Contents

Appendix A

Source code

A.1 The string.Format analyzer

```
1  using System.Collections.Immutable;
2  using System.Linq;
3  using System.Text.RegularExpressions;
4  using Microsoft.CodeAnalysis;
5  using Microsoft.CodeAnalysis.CSharp;
6  using
7      Microsoft.CodeAnalysis.CSharp.Syntax
8      ;
9  using Microsoft.CodeAnalysis.Diagnostics
10     ;
11 using VSDiagnostics.Utilities;
12
13 namespace
14     VSDiagnostics.Diagnostics.Strings.
15     StringDotFormatWithDifferentAmountOfArguments
16 {
17     [DiagnosticAnalyzer(
18         LanguageNames.CSharp)]
19     public class
20         StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
               :
21             DiagnosticAnalyzer
22     {
23         private const DiagnosticSeverity
24             Severity
25                 =
26                 DiagnosticSeverity.Error;
```

```
27
28          private static readonly string
29              Category =
30                  VSDiagnosticsResources
31                      .StringsCategory;
32
33          private static readonly string
34              Message =
35                  VSDiagnosticsResources
36                      .
                            StringDotFormatWithDifferentAmountOfArgumentsMessag
                            ;
37
38          private static readonly string
39              Title =
40                  VSDiagnosticsResources
41                      .
                            StringDotFormatWithDifferentAmountOfArgumentsTitle
                            ;
42
43          internal static
44              DiagnosticDescriptor Rule
45              =>
46                  new DiagnosticDescriptor
47                      (
48                      DiagnosticId
49                          .
                                StringDotFormatWithDifferentAmountOfArguments
                                ,
50                      Title, Message,
51                      Category,
52                      Severity, true);
53
54          public override
55              ImmutableArray
56                  <DiagnosticDescriptor>
57              SupportedDiagnostics
58              =>
59                  ImmutableArray.Create(
60                      Rule);
61
62          public override void Initialize(
63              AnalysisContext context)
```

```
64          {
65              context
66                  .RegisterSyntaxNodeAction
67                  (
68                      AnalyzeNode,
69                      SyntaxKind
70                          .InvocationExpression);
71          }
72
73          private void AnalyzeNode(
74              SyntaxNodeAnalysisContext
75                  context)
76          {
77              var invocation =
78                  context.Node as
79                      InvocationExpressionSyntax;
80              if (
81                  invocation?.ArgumentList ==
82                  null)
83              {
84                  return;
85              }
86
87              // Get the format string
88              // This corresponds to the argument passed to
                    the parameter with name 'format'
89              var invokedMethod =
90                  context.SemanticModel
91                         .GetSymbolInfo(
92                             invocation);
93              var methodSymbol =
94                  invokedMethod.Symbol as
95                      IMethodSymbol;
96
97              // Verify we're dealing with a call to a
                    method that accepts a variable named '
                    format' and a object, params object[] or a
                    plain object[]
98              // params object[] and object[] can both be
                    verified by looking for the latter
99              // This allows us to support similar calls
                    like Console.WriteLine("{0}", "test") as
                    well which carry an implicit string.Format
```

```
100             var formatParam =
101                 methodSymbol?.Parameters
102                              .FirstOrDefault
103                     (
104                         x =>
105                             x.Name ==
106                             "format");
107             if (formatParam == null)
108             {
109                 return;
110             }
111             var formatIndex =
112                 formatParam.Ordinal;
113
114             var formatParameters =
115                 methodSymbol.Parameters
116                             .Skip(
117                                 formatIndex +
118                                 1)
119                             .ToArray();
120             var hasObjectArray =
121                 formatParameters.Length ==
122                 1 &&
123                 formatParameters.All(
124                     x =>
125                         x.Type.Kind ==
126                         SymbolKind
127                             .ArrayType &&
128                         ((
129                             IArrayTypeSymbol
130                             ) x.Type)
131                             .ElementType
132                             .SpecialType ==
133                         SpecialType
134                             .System_Object);
135             var hasObject =
136                 formatParameters.All(
137                     x =>
138                         x.Type
139                          .SpecialType ==
140                         SpecialType
141                             .System_Object);
142
```

```
143             if (
144                 !(hasObject ||
145                   hasObjectArray))
146             {
147                 return;
148             }
149
150             var formatExpression =
151                 invocation.ArgumentList
152                           .Arguments[
153                               formatIndex
154                     ].Expression;
155             var formatString =
156                 context.SemanticModel
157                        .GetConstantValue
158                     (
159                         formatExpression);
160             if (!formatString.HasValue)
161             {
162                 return;
163             }
164
165             // Get the total amount of arguments passed in
                    for the format
166             // If the first one is the literal (aka: the
                format specified) then every other argument
                    is an argument to the format
167             // If not, it means the first one is the
                CultureInfo, the second is the format and
                all others are format arguments
168             // We also have to check whether or not the
                arguments are passed in through an explicit
                    array or whether they use the params
                syntax
169             var formatArguments =
170                 invocation.ArgumentList
171                           .Arguments
172                           .Skip(
173                               formatIndex +
174                               1)
175                           .ToArray();
176             var amountOfFormatArguments
177                 =
```

```
178                 formatArguments.Length;
179
180             if (
181                 formatArguments.Length ==
182                 1)
183             {
184                 var argumentType =
185                     context
186                         .SemanticModel
187                         .GetTypeInfo(
188                             formatArguments
189                                 [0]
190                                 .Expression);
191                 if (argumentType.Type ==
192                     null)
193                 {
194                     return;
195                 }
196
197                 // Inline array creation a la string.
