FREE eBook

LEARNING roslyn

#roslyn

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: roslyn

It is an unofficial and free roslyn ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official roslyn.

Chapter 1: Getting started with roslyn

Remarks

To start with Roslyn, take a look at:

- Syntax Tree API
- Semantic model API
- *Add more topics here*.

Examples

Installation or Setup

To start tinkering with Roslyn you will need the following NuGet packages:

- The C# and VB compilers - `Microsoft.Net.Compilers`. To install it you can run the following command in the Package Manager Console:

  ```
  nuget install Microsoft.Net.Compilers
  ```

- The Language APIs and Services - `Microsoft.CodeAnalysis`. To install it you can run the following command in the Package Manager Console:

  ```
  nuget install Microsoft.CodeAnalysis
  ```

Additionally it is a good to install the .NET Compiler Platform SDK Templates, that can be found here. This will get you:

- Templates for both C# and Visual Basic that enable the creation of Analyzers, CodeFixes and stand-alone analysis tools.
- The Syntax Visualizer tool for Visual Studio(`View -> Other Windows -> Syntax Visualizer`), which is extremely usefully for examining the syntax tree of existing code.

Additional tools and resources

- The Roslyn Quoter

A tool for converting an sample C# program to syntax tree API calls. The tool itself can be found here.

- Enhanced source viewer

An easy way to view the Roslyn source code can be found here.

Read Getting started with roslyn online: https://riptutorial.com/roslyn/topic/2905/getting-started-with-roslyn

Chapter 2: Analyze source code with Roslyn

Examples

Introspective analysis of an analyzer in C#

1. Create a new **Console Application**
2. Add the **NuGet** package `Microsoft.CodeAnalysis`
3. Import the namespaces `Microsoft.CodeAnalysis.MSBuild`, `System.Linq` and `Microsoft.CodeAnalysis.CSharp.Syntax`
4. Write the following example code in the `Main` method:

```
// Declaring a variable with the current project file path.
const string projectPath = @"C:\<your path to the project\<project file name>.csproj";

// Creating a build workspace.
var workspace = MSBuildWorkspace.Create();

// Opening this project.
var project = workspace.OpenProjectAsync(projectPath).Result;

// Getting the compilation.
var compilation = project.GetCompilationAsync().Result;

// As this is a simple single file program, the first syntax tree will be the current file.
var syntaxTree = compilation.SyntaxTrees.First();

// Getting the root node of the file.
var rootSyntaxNode = syntaxTree.GetRootAsync().Result;

// Finding all the local variable declarations in this file and picking the first one.
var firstLocalVariablesDeclaration =
rootSyntaxNode.DescendantNodesAndSelf().OfType<LocalDeclarationStatementSyntax>().First();

// Getting the first declared variable in the declaration syntax.
var firstVariable = firstLocalVariablesDeclaration.Declaration.Variables.First();

// Getting the text of the initialized value.
var variableInitializer = firstVariable.Initializer.Value.GetFirstToken().ValueText;

// This will print to screen the value assigned to the projectPath variable.
Console.WriteLine(variableInitializer);

Console.ReadKey();
```

When running the project, you will see the variable declared on top printed to the screen. This means that you successfully self analysed a project and found a variable in it.

Analyze a simple "Hello World" application in C#

Create a new console application with one line in the `Main` method: `Console.WriteLine("Hello World")`

Remember the path to the `.csproj` file and replace it in the example.

Create a new **Console Application** and install the `Microsoft.CodeAnalysis` NuGet package and try the following code:

```
const string projectPath = @"C:\HelloWorldApplication\HelloWorldProject.csproj";

// Creating a build workspace.
var workspace = MSBuildWorkspace.Create();

// Opening the Hello World project.
var project = workspace.OpenProjectAsync(projectPath).Result;

// Getting the compilation.
var compilation = project.GetCompilationAsync().Result;

foreach (var tree in compilation.SyntaxTrees)
{
    Console.WriteLine(tree.FilePath);

    var rootSyntaxNode = tree.GetRootAsync().Result;

    foreach (var node in rootSyntaxNode.DescendantNodes())
    {
        Console.WriteLine($" *** {node.Kind()}");
        Console.WriteLine($"     {node}");
    }
}

Console.ReadKey();
```

This will print all the files and all the **syntax nodes** in your **Hello World** project.

