [MethodImpl(MethodImplOptions.AggressiveInlining)]
13             get;
14         }
15     }
16 }

```

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

```

1 using System.IO;
2 using System.Runtime.CompilerServices;
3 using System.Text;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.RegularExpressions.Transformer
8 {
9     public class LoggingFileTransformer : FileTransformer
10     {
11         [MethodImpl(MethodImplOptions.AggressiveInlining)]
12         public LoggingFileTransformer(ITextTransformer textTransformer, string
13             ↪ sourceFileExtension, string targetFileExtension) : base(textTransformer,
14             ↪ sourceFileExtension, targetFileExtension) { }
15
16         [MethodImpl(MethodImplOptions.AggressiveInlining)]
17         protected override void TransformFile(string sourcePath, string targetPath)
18         {
19             base.TransformFile(sourcePath, targetPath);
20             // Logging
21             var sourceText = File.ReadAllText(sourcePath, Encoding.UTF8);
22             _textTransformer.WriteStepsToFiles(sourceText, targetPath, skipFilesWithNoChanges:
23                 ↪ true);
24         }
25     }
26 }
27
28 }
29
30 }
31
32 }
33
34 }
35
36 }
37
38 }
39
40 }
41
42 }
43
44 }
45
46 }
47
48 }
49
50 }
51
52 }
53
54 }
55
56 }
57
58 }
59
60 }
61
62 }
63
64 }
65
66 }
67
68 }
69
70 }
71
72 }
73
74 }
75
76 }
77
78 }
79
80 }
81
82 }
83
84 }
85
86 }
87
88 }
89
90 }
91
92 }
93
94 }
95
96 }
97
98 }
99
100 }
101
102 }
103
104 }
105
106 }
107
108 }
109
110 }
111
112 }
113
114 }
115
116 }
117
118 }
119
120 }
121
122 }
123
124 }
125
126 }
127
128 }
129
130 }
131
132 }
133
134 }
135
136 }
137
138 }
139
140 }
141
142 }
143
144 }
145
146 }
147
148 }
149
150 }
151
152 }
153
154 }
155
156 }
157
158 }
159
160 }
161
162 }
163
164 }
165
166 }
167
168 }
169
170 }
171
172 }
173
174 }
175
176 }
177
178 }
179
180 }
181
182 }
183
184 }
185
186 }
187
188 }
189
190 }
191
192 }
193
194 }
195
196 }
197
198 }
199
200 }
201
202 }
203
204 }
205
206 }
207
208 }
209
210 }
211
212 }
213
214 }
215
216 }
217
218 }
219
220 }
221
222 }
223
224 }
225
226 }
227
228 }
229
230 }
231
232 }
233
234 }
235
236 }
237
238 }
239
240 }
241
242 }
243
244 }
245
246 }
247
248 }
249
250 }
251
252 }
253
254 }
255
256 }
257
258 }
259
260 }
261
262 }
263
264 }
265
266 }
267
268 }
269
270 }
271
272 }
273
274 }
275
276 }
277
278 }
279
280 }
281
282 }
283
284 }
285
286 }
287
288 }
289
290 }
291
292 }
293
294 }
295
296 }
297
298 }
299
300 }
301
302 }
303
304 }
305
306 }
307
308 }
309
310 }
311
312 }
313
314 }
315
316 }
317
318 }
319
320 }
321
322 }
323
324 }
325
326 }
327
328 }
329
330 }
331
332 }
333
334 }
335
336 }
337
338 }
339
340 }
341
342 }
343
344 }
345
346 }
347
348 }
349
350 }
351
352 }
353
354 }
355
356 }
357
358 }
359
360 }
361
362 }
363
364 }
365
366 }
367
368 }
369
370 }
371
372 }
373
374 }
375
376 }
377
378 }
379
380 }
381
382 }
383
384 }
385
386 }
387
388 }
389
390 }
391
392 }
393
394 }
395
396 }
397
398 }
399
400 }
401
402 }
403
404 }
405
406 }
407
408 }
409
410 }
411
412 }
413
414 }
415
416 }
417
418 }
419
420 }
421
422 }
423
424 }
425
426 }
427
428 }
429
430 }
431
432 }
433
434 }
435
436 }
437
438 }
439
440 }
441
442 }
443
444 }
445
446 }
447
448 }
449
450 }
451
452 }
453
454 }
455
456 }
457
458 }
459
460 }
461
462 }
463
464 }
465
466 }
467
468 }
469
470 }
471
472 }
473