                       Format("{0}", new object[] { "test" })
198                 if (
199                     argumentType.Type
200                                 .TypeKind ==
201                     TypeKind.Array)
202                 {
203                     // We check for an invocation first to
                           account for the scenario where you
                           have both an invocation and an
                           array initializer
204                     // Think about something like this:
                           string.Format(""{0}{1}{2}"", new[]
                           { 1 }.Concat(new[] {2}).ToArray());
205                     var methodInvocation
206                         =
207                         formatArguments[
208                             0]
209                         .DescendantNodes
210                         ()
211                         .OfType
212                         <
213                             InvocationExpressionSyntax
214                             >()
```

```
215                         .FirstOrDefault
216                         ();
217                     if (
218                         methodInvocation !=
219                         null)
220                     {
221                         // We don't handle method calls
                               that return an array in the
                               case of a single argument
222                         return;
223                     }
224
225                     var
226                         inlineArrayCreation
227                             =
228                             formatArguments
229                                 [0]
230                                 .DescendantNodes
231                                 ()
232                                 .OfType
233                                 <
234                                     InitializerExpressionSyntax
                                    
235                                     >()
236                                 .FirstOrDefault
237                                 ();
238                     if (
239                         inlineArrayCreation !=
240                         null)
241                     {
242                         amountOfFormatArguments
243                             =
244                             inlineArrayCreation
245                                 .Expressions
246                                 .Count;
247                         goto
248                             placeholderVerification;
249                     }
250
251                     // If we got here it means the
                           arguments are passed in through an
                           identifier which resolves to an
                           array
```

```
252                     // aka: referencing a variable/field
                           that is of type T[]
253                     // We cannot reliably get the amount
                           of arguments if it's a method
254                     // We could get them when it's a field
                           /variable/property but that takes
                           some more work and thinking about
                           it
255                     // This is tracked in workitem https
                           ://github.com/Vannevelj/
                           VSDiagnostics/issues/330
256                     if (hasObjectArray)
257                     {
258                         return;
259                     }
260                 }
261             }
262
263             placeholderVerification:
264             // Get the placeholders we use stripped off
                   their format specifier, get the highest
                   value
265             // and verify that this value + 1 (to account
                   for 0-based indexing) is not greater than
                   the amount of placeholder arguments
266             var placeholders =
267                 PlaceholderHelpers
268                     .GetPlaceholders
269                     ((string)
270                         formatString
271                             .Value)
272                     .Cast<Match>()
273                     .Select(
274                         x => x.Value)
275                     .Select(
276                         PlaceholderHelpers
277                             .GetPlaceholderIndex)
278                     .Select(
279                         int.Parse)
280                     .ToList();
281
282             if (!placeholders.Any())
283             {
```

```
284                 return;
285             }
286
287             var highestPlaceholder =
288                 placeholders.Max();
289             if (highestPlaceholder + 1 >
290                 amountOfFormatArguments)
291             {
292                 context.ReportDiagnostic
293                     (
294                         Diagnostic
295                             .Create(
296                                 Rule,
297                                 formatExpression
298                                     .GetLocation
299                                     ()));
300             }
301         }
302     }
303 }
```

A.2 The analyzer its unit tests

```
1  using Microsoft.CodeAnalysis.Diagnostics
2      ;
3  using
4      Microsoft.VisualStudio.TestTools.
5          UnitTesting;
6  using RoslynTester.Helpers.CSharp;
7  using
8      VSDiagnostics.Diagnostics.Strings.
9          StringDotFormatWithDifferentAmountOfArguments
10     ;
11
12 namespace VSDiagnostics.Test.Tests.
13     Strings
14 {
15     [TestClass]
16     public class
17         StringDotFormatWithDifferentAmountOfArgumentsTests
               :
18             CSharpDiagnosticVerifier
19     {
```

```
20          protected override
21              DiagnosticAnalyzer
22              DiagnosticAnalyzer
23              =>
24                  new
                       StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

25                      ();
26
27          [TestMethod]
28          public void
29              StringDotFormatWithDifferentAmountOfArguments_WithValidScenari

30              ()
31          {
32              var original = @"
33  using System;
34  using System.Text;
35
36  namespace ConsoleApplication1
37  {
38      class MyClass
39      {
40          void Method(string input)
41          {
42              string s = string.Format(""abc {0}, def {1}"",
                    1, 2);
43          }
44      }
45  }";
46              VerifyDiagnostic(original);
47          }
48
49          [TestMethod]
50          public void
51              StringDotFormatWithDifferentAmountOfArguments_WithRepeatedPlac

52              ()
53          {
54              var original = @"
55  using System;
56  using System.Text;
57
```

```
58 namespace ConsoleApplication1
59 {
60     class MyClass
61     {
62         void Method(string input)
63         {
64             string s = string.Format(""abc {0}, def {0}"",
                   1);
65         }
66     }
67 }";
68             VerifyDiagnostic(original);
69         }
70
71         [TestMethod]
72         public void
73             StringDotFormatWithDifferentAmountOfArguments_WithExtraAr

74             ()
75         {
76             var original = @"
77 using System;