Analyze a simple "Hello World" application in VB.NET

Create a new console application with one line in the `Main` method: `Console.WriteLine("Hello World")`

Remember the path to the `.vbproj` file and replace it in the example.

Create a new **Console Application** and install the `Microsoft.CodeAnalysis` NuGet package and try the following code:

```
Const projectPath = "C:\HelloWorldApplication\HelloWorldProject.vbproj"

' Creating a build workspace.
Dim workspace = MSBuildWorkspace.Create()

' Opening the Hello World project.
Dim project = workspace.OpenProjectAsync(projectPath).Result

' Getting the compilation.
Dim compilation = project.GetCompilationAsync().Result

For Each tree In compilation.SyntaxTrees

    Console.WriteLine(tree.FilePath)
    Dim rootSyntaxNode = tree.GetRootAsync().Result

    For Each node In rootSyntaxNode.DescendantNodes()

        Console.WriteLine($" *** {node.Kind()}")
        Console.WriteLine($"     {node}")
    Next
Next

Console.ReadKey()
```

This will print all the files and all the **syntax nodes** in your **Hello World** project.

Parse source code from text in C#

```
var syntaxTree = CSharpSyntaxTree.ParseText(
@"using System;
using System.Collections;
using System.Linq;
using System.Text;

namespace HelloWorldApplication
{
class Program
{
static void Main(string[] args)
{
Console.WriteLine(""Hello World"");
}
}
}");

var root = syntaxTree.GetRoot() as CompilationUnitSyntax;

var namespaceSyntax = root.Members.OfType<NamespaceDeclarationSyntax>().First();

var programClassSyntax = namespaceSyntax.Members.OfType<ClassDeclarationSyntax>().First();

var mainMethodSyntax = programClassSyntax.Members.OfType<MethodDeclarationSyntax>().First();

Console.WriteLine(mainMethodSyntax.ToString());

Console.ReadKey();
```

This example will print the `Main` method from the text analyzing the syntax.

Get the type of 'var'

To get the actual type for a variable declared using `var`, call `GetSymbolInfo()` on the `SemanticModel`. You can open an existing solution using `MSBuildWorkspace`, then enumerate its projects and their documents. Use a document to obtain its `SyntaxRoot` and `SemanticModel`, then look for `VariableDeclarations` and retrieve the symbols for the `Type` of a declared variable like this:

```
var workspace = MSBuildWorkspace.Create();
var solution = workspace.OpenSolutionAsync("c:\\path\\to\\solution.sln").Result;

foreach (var document in solution.Projects.SelectMany(project => project.Documents))
{
    var rootNode = document.GetSyntaxRootAsync().Result;
    var semanticModel = document.GetSemanticModelAsync().Result;

    var variableDeclarations = rootNode
            .DescendantNodes()
            .OfType<LocalDeclarationStatementSyntax>();
    foreach (var varDeclaration in variableDeclarations)
    {
        var symbolInfo = semanticModel.GetSymbolInfo(varDeclaration.Declaration.Type);
        var typeSymbol = symbolInfo.Symbol; // the type symbol for the variable..
    }
}
```

Read Analyze source code with Roslyn online: https://riptutorial.com/roslyn/topic/4712/analyze-source-code-with-roslyn

Chapter 3: Change source code with Roslyn

Introduction

Practical examples of using Roslyn for source code transformations.

Remarks

- Roslyn syntax trees are immutable. By calling a method like ReplaceNodes we generate a new node rather than modifying the existing one. This requires you to always change the object you have been working on.