474 }
475
476 }
477
478 }
479
480 }
481
482 }
483
484 }
485
486 }
487
488 }
489
490 }
491
492 }
493
494 }
495
496 }
497
498 }
499
500 }
501
502 }
503
504 }
505
506 }
507
508 }
509
510 }
511
512 }
513
514 }
515
516 }
517
518 }
519
520 }
521
522 }
523
524 }
525
526 }
527
528 }
529
530 }
531
532 }
533
534 }
535
536 }
537
538 }
539
540 }
541
542 }
543
544 }
545
546 }
547
548 }
549
550 }
551
552 }
553
554 }
555
556 }
557
558 }
559
560 }
561
562 }
563
564 }
565
566 }
567
568 }
569
570 }
571
572 }
573
574 }
575
576 }
577
578 }
579
580 }
581
582 }
583
584 }
585
586 }
587
588 }
589
590 }
591
592 }
593
594 }
595
596 }
597
598 }
599
600 }
601
602 }
603
604 }
605
606 }
607
608 }
609
610 }
611
612 }
613
614 }
615
616 }
617
618 }
619
620 }
621
622 }
623
624 }
625
626 }
627
628 }
629
630 }
631
632 }
633
634 }
635
636 }
637
638 }
639
640 }
641
642 }
643
644 }
645
646 }
647
648 }
649
650 }
651
652 }
653
654 }
655
656 }
657
658 }
659
660 }
661
662 }
663
664 }
665
666 }
667
668 }
669
670 }
671
672 }
673
674 }
675
676 }
677
678 }
679
680 }
681
682 }
683
684 }
685
686 }
687
688 }
689
690 }
691
692 }
693
694 }
695
696 }
697
698 }
699
700 }
701
702 }
703
704 }
705
706 }
707
708 }
709
710 }
711
712 }
713
714 }
715
716 }
717
718 }
719
720 }
721
722 }
723
724 }
725
726 }
727
728 }
729
730 }
731
732 }
733
734 }
735
736 }
737
738 }
739
740 }
741
742 }
743
744 }
745
746 }
747
748 }
749
750 }
751
752 }
753
754 }
755
756 }
757
758 }
759
760 }
761
762 }
763
764 }
765
766 }
767
768 }
769
770 }
771
772 }
773
774 }
775
776 }
777
778 }
779
780 }
781
782 }
783
784 }
785
786 }
787
788 }
789
790 }
791
792 }
793
794 }
795
796 }
797
798 }
799
800 }
801
802 }
803
804 }
805
806 }
807
808 }
809
810 }
811
812 }
813
814 }
815
816 }
817
818 }
819
820 }
821
822 }
823
824 }
825
826 }
827
828 }
829
830 }
831
832 }
833
834 }
835
836 }
837
838 }
839
840 }
841
842 }
843
844 }
845
846 }
847
848 }
849
850 }
851
852 }
853
854 }
855
856 }
857
858 }
859
860 }
861
862 }
863
864 }
865
866 }
867
868 }
869
870 }
871
872 }
873
874 }
875
876 }
877
878 }
879
880 }
881
882 }
883
884 }
885
886 }
887
888 }
889
890 }
891
892 }
893
894 }
895
896 }
897
898 }
899
900 }
901
902 }
903
904 }
905
906 }
907
908 }
909
910 }
911
912 }
913
914 }
915
916 }
917
918 }
919
920 }
921
922 }
923
924 }
925
926 }
927
928 }
929
930 }
931
932 }
933
934 }
935
936 }
937
938 }
939
940 }
941
942 }
943
944 }
945
946 }
947
948 }
949
950 }
951
952 }
953
954 }
955
956 }
957
958 }
959
960 }
961
962 }
963
964 }
965
966 }
967
968 }
969
970 }
971
972 }
973
974 }
975
976 }
977
978 }
979
980 }
981
982 }
983
984 }
985
986 }
987
988 }
989
990 }
991
992 }
993
994 }
995
996 }
997
998 }
999
1000 }
1001
1002 }
1003
1004 }
1005
1006 }
1007
1008 }
1009
1010 }
1011
1012 }
1013
1014 }
1015
1016 }
1017
1018 }
1019
1020 }
1021
1022 }
1023
1024 }
1025
1026 }
1027
1028 }
1029
1030 }
1031
1032 }
1033
1034 }
1035
1036 }
1037
1038 }
1039
1040 }
1041
1042 }
1043
1044 }
1045
1046 }
1047
1048 }
1049
1050 }
1051
1052 }
1053
1054 }
1055
1056 }
1057
1058 }
1059
1060 }
1061
1062 }
1063
1064 }
1065
1066 }
1067
1068 }
1069
1070 }
1071
1072 }
1073
1074 }
1075
1076 }
1077
1078 }
1079
1080 }
1081
1082 }
1083
1084 }
1085
1086 }
1087
1088 }
1089
1090 }
1091
1092 }
1093
1094 }
1095
1096 }
1097
1098 }
1099
1100 }
1101
1102 }
1103
1104 }
1105
1106 }
1107
1108 }
1109
1110 }
1111
1112 }
1113
1114 }
1115
1116 }
1117
11
```