78 using System.Text;
79
80 namespace ConsoleApplication1
81 {
82     class MyClass
83     {
84         void Method(string input)
85         {
86             string s = string.Format(""abc {0}, def {1}"",
                   1, 2, 3, 4, 5);
87         }
88     }
89 }";
90             VerifyDiagnostic(original);
91         }
92
93         [TestMethod]
94         public void
95             StringDotFormatWithDifferentAmountOfArguments_WithLacking

96             ()
```

```
97          {
98              var original = @"
99  using System;
100 using System.Text;
101
102 namespace ConsoleApplication1
103 {
104     class MyClass
105     {
106         void Method(string input)
107         {
108             string s = string.Format(""abc {0}, def {1}"",
                    1);
109         }
110     }
111 }";
112             VerifyDiagnostic(original,
113                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

114                     .Rule.MessageFormat
115                     .ToString());
116         }
117
118         [TestMethod]
119         public void
120             StringDotFormatWithDifferentAmountOfArguments_WithLackingArgum

121             ()
122         {
123             var original = @"
124 using System;
125 using System.Text;
126
127 namespace ConsoleApplication1
128 {
129     class MyClass
130     {
131         void Method(string input)
132         {
133             string s = string.Format(""abc {1}, def {2}"",
                    123, 456);
134         }
135     }
```

```
136 }";
137             VerifyDiagnostic(original,
138                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

139                     .Rule.MessageFormat
140                     .ToString());
141         }
142
143         [TestMethod]
144         public void
145             StringDotFormatWithDifferentAmountOfArguments_WithEqualAr

146             ()
147         {
148             var original = @"
149 using System;
150 using System.Text;
151
152 namespace ConsoleApplication1
153 {
154     class MyClass
155     {
156         void Method(string input)
157         {
158             string s = string.Format(""abc {0}, def {0}"",
                    1, 2);
159         }
160     }
161 }";
162             VerifyDiagnostic(original);
163         }
164
165         [TestMethod]
166         public void
167             StringDotFormatWithDifferentAmountOfArguments_WithEscaped

168             ()
169         {
170             var original = @"
171 using System;
172 using System.Text;
173
174 namespace ConsoleApplication1
```

```
175 {
176     class MyClass
177     {
178         void Method(string input)
179         {
180             string s = string.Format(""abc {0}, def {{1}}"
                    ", 1);
181         }
182     }
183 }";
184             VerifyDiagnostic(original);
185         }
186 
187         [TestMethod]
188         public void
189             StringDotFormatWithDifferentAmountOfArguments_WithPlaceholderF

190             ()
191         {
192             var original = @"
193 using System;
194 using System.Text;
195 
196 namespace ConsoleApplication1
197 {
198     class MyClass
199     {
200         void Method(string input)
201         {
202             string s = string.Format(""abc {1:00}, def {1}
                    "", 1);
203         }
204     }
205 }";
206             VerifyDiagnostic(original,
207                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

208                     .Rule.MessageFormat
209                     .ToString());
210         }
211 
212         [TestMethod]
213         public void
```

```
214             StringDotFormatWithDifferentAmountOfArguments_InDifferent

215             ()
216         {
217             var original = @"
218 using System;
219 using System.Text;
220
221 namespace ConsoleApplication1
222 {
223     class MyClass
224     {
225         void Method(string input)
226         {
227             string s = string.Format(""abc {1}, def {0}"",
                    1);
228         }
229     }
230 }";
231             VerifyDiagnostic(original,
232                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

233                     .Rule.MessageFormat
234                     .ToString());
235         }
236
237         [TestMethod]
238         public void
239             StringDotFormatWithDifferentAmountOfArguments_WithoutForm

240             ()
241         {
242             var original = @"
243 using System;
244 using System.Text;
245
246 namespace ConsoleApplication1
247 {
248     class MyClass
249     {
250         void Method(string input)
251         {
252             string format = ""abc {0}, def {1}"";
```

```
253             string s = string.Format(format, 1, 2);
254         }
255     }
256 }";
257             VerifyDiagnostic(original);
258         }
259 
260         [TestMethod]
261         public void
262             StringDotFormatWithDifferentAmountOfArguments_WithInterpolated
                
263             ()
264         {
265             var original = @"
266 using System;
267 using System.Text;
268 
269 namespace ConsoleApplication1
270 {
271     class MyClass
272     {
273         void Method(string input)
274         {
275             string name = ""Jeroen"";
276             string s = string.Format($""abc {name}, def
                    {0} ghi {1}"", 1);
277         }
278     }
279 }";
280             VerifyDiagnostic(original);
281         }
282 
283         [TestMethod]
284         public void
285             StringDotFormatWithDifferentAmountOfArguments_WithInterpolated
                
286             ()
287         {
288             var original = @"
289 using System;
290 using System.Globalization;
291 using System.Text;
292 
```

```
293 namespace ConsoleApplication1
294 {
295     class MyClass
296     {
297         void Method(string input)
298         {
299             string name = ""Jeroen"";
300             string s = string.Format(CultureInfo.
                    InvariantCulture, $""abc {name}, def {0}
                    ghi {1}"", 1);
301         }
302     }
303 }";
304             VerifyDiagnostic(original);
305         }
306
307         [TestMethod]
308         public void
309             StringDotFormatWithDifferentAmountOfArguments_WithFormatP

310             ()
311         {
312             var original = @"
313 using System;
314 using System.Globalization;
315 using System.Text;
316
317 namespace ConsoleApplication1
318 {
319     class MyClass
320     {
321         void Method(string input)
322         {
323             string s = string.Format(CultureInfo.
                    InvariantCulture, ""def {0} ghi {1}"", 1);
324         }
325     }
326 }";
327             VerifyDiagnostic(original,
328                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

329                     .Rule.MessageFormat
330                     .ToString());
```

```
331         }
332
333         [TestMethod]
334         public void
335             StringDotFormatWithDifferentAmountOfArguments_WithFormatProvid
                ()
336         {
337             var original = @"
338 using System;
339 using System.Globalization;
340 using System.Text;
341
342 namespace ConsoleApplication1
343 {
344     class MyClass
345     {
346         void Method(string input)
347         {
348             string s = string.Format(CultureInfo.