Examples

Replace existing Attributes for all methods in C# using the syntax tree

The following snippet replaces all Attributes called `PreviousAttribute` by an Attribute called `ReplacementAttribute` for an entire solution. The sample manually searches the Syntax tree and replaces all affected nodes.

```
    static async Task<bool> ModifySolution(string solutionPath)
    {
        using (var workspace = MSBuildWorkspace.Create())
        {
            // Selects a Solution File
            var solution = await workspace.OpenSolutionAsync(solutionPath);
            // Iterates through every project
            foreach (var project in solution.Projects)
            {
                // Iterates through every file
                foreach (var document in project.Documents)
                {
                    // Selects the syntax tree
                    var syntaxTree = await document.GetSyntaxTreeAsync();
                    var root = syntaxTree.GetRoot();
                    // Finds all Attribute Declarations in the Document
                    var existingAttributesList =
root.DescendantNodes().OfType<AttributeListSyntax>()
                        // Where the Attribute is declared on a method
                        .Where(curr => curr.Parent is MethodDeclarationSyntax)
                        // And the attribute is named "PreviousAttribute"
                        .Where(curr => curr.Attributes.Any(currentAttribute =>
currentAttribute.Name.GetText().ToString() == "PreviousAttribute"))
                        .ToList();
                    if (existingAttributesList.Any())
                    {
                        // Generates a replacement for every attribute
                        var replacementAttribute = SyntaxFactory.AttributeList(
                            SyntaxFactory.SingletonSeparatedList(

SyntaxFactory.Attribute(SyntaxFactory.IdentifierName("ReplacementAttribute"),
                                SyntaxFactory.AttributeArgumentList(
                                    SyntaxFactory.SeparatedList(new[]
                                    {
                                    SyntaxFactory.AttributeArgument(
                                        SyntaxFactory.LiteralExpression(
                                            SyntaxKind.StringLiteralExpression,
SyntaxFactory.Literal(@"Sample"))
                                        )
                                    })))));
                        // Replaces all attributes at once.
                        // Note that you should not use root.ReplaceNode
                        // since it would only replace the first note
                        root = root.ReplaceNodes(existingAttributesList, (node, n2) =>
replacementAttribute);
                        // Exchanges the document in the solution by the newly generated
document
                        solution = solution.WithDocumentSyntaxRoot(document.Id, root);
                    }
                }
            }
            // applies the changes to the solution
            var result = workspace.TryApplyChanges(solution);
            return result;
        }
    }
```

The above example can be tested for the following class:

```
public class Program
{
    [PreviousAttribute()]
    static void Main(string[] args)
    {
    }
}
```

You should not use the Methode root.ReplaceNode to replace multiple nodes. Since the tree is immutable you will be working on different objects. Using the following snippet in the above example would not yield the expected result:

```
foreach(var node in existingAttributesList){
    root = root.ReplaceNode(node, replacementAttribute);
}
```

The first call to `ReplaceNode` would create a new root element. However the elements in `existingAttributesList` belong to a different root (the previous root element) and cannot be replaced because of this. This would result in the first Attribute being replaced and the following Attributes remaining unchanged since all consecutive calls would be performed on a node not present in the new tree.