```

3 using System.Text.RegularExpressions;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.RegularExpressions.Transformer
8 {
9     public static class RegexExtensions
10    {
11        [MethodImpl(MethodImplOptions.AggressiveInlining)]
12        public static Regex OverrideOptions(this Regex regex, RegexOptions options, TimeSpan
            ↳ matchTimeout)
13        {
14            if (regex == null)
15            {
16                return null;
17            }
18            return new Regex(regex.ToString(), options, matchTimeout);
19        }
20    }
21 }

```

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

```

1 using System.IO;
2 using System.Runtime.CompilerServices;
3 using System.Text;
4
5 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7 namespace Platform.RegularExpressions.Transformer
8 {
9     internal static class StringExtensions
10    {
11        [MethodImpl(MethodImplOptions.AggressiveInlining)]
12        public static void GetPathParts(this string path, out string directoryName, out string
            ↳ targetFilename, out string targetExtension) => (directoryName, targetFilename,
            ↳ targetExtension) = (Path.GetDirectoryNames(path),
            ↳ Path.GetFileNameWithoutExtension(path), Path.GetExtension(path));
13
14        [MethodImpl(MethodImplOptions.AggressiveInlining)]
15        public static void WriteStepToFile(this string text, string directoryName, string
            ↳ targetFilename, string targetExtension, int currentStep) =>
            ↳ File.WriteAllText(Path.Combine(directoryName,
            ↳ $"{targetFilename}.{currentStep}{targetExtension}"), text, Encoding.UTF8);
16    }
17 }

```

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

```

1 using System;
2 using System.Runtime.CompilerServices;
3 using System.Text;
4 using System.Text.RegularExpressions;
5
6 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
7
8 namespace Platform.RegularExpressions.Transformer
9 {
10    public class SubstitutionRule : ISubstitutionRule
11    {
12        public static readonly TimeSpan DefaultMatchTimeout = TimeSpan.FromMinutes(5);
13        public static readonly RegexOptions DefaultMatchPatternRegexOptions =
            ↳ RegexOptions.Compiled | RegexOptions.Multiline;
14
15        public Regex MatchPattern
16        {
17            [MethodImpl(MethodImplOptions.AggressiveInlining)]
18            get;
19            [MethodImpl(MethodImplOptions.AggressiveInlining)]
20            set;
21        }
22
23        public string SubstitutionPattern
24        {
25            [MethodImpl(MethodImplOptions.AggressiveInlining)]
26            get;
27            [MethodImpl(MethodImplOptions.AggressiveInlining)]
28            set;
29        }
30
31        public Regex PathPattern

```

```

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

```



```

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

```

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