                    InvariantCulture, ""abc {0}"");
349         }
350     }
351 }";
352             VerifyDiagnostic(original,
353                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                        .Rule.MessageFormat
354                     .ToString());
355         }
356
357         [TestMethod]
358         public void
359             StringDotFormatWithDifferentAmountOfArguments_WithEscapedBrace
                ()
360         {
361             var original = @"
362 using System;
363 using System.Text;
364
365 namespace ConsoleApplication1
366 {
```

```
370         class MyClass
371         {
372             void Method(string input)
373             {
374                 string s = string.Format(""def {0} ghi {{1}}""
                        , 1);
375             }
376         }
377 }";
378                 VerifyDiagnostic(original);
379             }
380 
381             [TestMethod]
382             public void
383                 StringDotFormatWithDifferentAmountOfArguments_WithNestedB
                    
384                 ()
385             {
386                 var original = @"
387 using System;
388 using System.Text;
389 
390 namespace ConsoleApplication1
391 {
392     class MyClass
393     {
394         void Method(string input)
395         {
396             string s = string.Format(""{{def {0} ghi {1}}}
                        "", 1);
397         }
398     }
399 }";
400                 VerifyDiagnostic(original,
401                     StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                        
402                         .Rule.MessageFormat
403                         .ToString());
404             }
405 
406 
407             [TestMethod]
408             public void
```

```
409             StringDotFormatWithDifferentAmountOfArguments_WithSimilarInvoc
410             ()
411         {
412             var original = @"
413 using System;
414 using System.Text;
415
416 namespace ConsoleApplication1
417 {
418     class MyClass
419     {
420         MyClass()
421         {
422             Method(""{{def {0} ghi {1}}}"", 1);
423         }
424
425         void Method(string format, int x)
426         {
427         }
428     }
429 }";
430             VerifyDiagnostic(original);
431         }
432
433         [TestMethod]
434         public void
435             StringDotFormatWithDifferentAmountOfArguments_WithSimilarInvoc
436             ()
437         {
438             var original = @"
439 using System;
440 using System.Text;
441
442 namespace ConsoleApplication1
443 {
444     class MyClass
445     {
446         MyClass()
447         {
448             Method(""{{def {0} ghi {1}}}"", 1);
449         }
```

```
450
451             void Method(string format, object x)
452             {
453             }
454         }
455 }";
456                 VerifyDiagnostic(original,
457                     StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
458                         .Rule.MessageFormat
459                         .ToString());
460             }
461
462             [TestMethod]
463             public void
464                 StringDotFormatWithDifferentAmountOfArguments_WithSimilar
465                 ()
466             {
467                 var original = @"
468 using System;
469 using System.Text;
470
471 namespace ConsoleApplication1
472 {
473     class MyClass
474     {
475         MyClass()
476         {
477             Method(""{{def {0} ghi {1}}}"", 1);
478         }
479
480         void Method(string s, object x)
481         {
482         }
483     }
484 }";
485                 VerifyDiagnostic(original);
486             }
487
488             [TestMethod]
489             public void
490                 StringDotFormatWithDifferentAmountOfArguments_WithoutArgu
```

```
491             ()
492         {
493             var original = @"
494 using System;
495 using System.Text;
496 
497 namespace ConsoleApplication1
498 {
499     class MyClass
500     {
501         void Method(string input)
502         {
503             string s = string.Format(""abc {0}, def {1}"")
                    ;
504         }
505     }
506 }";
507             VerifyDiagnostic(original,
508                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                        .Rule.MessageFormat
509                     .ToString());
510         }
511 
512         [TestMethod]
513         public void
514             StringDotFormatWithDifferentAmountOfArguments_WithoutPlacehold
                ()
515         {
516             var original = @"
517 using System;
518 using System.Text;
519 
520 namespace ConsoleApplication1
521 {
522     class MyClass
523     {
524         void Method(string input)
525         {
526             string s = string.Format(""abc, def"");
527         }
```

```
530         }
531 }";
532                 VerifyDiagnostic(original);
533             }
534 
535             [TestMethod]
536             public void
537                 StringDotFormatWithDifferentAmountOfArguments_WithExplici
                    
538                 ()
539             {
540                 var original = @"
541 using System;
542 using System.Text;
543 
544 namespace ConsoleApplication1
545 {
546     class MyClass
547     {
548         void Method(string input)
549         {
550             string s = string.Format(""abc {0}"", new
                        object[] {""hello""});
551         }
552     }
553 }";
554                 VerifyDiagnostic(original);
555             }
556 
557             [TestMethod]
558             public void
559                 StringDotFormatWithDifferentAmountOfArguments_WithExplici
                    
560                 ()
561             {
562                 var original = @"
563 using System;
564 using System.Text;
565 
566 namespace ConsoleApplication1
567 {
568     class MyClass
569     {
```

```
570             void Method(string input)
571             {
572                 string s = string.Format(""abc {0} {1}"", new
                        object[] {""hello""});
573             }
574         }
575 }";
576                 VerifyDiagnostic(original,
577                     StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

578                         .Rule.MessageFormat
579                         .ToString());
580             }
581
582
583             [TestMethod]
584             public void
585                 StringDotFormatWithDifferentAmountOfArguments_WithExplicitArra

586                 ()
587             {
588                 var original = @"
589 using System;
590 using System.Text;
591
592 namespace ConsoleApplication1
593 {
594     class MyClass
595     {
596         void Method(string input)
597         {
598             string s = string.Format(""abc {0} {1}"", new
                        object[] {""hello"", ""bye""});
599         }
600     }
601 }";
602                 VerifyDiagnostic(original);
603             }
604
605             [TestMethod]
606             public void
607                 StringDotFormatWithDifferentAmountOfArguments_WithExplicitArra
```

```
608             ()
609         {
610             var original = @"
611 using System;
612 using System.Text;
613
614 namespace ConsoleApplication1
615 {
616     class MyClass
617     {
618         void Method(string input)
619         {
620             var args = getArgs();
621             string s = string.Format(""abc {0} {1}"", args
                    );
622         }
623
624         object[] getArgs()
625         {
626             return new object[] {""hello"", ""bye""};
627         }
628     }
629 }";
630             VerifyDiagnostic(original);
631         }
632
633         [TestMethod]
634         public void
635             StringDotFormatWithDifferentAmountOfArguments_WithExplici
636             ()
637         {
638             var original = @"
639 using System;
640 using System.Text;
641
642 namespace ConsoleApplication1
643 {
644     class MyClass
645     {
646         void Method(string input)
647         {
648             var args = getArgs();
```

```
649             string s = string.Format(""abc {0} {1}"", args
                    );
650         }
651 
652         object[] getArgs()
653         {
654             return new object[] {""hello""};
655         }
656     }
657 }";
658             VerifyDiagnostic(original);
659         }
660 
661         [TestMethod]
662         public void
663             StringDotFormatWithDifferentAmountOfArguments_WithExplicitArra

664             ()
665         {
666             var original = @"
667 using System;
668 using System.Text;
669 
670 namespace ConsoleApplication1
671 {
672     class MyClass
673     {
674         void Method(string input)
675         {
676             object[] arr = new object[] {""hello"", ""bye""
                    "};
677             string s = string.Format(""abc {0} {1}"", arr)
                    ;
678         }
679     }
680 }";
681             VerifyDiagnostic(original);
682         }
683 
684         [TestMethod]
685         public void
686             StringDotFormatWithDifferentAmountOfArguments_WithExplicitArra
```

```
687             ()
688         {
689             var original = @"
690 using System;
691 using System.Text;
692 
693 namespace ConsoleApplication1
694 {
695     class MyClass
696     {
697         void Method(string input)
698         {
699             string s = string.Format(""abc {0} {1}"",
                    getArguments());
700         }
701 
702         object[] getArguments()
703         {
704             return new object[] {""hello"", ""bye""};
705         }
706     }
707 }";
708             VerifyDiagnostic(original);
709         }
710 
711         [TestMethod]
712         public void
713             StringDotFormatWithDifferentAmountOfArguments_WithExplici
                
714             ()
715         {
716             var original = @"
717 using System;
718 using System.Text;
719 
720 namespace ConsoleApplication1
721 {
722     class MyClass
723     {
724         void Method(string input)
725         {
726             string s = string.Format(""abc {0} {1} {2}"",
                    new object[] {""hello"", ""bye"", ""uhoh""
```

```
                }, ""test"");
        }
    }
}";
            VerifyDiagnostic(original,
                StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                    .Rule.MessageFormat
                    .ToString());
        }

        [TestMethod]
        public void
            StringDotFormatWithDifferentAmountOfArguments_WithFormatProvid
            ()
        {
            var original = @"
using System;
using System.Globalization;
using System.Text;

namespace ConsoleApplication1
{
    class MyClass
    {
        void Method(string input)
        {
            string s = string.Format(CultureInfo.
