LinksPlatform's Platform.RegexTransformer Class Library

./Context.cs

```
#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public class Context : IContext
    {
        public string Path { get; }

        public Context(string path) => Path = path;
    }
}
```

./IContext.cs

```
#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public interface IContext
    {
        public string Path { get; }
    }
}
```

./ISubstitutionRule.cs

```
using System.Text.RegularExpressions;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public interface ISubstitutionRule
    {
        Regex MatchPattern { get; }
        string SubstitutionPattern { get; }
        Regex PathPattern { get; }
        int MaximumRepeatCount { get; }
    }
}
```

./ITransformer.cs

```
#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public interface ITransformer
    {
        string Transform(string source, IContext context);
    }
}
```

./RegexExtensions.cs

```
using System;
using System.Text.RegularExpressions;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public static class RegexExtensions
    {
        public static Regex OverrideOptions(this Regex regex, RegexOptions options, TimeSpan
        ↪  matchTimeout) => new Regex(regex.ToString(), options, matchTimeout);
    }
}
```

./SubstitutionRule.cs

```
using System;
using System.Text.RegularExpressions;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public class SubstitutionRule : ISubstitutionRule
    {
        public static readonly TimeSpan DefaultMatchTimeout = TimeSpan.FromMinutes(5);
```

```
        public static readonly RegexOptions DefaultMatchPatternRegexOptions =
        ↪  RegexOptions.Compiled | RegexOptions.Multiline;
        public static readonly RegexOptions DefaultPathPatternRegexOptions =
        ↪  RegexOptions.Compiled | RegexOptions.Singleline;

        public Regex MatchPattern { get; set; }

        public string SubstitutionPattern { get; set; }

        public Regex PathPattern { get; set; }

        public int MaximumRepeatCount { get; set; }

        public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
        ↪  pathPattern, int maximumRepeatCount, RegexOptions? matchPatternOptions,
        ↪  RegexOptions? pathPatternOptions, TimeSpan? matchTimeout)
        {
            MatchPattern = matchPattern;
            SubstitutionPattern = substitutionPattern;
            PathPattern = pathPattern;
            MaximumRepeatCount = maximumRepeatCount;
            OverrideMatchPatternOptions(matchPatternOptions ?? matchPattern.Options,
            ↪  matchTimeout ?? matchPattern.MatchTimeout);
            OverrideMatchPatternOptions(pathPatternOptions ?? pathPattern.Options, matchTimeout
            ↪  ?? pathPattern.MatchTimeout);
        }

        public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
        ↪  pathPattern, int maximumRepeatCount, bool useDefaultOptions) : this(matchPattern,
        ↪  substitutionPattern, pathPattern, maximumRepeatCount, useDefaultOptions ?
        ↪  DefaultMatchPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
        ↪  DefaultPathPatternRegexOptions : (RegexOptions?)null, useDefaultOptions ?
        ↪  DefaultMatchTimeout : (TimeSpan?)null) { }

        public SubstitutionRule(Regex matchPattern, string substitutionPattern, Regex
        ↪  pathPattern, int maximumRepeatCount) : this(matchPattern, substitutionPattern,
        ↪  pathPattern, maximumRepeatCount, true) { }

        public SubstitutionRule(Regex matchPattern, string substitutionPattern, int
        ↪  maximumRepeatCount) : this(matchPattern, substitutionPattern, null,
        ↪  maximumRepeatCount) { }

        public SubstitutionRule(Regex matchPattern, string substitutionPattern) :
        ↪  this(matchPattern, substitutionPattern, null, 0) { }

        public static implicit operator SubstitutionRule(ValueTuple<Regex, string> tuple) => new
        ↪  SubstitutionRule(tuple.Item1, tuple.Item2);

        public static implicit operator SubstitutionRule(ValueTuple<Regex, string, int> tuple)
        ↪  => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3);

        public static implicit operator SubstitutionRule(ValueTuple<Regex, string, Regex, int>
        ↪  tuple) => new SubstitutionRule(tuple.Item1, tuple.Item2, tuple.Item3, tuple.Item4);

        public void OverrideMatchPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
        ↪  MatchPattern = MatchPattern.OverrideOptions(options, matchTimeout);

        public void OverridePathPatternOptions(RegexOptions options, TimeSpan matchTimeout) =>
        ↪  PathPattern = PathPattern.OverrideOptions(options, matchTimeout);
    }
}
```

