# Creating a Code Analyzer using F#

Nov 2, 2015

**Note**: This article covers creating a C#/VB code analyzer using F#. At this time there is no Roslyn support for analyzing F# code.

In the past we have covered creating code analyzers using C# and VB. Creating an analyzer in F# is just as easy. I have just started the process of learning F# and figured an analyzer would be a great test project to learn on. There aren't official templates for F# analyzers, but you can take the C# templates and use those as a starting point for F#.

To start, make sure you have the latest version of the .Net Compiler Platform SDK installed.

Next, you'll want to create an "Analyzer with Code Fix (NuGet + VSIX)" project under the Visual C#->Extensibility group in the project templates.


Once you have the project created, add a new F# library project to the solution. With the new project added to the solution, you'll next want to modify the VSIX project to deploy the F# project instead of the C# project. To do this, modify the source.extension.vsixmanifest file and go to the Assets tab. Switch the project on both the `Analyzer` and `MefComponent` to be the new F# library.


Now you can remove the C# analyzer and test project form the solution.

You solution is now set, but the F# project needs the appropriate references in order to work with analyzers. Add the `Microsoft.CodeAnalysis` NuGet package to the F# project.

Now we can start coding the analyzer. We'll implement the same basic analyzer that comes with the C# samples, where it raises a diagnostic whenever there are lowercase characters in type names.

We'll start creating our analyzer by importing some namespaces we'll need later in the code.

```
namespace FSharpFirstAnalyzer
open Microsoft.CodeAnalysis
open Microsoft.CodeAnalysis.Diagnostics
open System.Collections.Immutable
```

```
open System.Linq
open System
```

Next we'll declare our analyzer and inherit from the DiagnosticAnalyzer base class. Notice that we are registering our analyzer as a C# only analyzer.

```
[<DiagnosticAnalyzer(Microsoft.CodeAnalysis.LanguageNames.CSharp)>]
type public MyFirstFSAnalyzer() =
    inherit DiagnosticAnalyzer()
```

Now we can create a descriptor for our diagnostic and override the `SupportedDiagnostics` property to return our diagnostics.

```
let descriptor = DiagnosticDescriptor("FSharpIsLowerCase",
                          "Types cannot contain lowercase letters",
                          "{0} contains lowercase letters" ,
                          "Naming",
                          DiagnosticSeverity.Warning,
                          true,
                          "User declared types should not contain lowercase
letters.",
                          null)

override x.SupportedDiagnostics with get() = ImmutableArray.Create(descriptor)
```

Finally, we can do our work in the `Initialize` override. We'll create a symbol analysis function and check if the symbol has any lowercase characters. To do this, we'll perform a match on the `Symbol.Name`. Finally we'll register that function with the context passed into the Initialize method.

```
    override x.Initialize (context: AnalysisContext) =
        let isLower = System.Func<_,_>(fun l -> Char.IsLower(l))
        let analyze (ctx: SymbolAnalysisContext) =
            match ctx.Symbol with
                | z when z.Name.ToCharArray().Any(isLower) ->
                    let d = Diagnostic.Create(descriptor, z.Locations.First(),
z.Name)
                    ctx.ReportDiagnostic(d)
                | _->()

        context.RegisterSymbolAction(analyze, SymbolKind.NamedType)
```

At this point, we can run the analyzer solution and a new Visual Studio instance will appear. This is referred to as the experimental instance and has completely isolated settings from the instance in which you do

your main development. Create a simple C# console application and open the `Program.cs`. You should get a diagnostic on the `Program` class indicating it contains lowercase letters.


lowercaseLetters

The full code for our F# analyzer is:

```
namespace FSharpFirstAnalyzer
open Microsoft.CodeAnalysis
open Microsoft.CodeAnalysis.Diagnostics
open System.Collections.Immutable
open System.Linq
open System

[<DiagnosticAnalyzer(Microsoft.CodeAnalysis.LanguageNames.CSharp)>]
type public MyFirstFSAnalyzer() =
    inherit DiagnosticAnalyzer()
    let descriptor = DiagnosticDescriptor("FSharpIsLowerCase",
                            "Types cannot contain lowercase letters",
                            "{0} contains lowercase letters" ,
                            "Naming",
                            DiagnosticSeverity.Warning,
                            true,
                            "User declared types should not contain lowercase
letters.",
                            null)

    override x.SupportedDiagnostics with get() = ImmutableArray.Create(descriptor)

    override x.Initialize (context: AnalysisContext) =
        let isLower = System.Func<_,_>(fun l -> Char.IsLower(l))
        let analyze (ctx: SymbolAnalysisContext) =
            match ctx.Symbol with
                | z when z.Name.ToCharArray().Any(isLower) ->
                    let d = Diagnostic.Create(descriptor, z.Locations.First(),
z.Name)
                    ctx.ReportDiagnostic(d)
                | _->()

        context.RegisterSymbolAction(analyze, SymbolKind.NamedType)
```

As you can see, creating an analyzer in F# is possible, and once you have the tooling setup, the development flow is not much different than that of a C# or VB analyzer. Overall, I think matching functionality in F# provides interesting possibilities when creating analyzers for Roslyn.

**wk** • 2 years ago

I follow your article, but I got error get_SupportedDiagnostics does not have an implementation. Don't know what wrong, here my repository https://github.com/wk-j/cod...

17 • Reply • Share ›

**Nejc Skofic** wk • 2 years ago

For anyone still having those problems: you should really watch out for referenced DLL's versions. VS requires System.Collections.Immutable of version 1.1.36 and not 1.2.0. This is why you get error that 'get_SupportedDiagnostics does not have an implementation', since to .net runtime those two assemblies are different. You should create analyzer from template and add nuget packages of exactly the same version as defined in template. Also you do not have to include System.Collections.Immutable to your VSIX since correct version is already loaded in VS runtime.

• Reply • Share ›