               InvariantCulture, ""abc {0}"", new object[]
                {""hello""});
        }
    }
}";
            VerifyDiagnostic(original);
        }

        [TestMethod]
        public void
            StringDotFormatWithDifferentAmountOfArguments_WithFormatProvid
            ()
        {
```

```
764             var original = @"
765 using System;
766 using System.Globalization;
767 using System.Text;
768
769 namespace ConsoleApplication1
770 {
771     class MyClass
772     {
773         void Method(string input)
774         {
775             string s = string.Format(CultureInfo.
                    InvariantCulture, ""abc {0}{1}"", new
                    object[] {""hello""});
776         }
777     }
778 }";
779             VerifyDiagnostic(original,
780                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
781                     .Rule.MessageFormat
782                     .ToString());
783         }
784
785         [TestMethod]
786         public void
787             StringDotFormatWithDifferentAmountOfArguments_WithFormatP
788             ()
789         {
790             var original = @"
791 using System;
792 using System.Globalization;
793 using System.Text;
794
795 namespace ConsoleApplication1
796 {
797     class MyClass
798     {
799         void Method(string input)
800         {
801             var args = new object[] {""hello""};
802             string s = string.Format(CultureInfo.
```

```
                InvariantCulture, ""abc {0}{1}"", args);
803         }
804     }
805 }";
806             VerifyDiagnostic(original);
807         }
808 
809         [TestMethod]
810         public void
811             StringDotFormatWithDifferentAmountOfArguments_WithFormatProvid
                
812             ()
813         {
814             var original = @"
815 using System;
816 using System.Globalization;
817 using System.Text;
818 
819 namespace ConsoleApplication1
820 {
821     class MyClass
822     {
823         void Method(string input)
824         {
825             string s = string.Format(CultureInfo.
                    InvariantCulture, ""abc {0}{1}"", getArgs()
                    );
826         }
827 
828         object[] getArgs()
829         {
830             return new object[] {""hello""};
831         }
832     }
833 }";
834             VerifyDiagnostic(original);
835         }
836 
837         [TestMethod]
838         public void
839             StringDotFormatWithDifferentAmountOfArguments_WithConstantForm
                
840             ()
```

```
841         {
842             var original = @"
843 using System;
844 using System.Globalization;
845 using System.Text;
846 
847 namespace ConsoleApplication1
848 {
849     class MyClass
850     {
851         void Method(string input)
852         {
853             const string format = ""{0}{1}"";
854             string s = string.Format(format, ""arg"");
855         }
856     }
857 }";
858             VerifyDiagnostic(original,
859                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                    .Rule.MessageFormat
860                     .ToString());
861         }
862 
863         [TestMethod]
864         public void
865             StringDotFormatWithDifferentAmountOfArguments_WithConstar
                ()
866         {
867             var original = @"
868 using System;
869 using System.Globalization;
870 using System.Text;
871 
872 namespace ConsoleApplication1
873 {
874     class MyClass
875     {
876         void Method(string input)
877         {
878             const string a = ""{0}"";
879             const string b = ""{1}"";
880 
881 
```

```
882             const string format = a + b;
883             string s = string.Format(format, ""arg"");
884         }
885     }
886 }";
887             VerifyDiagnostic(original,
888                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
889                     .Rule.MessageFormat
890                     .ToString());
891         }
892 
893         [TestMethod]
894         public void
895             StringDotFormatWithDifferentAmountOfArguments_WithStaticImport
896             ()
897         {
898             var original = @"
899 using static System.String;
900 
901 namespace ConsoleApplication1
902 {
903     class MyClass
904     {
905         void Method(string input)
906         {
907             const string a = ""{0}"";
908             const string b = ""{1}"";
909             const string format = a + b;
910             string s = Format(format, ""arg"");
911         }
912     }
913 }";
914             VerifyDiagnostic(original,
915                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
916                     .Rule.MessageFormat
917                     .ToString());
918         }
919 
920         [TestMethod]
921         public void
```

```
922             StringDotFormatWithDifferentAmountOfArguments_WithParenth

923             ()
924         {
925             var original = @"
926 using System;
927 using System.Globalization;
928 using System.Text;
929 
930 namespace ConsoleApplication1
931 {
932     class MyClass
933     {
934         void Method(string input)
935         {
936             string s = string.Format((""{0}{1}""), ""arg""
                    );
937         }
938     }
939 }";
940             VerifyDiagnostic(original,
941                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

942                     .Rule.MessageFormat
943                     .ToString());
944         }
945 
946         [TestMethod]
947         public void
948             StringDotFormatWithDifferentAmountOfArguments_WithConsole

949             ()
950         {
951             var original = @"
952 using System;
953 
954 namespace ConsoleApplication1
955 {
956     class MyClass
957     {
958         void Method(string input)
959         {
960             Console.WriteLine(""{0}{1}"", ""arg"");
```

```
961         }
962     }
963 }";
964             VerifyDiagnostic(original,
965                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                    .Rule.MessageFormat
966                     .ToString());
967         }
968
969         [TestMethod]
970         public void
971             StringDotFormatWithDifferentAmountOfArguments_WithConsoleWrite
                ()
972             {
973             var original = @"
974 using System;
975
976 namespace ConsoleApplication1
977 {
978     class MyClass
979     {
980         void Method(string input)
981         {
982             Console.WriteLine(""{0}{1}{2}"", ""arg"", ""
                    arg2"");
983         }
984     }
985 }";
986             VerifyDiagnostic(original,
987                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