./TransformerCLI.cs

```
using System.Diagnostics;
using System.IO;
using System.Text;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public class TransformerCLI
    {
        private readonly ITransformer _transformer;

        public TransformerCLI(ITransformer transformer) => _transformer = transformer;

        public bool Run(string[] args, out string message)
        {
```

```
            message = "";
            var sourcePath = GetArgOrDefault(args, 0);
            if (!File.Exists(sourcePath))
            {
                message = $"{sourcePath} file does not exist.";
                return false;
            }
            var targetPath = GetArgOrDefault(args, 1);
            if (string.IsNullOrWhiteSpace(targetPath))
            {
                targetPath = Path.ChangeExtension(sourcePath, ".cpp");
            }
            else if ((Directory.Exists(targetPath) &&
            ↪  File.GetAttributes(targetPath).HasFlag(FileAttributes.Directory)) ||
            ↪  LooksLikeDirectoryPath(targetPath))
            {
                targetPath = Path.Combine(targetPath,
                ↪  Path.ChangeExtension(Path.GetFileName(sourcePath), ".cpp"));
            }
            if (File.Exists(targetPath))
            {
                var applicationPath = Process.GetCurrentProcess().MainModule.FileName;
                var targetFileLastUpdateDateTime = new FileInfo(targetPath).LastWriteTimeUtc;
                if (new FileInfo(sourcePath).LastWriteTimeUtc < targetFileLastUpdateDateTime &&
                ↪  new FileInfo(applicationPath).LastWriteTimeUtc <
                ↪  targetFileLastUpdateDateTime)
                {
                    return true;
                }
            }
            File.WriteAllText(targetPath, _transformer.Transform(File.ReadAllText(sourcePath,
            ↪  Encoding.UTF8), new Context(sourcePath)), Encoding.UTF8);
            message = $"{targetPath} file written.";
            return true;
        }

        private static bool LooksLikeDirectoryPath(string targetPath) =>
        ↪  targetPath.EndsWith(Path.DirectorySeparatorChar) ||
        ↪  targetPath.EndsWith(Path.AltDirectorySeparatorChar);

        private static string GetArgOrDefault(string[] args, int index) => args.Length > index ?
        ↪  args[index] : null;
    }
}
```

./Transformer.cs

```
using System.Collections.Generic;

#pragma warning disable CS1591 // Missing XML comment for publicly visible type or member

namespace Platform.RegexTransformer
{
    public class Transformer : ITransformer
    {
        private readonly IList<ISubstitutionRule> _substitutionRules;

        public Transformer(IList<ISubstitutionRule> substitutionRules) => _substitutionRules =
        ↪  substitutionRules;

        public string Transform(string source, IContext context)
        {
            var current = source;
            for (var i = 0; i < _substitutionRules.Count; i++)
            {
                var rule = _substitutionRules[i];
                var matchPattern = rule.MatchPattern;
                var substitutionPattern = rule.SubstitutionPattern;
                var pathPattern = rule.PathPattern;
                var maximumRepeatCount = rule.MaximumRepeatCount;
                if (pathPattern == null || pathPattern.IsMatch(context.Path))
                {
                    var replaceCount = 0;
                    do
                    {
                        current = matchPattern.Replace(current, substitutionPattern);
                        if (++replaceCount > maximumRepeatCount)
                        {
                            break;
                        }
```

```
                    }
                    while (matchPattern.IsMatch(current));
                }
            }
            return current;
        }
    }
}
```

Index