```

1  using System;
2  using System.Collections.Generic;
3  using System.Runtime.CompilerServices;
4
5  #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
6
7  namespace Platform.RegularExpressions.Transformer
8  {
9      public class TextSteppedTransformer : ITransformer
10     {
11         public IList<ISubstitutionRule> Rules
12         {
13             [MethodImpl(MethodImplOptions.AggressiveInlining)]
14             get;
15             [MethodImpl(MethodImplOptions.AggressiveInlining)]
16             set;
17         }
18
19         public string Text
20         {
21             [MethodImpl(MethodImplOptions.AggressiveInlining)]
22             get;
23             [MethodImpl(MethodImplOptions.AggressiveInlining)]
24             set;
25         }
26
27         public int Current
28         {
29             [MethodImpl(MethodImplOptions.AggressiveInlining)]
30             get;
31             [MethodImpl(MethodImplOptions.AggressiveInlining)]
32             set;
33         }
34
35         [MethodImpl(MethodImplOptions.AggressiveInlining)]
36         public TextSteppedTransformer(IList<ISubstitutionRule> rules, string text, int current)
37             => Reset(rules, text, current);
38
39         [MethodImpl(MethodImplOptions.AggressiveInlining)]
40         public TextSteppedTransformer(IList<ISubstitutionRule> rules, string text) =>
41             Reset(rules, text);
42
43         [MethodImpl(MethodImplOptions.AggressiveInlining)]
44         public TextSteppedTransformer(IList<ISubstitutionRule> rules) => Reset(rules);
45
46         [MethodImpl(MethodImplOptions.AggressiveInlining)]
47         public TextSteppedTransformer() => Reset();
48
49         [MethodImpl(MethodImplOptions.AggressiveInlining)]
50         public void Reset(IList<ISubstitutionRule> rules, string text, int current)
51         {
52             Rules = rules;
53             Text = text;
54             Current = current;
55         }
56     }
57 }

```

```

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

```

1.13 ./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         private readonly TextSteppedTransformer _baseTransformer;
11
12         public IList<ISubstitutionRule> Rules
13         {
14             [MethodImpl(MethodImplOptions.AggressiveInlining)]
15             get;
16             [MethodImpl(MethodImplOptions.AggressiveInlining)]
17             private set;
18         }
19
20         [MethodImpl(MethodImplOptions.AggressiveInlining)]
21         public TextTransformer(IList<ISubstitutionRule> substitutionRules)
22         {
23             Rules = substitutionRules;
24             _baseTransformer = new TextSteppedTransformer(substitutionRules);
25         }
26
27         [MethodImpl(MethodImplOptions.AggressiveInlining)]
28         public string Transform(string source)
29         {
30             _baseTransformer.Reset(source);
31             while (_baseTransformer.Next());
32             return _baseTransformer.Text;
33         }
34     }
35 }

```

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

```

1  using System.Runtime.CompilerServices;
2  using Platform.Collections.Arrays;

```

```

3
4 #pragma warning disable CS1591 // Missing XML comment for publicly visible type or member
5
6 namespace Platform.RegularExpressions.Transformer
7 {
8     public class TransformerCLI
9     {
10         private readonly IFileTransformer _transformer;
11
12         [MethodImpl(MethodImplOptions.AggressiveInlining)]
13         public TransformerCLI(IFileTransformer transformer) => _transformer = transformer;
14
15         [MethodImpl(MethodImplOptions.AggressiveInlining)]
16         public void Run(string[] args)
17         {
18             var sourcePath = args.GetElementOrDefault(0);
19             var targetPath = args.GetElementOrDefault(1);
20             _transformer.Transform(sourcePath, targetPath);
21         }
22     }
23 }

```

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

```

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

```

```

53         Assert.False(File.Exists(Path.Combine(targetFolderPath, "x.cpp")));
54
55         Directory.Delete(sourceFolderPath, true);
56         Directory.Delete(targetFolderPath, true);
57     }
58 }
59 }