                    .Rule.MessageFormat
988                     .ToString());
989         }
990
991         [TestMethod]
992         public void
993             StringDotFormatWithDifferentAmountOfArguments_WithConsoleWrite
                ()
994         {
```

```
999             var original = @"
1000 using System;
1001
1002 namespace ConsoleApplication1
1003 {
1004     class MyClass
1005     {
1006         void Method(string input)
1007         {
1008             Console.WriteLine(""{0}{1}{2}"", new object[]
                    { ""arg"", ""arg2"" });
1009         }
1010     }
1011 }";
1012             VerifyDiagnostic(original,
1013                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

1014                     .Rule.MessageFormat
1015                     .ToString());
1016         }
1017
1018         [TestMethod]
1019         public void
1020             StringDotFormatWithDifferentAmountOfArguments_WithConsole

1021             ()
1022         {
1023             var original = @"
1024 using System;
1025
1026 namespace ConsoleApplication1
1027 {
1028     class MyClass
1029     {
1030         void Method(string input)
1031         {
1032             var args = new object[] { ""arg"", ""arg2"" };
1033             Console.WriteLine(""{0}{1}{2}"", args);
1034         }
1035     }
1036 }";
1037             VerifyDiagnostic(original);
1038         }
```

```
1039
1040            [TestMethod]
1041            public void
1042                StringDotFormatWithDifferentAmountOfArguments_WithSimilarInvoc
                    ()
1043                ()
1044            {
1045                var original = @"
1046    using System;
1047
1048    namespace ConsoleApplication1
1049    {
1050        class MyClass
1051        {
1052            void Method(string input)
1053            {
1054                Other(""{0}{1}{2}"", new object[] { ""arg"", "
                        "arg2"" }, 5);
1055            }
1056
1057            void Other(string format, object[] args, int
                    something)
1058            {
1059            }
1060        }
1061    }";
1062                VerifyDiagnostic(original);
1063            }
1064
1065            [TestMethod]
1066            public void
1067                StringDotFormatWithDifferentAmountOfArguments_WithSimilarInvoc
                    ()
1068                ()
1069            {
1070                var original = @"
1071    using System;
1072
1073    namespace ConsoleApplication1
1074    {
1075        class MyClass
1076        {
1077            void Method(string input)
```

```
1078            {
1079                Other(""{0}{1}{2}"", new object[] { ""arg"", "
                        "arg2"" }, new object[] {});
1080            }
1081
1082            void Other(string format, object[] args, object[]
                    args2)
1083            {
1084            }
1085        }
1086    }";
1087                VerifyDiagnostic(original);
1088            }
1089
1090            [TestMethod]
1091            public void
1092                StringDotFormatWithDifferentAmountOfArguments_WithObjectA

1093                ()
1094            {
1095                var original = @"
1096    using System;
1097
1098    namespace ConsoleApplication1
1099    {
1100        class MyClass
1101        {
1102            void Method(string input)
1103            {
1104                var args = new object[] { ""a"", ""b""};
1105                string s = string.Format(""{0}{1}"", (object)
                        args);
1106            }
1107        }
1108    }";
1109                VerifyDiagnostic(original,
1110                    StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

1111                        .Rule.MessageFormat
1112                        .ToString());
1113            }
1114
1115            [TestMethod]
```

```
1116            public void
1117                StringDotFormatWithDifferentAmountOfArguments_WithParenthesize

1118                ()
1119            {
1120                var original = @"
1121    using System;
1122    using System.Text;
1123
1124    namespace ConsoleApplication1
1125    {
1126        class MyClass
1127        {
1128            void Method(string input)
1129            {
1130                string s = string.Format(""abc {0}{1}"", (new
                        [] { 5 }));
1131            }
1132        }
1133    }";
1134                VerifyDiagnostic(original,
1135                    StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

1136                        .Rule.MessageFormat
1137                        .ToString());
1138            }
1139
1140            [TestMethod]
1141            public void
1142                StringDotFormatWithDifferentAmountOfArguments_WithParenthesize

1143                ()
1144            {
1145                var original = @"
1146    using System;
1147    using System.Text;
1148
1149    namespace ConsoleApplication1
1150    {
1151        class MyClass
1152        {
1153            void Method(string input)
1154            {
```

```
1155                string s = string.Format(""abc {0}{1}"", (new
                        [] { 5, 2 }));
1156            }
1157        }
1158    }";
1159                VerifyDiagnostic(original);
1160            }
1161
1162            [TestMethod]
1163            public void
1164                StringDotFormatWithDifferentAmountOfArguments_WithMethod_

1165                ()
1166            {
1167                var original = @"
1168    using System;
1169    using System.Text;
1170
1171    namespace ConsoleApplication1
1172    {
1173        class MyClass
1174        {
1175            void Method(string input)
1176            {
1177                string s = string.Format(""abc {0}{1}"", Other
                        ());
1178            }
1179
1180            int Other()
1181            {
1182                return 4;
1183            }
1184        }
1185    }";
1186                VerifyDiagnostic(original,
1187                    StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

1188                        .Rule.MessageFormat
1189                        .ToString());
1190            }
1191
1192            [TestMethod]
1193            public void
```

```
1194            StringDotFormatWithDifferentAmountOfArguments_WithMethod_WithP

1195            ()
1196        {
1197            var original = @"
1198using System;
1199using System.Text;
1200
1201namespace ConsoleApplication1
1202{
1203    class MyClass
1204    {
1205        void Method(string input)
1206        {
1207            string s = string.Format(""abc {0}{1}"", (
                    Other()));
1208        }
1209
1210        int Other()
1211        {
1212            return 4;
1213        }
1214    }
1215}";
1216            VerifyDiagnostic(original,
1217                StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

1218                    .Rule.MessageFormat
1219                    .ToString());
1220        }
1221
1222        [TestMethod]
1223        public void
1224            StringDotFormatWithDifferentAmountOfArguments_WithMethod_ThatD

1225            ()
1226        {
1227            var original = @"
1228using System;
1229using System.Text;
1230
1231namespace ConsoleApplication1
1232{
```

```
1233        class MyClass
1234        {
1235            void Method(string input)
1236            {
1237                string s = string.Format(""abc {0}{1}{2}"",
                        Other(), Other());
1238            }
1239
1240            int Other()
1241            {
1242                return 4;
1243            }
1244        }
1245    }";
1246                VerifyDiagnostic(original,
1247                    StringDotFormatWithDifferentAmountOfArgumentsAnalyzer

1248                        .Rule.MessageFormat
1249                        .ToString());
1250            }
1251
1252            [TestMethod]
1253            public void
1254                StringDotFormatWithDifferentAmountOfArguments_WithMethod_

1255                ()
1256            {
1257                var original = @"
1258    using System;
1259    using System.Text;
1260
1261    namespace ConsoleApplication1
1262    {
1263        class MyClass
1264        {
1265            void Method(string input)
1266            {
1267                string s = string.Format(""abc {0}{1}"", Other
                        (), Another());
1268            }
1269
1270            int Other()
1271            {
```

```
1272                return 4;
1273            }
1274
1275            object Another()
1276            {
1277                return 5;
1278            }
1279        }
1280    }";
1281                VerifyDiagnostic(original);
1282            }
1283
1284            [TestMethod]
1285            public void
1286                StringDotFormatWithDifferentAmountOfArguments_WithConcatArgs
                    ()
1287
1288            {
1289                var original = @"
1290    using System;
1291    using System.Linq;
1292
1293    namespace ConsoleApplication1
1294    {
1295        class MyClass
1296        {
1297            void Method(string input)
1298            {
1299                string.Format(""{0}{1}"", new[] { 1 }.Concat(
                        new[] {2}).ToArray());
1300            }
1301        }
1302    }";
1303                VerifyDiagnostic(original);
1304            }
1305
1306            [TestMethod]
1307            public void
1308                StringDotFormatWithDifferentAmountOfArguments_WithLackingConca
                    ()
1309
1310            {
1311                var original = @"
```

```
1312 using System;
1313 using System.Linq;
1314
1315 namespace ConsoleApplication1
1316 {
1317     class MyClass
1318     {
1319         void Method(string input)
1320         {
1321             string.Format(""{0}{1}{2}"", new[] { 1 }.
                    Concat(new[] {2}).ToArray());
1322         }
1323     }
1324 }";
1325             VerifyDiagnostic(original);
1326         }
1327
1328         [TestMethod]
1329         public void
1330             StringDotFormatWithDifferentAmountOfArguments_WithObject
                ()
1331         {
1332             var original = @"
1333 using System;
1334 using System.Linq;
1335
1336 namespace ConsoleApplication1
1337 {
1338     class MyClass
1339     {
1340         void Method(string input)
1341         {
1342             string.Format(""{0}{1}"", new MyClass { Prop1
                    = 5, Prop2 = 6});
1343         }
1344
1345       public int Prop1 { get; set; }
1346       public int Prop2 { get; set; }
1347     }
1348 }";
1349             VerifyDiagnostic(original,
1350                 StringDotFormatWithDifferentAmountOfArgumentsAnalyzer
1351
```

```
1352                        .Rule.MessageFormat
1353                        .ToString());
1354            }
1355        }
1356    }
```

A.3 PlaceholderHelpers.cs

```
1   using System.Text.RegularExpressions;
2
3   namespace VSDiagnostics.Diagnostics.
4       Strings
5   {
6       //TODO: tests
7       internal static class
8           PlaceholderHelpers
9       {
10          /// <summary>
11          ///     Removes all curly braces and formatting
                definitions from the placeholder
12          /// </summary>
13          /// <param name="input">The placeholder entry to
                parse.</param>
14          /// <returns>Returns the placeholder index.</
                returns>
15          internal static string
16              GetPlaceholderIndex(
17              string input)
18          {
19              var temp = input.Trim('{',
20                  '}');
21              var colonIndex =
22                  temp.IndexOf(':');
23              if (colonIndex > 0)
24              {
25                  return
26                      temp.Remove(
27                          colonIndex);
28              }
29
30              return temp;
31          }
```

```
32
33          /// <summary>
34          ///     Get all elements in a string that are
                enclosed by an uneven amount of curly brackets
                (to account for escaped
35          ///     brackets).
36          ///     The result will be elements that are
                either plain integers or integers with a format
                 appended to it, delimited by a
37          ///     colon.
38          /// </summary>
39          /// <param name="input">The format string with
                placeholders.</param>
40          /// <returns>Returns a collection of matches
                according to the regex.</returns>
41          internal static MatchCollection
42              GetPlaceholders(string input)
43          {
44              // This regex uses a named group so we can
                    easily access the actual value
45              return Regex.Matches(input,
46                  @"(?<!\{)\{(?:\{\{)*((?<index>\d+)(?::.*?)
                        ?)\}(?:\}\})*(?!\})");
47          }
48
49          /// <summary>
50          ///     Returns all elements from the input -
                split on the placeholders - and includes the
                placeholders themselves as well.
51          ///     This method is useful if you want to make
                use of the rest of the string as well.
52          /// </summary>
53          internal static string[]
54              GetPlaceholdersSplit(
55              string input)
56          {
57              return Regex.Split(input,
58                  @"(?<!\{)\{(?:\{\{)*(\d+(?::.*?)?)