```

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

```

1  using System.Text.RegularExpressions;
2  using Xunit;
3
4  namespace Platform.RegularExpressions.Transformer.Tests
5  {
6      public class MarkovAlgorithmsTests
7      {
8          /// <remarks>
9          /// Example is from https://en.wikipedia.org/wiki/Markov_algorithm.
10         /// </remarks>
11         [Fact]
12         public void BinaryToUnaryNumbersTest()
13         {
14             var rules = new SubstitutionRule[]
15             {
16                 ("1", "0|", int.MaxValue), // "1" -> "0|" repeated forever
17                 // | symbol should be escaped for regular expression pattern, but not in the
18                 // ↪ substitution pattern
19                 ("0", "0||", int.MaxValue), // "\|0" -> "0||" repeated forever
20                 ("0", "", int.MaxValue), // "0" -> "" repeated forever
21             };
22             var transformer = new TextTransformer(rules);
23             var input = "101";
24             var expectedOutput = "||||";
25             var output = transformer.Transform(input);
26             Assert.Equal(expectedOutput, output);
27         }
28     }
29 }

```

1.17 ./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 }

```

1.18 ./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     }
23 }

```

```

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 [Fact]
98 public void DebugOutputUsingTransformersGenerationTest()
99 {
100     var sourceText = "aaaa";

```

```

95     var firstStepReferenceText = "bbbb";
96     var secondStepReferenceText = "cccc";
97
98     var transformer = new TextTransformer(new SubstitutionRule[] {
99         (new Regex("a"), "b"),
100         (new Regex("b"), "c")
101     });
102
103     var steps =
104         ↪ transformer.GenerateTransformersForEachRule().TransformWithAll(sourceText);
105
106     Assert.Equal(2, steps.Count);
107     Assert.Equal(firstStepReferenceText, steps[0]);
108     Assert.Equal(secondStepReferenceText, steps[1]);
109 }
110
111 [Fact]
112 public void DebugFilesOutputUsingTransformersGenerationTest()
113 {
114     var sourceText = "aaaa";
115     var firstStepReferenceText = "bbbb";
116     var secondStepReferenceText = "cccc";
117
118     var transformer = new TextTransformer(new SubstitutionRule[] {
119         (new Regex("a"), "b"),
120         (new Regex("b"), "c")
121     });
122
123     var targetFilename = Path.GetTempFileName();
124
125     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
126         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: false);
127
128     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
129     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
130
131     Assert.True(File.Exists(firstStepReferenceFilename));
132     Assert.True(File.Exists(secondStepReferenceFilename));
133
134     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
135         ↪ Encoding.UTF8));
136     Assert.Equal(secondStepReferenceText, File.ReadAllText(secondStepReferenceFilename,
137         ↪ Encoding.UTF8));
138
139     File.Delete(firstStepReferenceFilename);
140     File.Delete(secondStepReferenceFilename);
141 }
142
143 [Fact]
144 public void FilesWithNoChangesSkippedWhenUsingTransformersGenerationTest()
145 {
146     var sourceText = "aaaa";
147     var firstStepReferenceText = "bbbb";
148     var thirdStepReferenceText = "cccc";
149
150     var transformer = new TextTransformer(new SubstitutionRule[] {
151         (new Regex("a"), "b"),
152         (new Regex("x"), "y"),
153         (new Regex("b"), "c")
154     });
155
156     var targetFilename = Path.GetTempFileName();
157
158     transformer.GenerateTransformersForEachRule().TransformWithAllToFiles(sourceText,
159         ↪ $"{targetFilename}.txt", skipFilesWithNoChanges: true);
160
161     var firstStepReferenceFilename = $"{targetFilename}.0.txt";
162     var secondStepReferenceFilename = $"{targetFilename}.1.txt";
163     var thirdStepReferenceFilename = $"{targetFilename}.2.txt";
164
165     Assert.True(File.Exists(firstStepReferenceFilename));
166     Assert.False(File.Exists(secondStepReferenceFilename));
167     Assert.True(File.Exists(thirdStepReferenceFilename));
168
169     Assert.Equal(firstStepReferenceText, File.ReadAllText(firstStepReferenceFilename,
170         ↪ Encoding.UTF8));
171     Assert.Equal(thirdStepReferenceText, File.ReadAllText(thirdStepReferenceFilename,
172         ↪ Encoding.UTF8));

```

```
167         File.Delete(firstStepReferenceFilename);
168         File.Delete(secondStepReferenceFilename);
169         File.Delete(thirdStepReferenceFilename);
170     }
171 }
172 }
```

Index

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