                        \}(?:\}\})*(?!\})");
59          }
60      }
61  }
```

A.4 SyntaxWalker

```
1  using System;
2  using System.Linq;
3  using Microsoft.CodeAnalysis;
4  using Microsoft.CodeAnalysis.CSharp;
5  using
6      Microsoft.CodeAnalysis.CSharp.Syntax
7      ;
8
9  namespace ConsoleApplication1
10 {
11     public class MySyntaxWalker :
12         CSharpSyntaxWalker
13     {
14         private readonly int
15             _spacesPerLevel = 4;
16
17         private int _indentLevel;
18
19         internal static void Execute()
20         {
21             const string source = @"
22 public class MyOuterClass
23 {
24     public void FirstMethod(int x, int y) { }
25     private void SecondMethod(string s) { }
26
27     public int Property1 { get; set }
28     internal StringBuilder Property2 { get; set }
29
30     private class MyInnerClass
31     {
32         protected int InnerMethod(Func<int, int> f) {
               return 42; }
33     }
34 }
35 ";
36             var tree =
37                 CSharpSyntaxTree
38                     .ParseText(source);
39             new MySyntaxWalker().Visit(
40                 tree.GetRoot());
```

```
41          }
42
43          private string GetAccessLevel(
44              SyntaxTokenList modifiers)
45          {
46              var levels = new[]
47              {
48                  SyntaxKind.PublicKeyword,
49                  SyntaxKind
50                      .ProtectedKeyword,
51                  SyntaxKind
52                      .InternalKeyword,
53                  SyntaxKind
54                      .PrivateKeyword
55              };
56              var foundModifier =
57                  modifiers.FirstOrDefault
58                      (modifier =>
59                          levels.Any(
60                              level =>
61                                  modifier
62                                      .Kind
63                                      () ==
64                                  level));
65              return
66                  foundModifier.Kind() ==
67                  SyntaxKind.PublicKeyword
68                      ? "+"
69                      : foundModifier.Kind
70                          () ==
71                        SyntaxKind
72                            .InternalKeyword ||
73                        foundModifier.Kind
74                            () ==
75                        SyntaxKind
76                            .ProtectedKeyword
77                          ? "#"
78                          : "-";
79          }
80
81          private void Write(string text,
82                             string
83                                 accessLevel,
```

```
84                              Action
85                                  visitBase)
86          {
87              _indentLevel++;
88              var indentation =
89                  new string(' ',
90                      _indentLevel*
91                      _spacesPerLevel);
92              Console.WriteLine(
93                  $"{indentation}{accessLevel} {text}");
94              visitBase();
95              _indentLevel--;
96          }
97
98          public override void
99              VisitClassDeclaration(
100             ClassDeclarationSyntax node)
101         {
102             Write(
103                 node.Identifier
104                     .ValueText,
105                 GetAccessLevel(
106                     node.Modifiers),
107                 () =>
108                     base
109                         .VisitClassDeclaration
110                         (node));
111         }
112
113         public override void
114             VisitMethodDeclaration(
115             MethodDeclarationSyntax node)
116         {
117             var identifier =
118                 node.Identifier
119                     .ValueText;
120             var parameters =
121                 string.Join(", ",
122                     node.ParameterList
123                         .Parameters
124                         .Select(
125                             x => x.Type));
126             var returnType =
```

```
127                 node.ReturnType;
128
129             Write(
130                 $"{identifier}({parameters}) : {returnType
                        }",
131                 GetAccessLevel(
132                     node.Modifiers),
133                 () =>
134                     base
135                         .VisitMethodDeclaration
136                         (node));
137         }
138
139         public override void
140             VisitPropertyDeclaration(
141             PropertyDeclarationSyntax
142                 node)
143         {
144             var identifier =
145                 node.Identifier
146                     .ValueText;
147             Write(
148                 $"prop: {identifier} : {node.Type}",
149                 GetAccessLevel(
150                     node.Modifiers),
151                 () =>
152                     base
153                         .VisitPropertyDeclaration
154                         (node));
155         }
156     }
157 }
```

A.5 SyntaxRewriter

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using Microsoft.CodeAnalysis;
5 using Microsoft.CodeAnalysis.CSharp;
6 using
7     Microsoft.CodeAnalysis.CSharp.Syntax
8     ;
```

```
9  using Microsoft.CodeAnalysis.Formatting;
10
11 namespace ConsoleApplication1
12 {
13     public class MyRewriter :
14         CSharpSyntaxRewriter
15     {
16         internal static void Execute()
17         {
18             const string source = @"
19 public class MyOuterClass
20 {
21     private int _myField;
22     private string anotherField;
23     private double x, _y;
24
25
26     private class MyInnerClass
27     {
28         private int _innerField;
29         public int anotherInnerField;
30     }
31 }
32 ";
33             var tree =
34                 CSharpSyntaxTree
35                     .ParseText(source);
36             var rewriter =
37                 new MyRewriter().Visit(
38                     tree.GetRoot());
39             var formattedTree =
40                 Formatter.Format(
41                     rewriter,
42                     Formatter.Annotation,
43                     new AdhocWorkspace());
44             Console.WriteLine(
45                 formattedTree
46                     .ToFullString());
47         }
48
49         public override SyntaxNode
50             VisitFieldDeclaration(
51             FieldDeclarationSyntax node)
```

```
52          {
53              var newDeclarators =
54                  new List
55                      <
56                          VariableDeclaratorSyntax
57                      >();
58
59              foreach (
60                  var declarator in
61                  node.Declaration
62                      .Variables)
63              {
64                  var currentIdentifier =
65                  declarator
66                      .Identifier;
67                  if (
68                      !currentIdentifier
69                          .ValueText
70                          .StartsWith("_") &&
71                      node.Modifiers.Any(
72                          x =>
73                              x.Kind() ==
74                              SyntaxKind
75                                  .PrivateKeyword))
76                  {
77                      var newIdentifier =
78                      SyntaxFactory
79                          .Identifier(
80                              currentIdentifier
81                                  .LeadingTrivia,
82                              "_" +
83                              currentIdentifier
84                                  .ValueText,
85                              currentIdentifier
86                                  .TrailingTrivia);
87                      var newDeclarator =
88                      declarator
89                          .WithIdentifier
90                          (newIdentifier);
91                      newDeclarators.Add(
92                          newDeclarator);
93                  }
94                  else
```

```
95                  {
96                      newDeclarators.Add(
97                          declarator);
98                  }
99              }
100             var newDeclaration =
101                 SyntaxFactory
102                     .VariableDeclaration
103                     (node.Declaration
104                          .Type,
105                         SyntaxFactory
106                             .SeparatedList
107                             (newDeclarators));
108             var newField =
109                 SyntaxFactory
110                     .FieldDeclaration(
111                         node
112                             .AttributeLists,
113                         node.Modifiers,
114                         newDeclaration)
115                     .WithAdditionalAnnotations
116                     (Formatter
117                         .Annotation);
118             return newField;
119         }
120     }
121 }
```