Replace existing Attributes for all methods in C# using a SyntaxRewriter

The following snippet replaces all Attributes called "PreviousAttribute" by an Attribute called "ReplacementAttribute" for an entire solution. The sample manually uses a SyntaxRewriter to exchange the attributes.

```
/// <summary>
/// The CSharpSyntaxRewriter allows to rewrite the Syntax of a node
/// </summary>
public class AttributeStatementChanger : CSharpSyntaxRewriter
{
    /// Visited for all AttributeListSyntax nodes
    /// The method replaces all PreviousAttribute attributes annotating a method by
ReplacementAttribute attributes
    public override SyntaxNode VisitAttributeList(AttributeListSyntax node)
    {
        // If the parent is a MethodDeclaration (= the attribute annotes a method)
        if (node.Parent is MethodDeclarationSyntax &&
            // and if the attribute name is PreviousAttribute
            node.Attributes.Any(
                currentAttribute => currentAttribute.Name.GetText().ToString() ==
"PreviousAttribute"))
        {
            // Return an alternate node that is injected instead of the current node
            return SyntaxFactory.AttributeList(
                            SyntaxFactory.SingletonSeparatedList(

SyntaxFactory.Attribute(SyntaxFactory.IdentifierName("ReplacementAttribute"),
                                SyntaxFactory.AttributeArgumentList(
                                    SyntaxFactory.SeparatedList(new[]
                                    {
                                    SyntaxFactory.AttributeArgument(
                                        SyntaxFactory.LiteralExpression(
                                            SyntaxKind.StringLiteralExpression,
SyntaxFactory.Literal(@"Sample"))
                                        )
                                    })))));
        }
        // Otherwise the node is left untouched
        return base.VisitAttributeList(node);
    }
}

/// The method calling the Syntax Rewriter
private static async Task<bool> ModifySolutionUsingSyntaxRewriter(string solutionPath)
{
    using (var workspace = MSBuildWorkspace.Create())
    {
        // Selects a Solution File
        var solution = await workspace.OpenSolutionAsync(solutionPath);
        // Iterates through every project
        foreach (var project in solution.Projects)
        {
            // Iterates through every file
            foreach (var document in project.Documents)
            {
                // Selects the syntax tree
                var syntaxTree = await document.GetSyntaxTreeAsync();
                var root = syntaxTree.GetRoot();

                // Generates the syntax rewriter
                var rewriter = new AttributeStatementChanger();
                root = rewriter.Visit(root);

                // Exchanges the document in the solution by the newly generated document
                solution = solution.WithDocumentSyntaxRoot(document.Id, root);
            }
        }
        // applies the changes to the solution
        var result = workspace.TryApplyChanges(solution);
        return result;
    }
}
```

The above example can be tested for the following class:

```
public class Program
{
    [PreviousAttribute()]
    static void Main(string[] args)
    {
    }
}
```

Read Change source code with Roslyn online: https://riptutorial.com/roslyn/topic/5221/change-source-code-with-roslyn

Chapter 4: Semantic Model

Introduction

In contrast to the Syntax API the exposes all kinds of syntax level information, the semantic model gives our code more "meaning" and allows us to answer questions like "What names are in scope at this location?", "What members are accessible from this method?", "What variables are used in this block of text?", "What does this name/expression refer to?".

Remarks

- Querying the Semantic Model is more costly than querying the Syntax Tree, due to the fact that it most commonly triggers a compilation.

Examples

Getting the Semantic Model

There qutie a fiew ways to get the sematic model.

- From a `Document` class

  ```
  Document document = ...;
  SemanticModel semanticModel = await document.GetSemanticModelAsync();
  ```

- From a `Compilation`class

  ```
  CSharpCompilation compilation = ...;
  var semanticModel = await compilation.GetSemanticModel(syntaxTree);
  ```

- From an `AnalysisContext`. Fro example inside a `DiagnosticAnalyzer` you can do:

  ```
  public override void Initialize(AnalysisContext context)
  {
      context.RegisterSemanticModelAction(x =>
      {
          var semanticModel = x.SemanticModel;
          // Do magical magic here.
      });
  }
  ```

Get all the references to a method

```
var syntaxRoot = await document.GetSyntaxRootAsync();

var semanticModel = await document.GetSemanticModelAsync();
```

```
var sampleMethodInvocation = syntaxRoot
    .DescendantNodes()
    .OfType<InvocationExpressionSyntax>()
    .First();

var sampleMethodSymbol = semanticModel.GetSymbolInfo(sampleMethodInvocation).Symbol;
var referencesToSampleMethod = await SymbolFinder.FindReferencesAsync(sampleMethodSymbol,
document.Project.Solution);
```

Read Semantic Model online: https://riptutorial.com/roslyn/topic/9772/semantic-model

Chapter 5: Syntax Tree

Introduction

One of the major parts of the Roslyn compiler is the Syntax API. It exposes the syntax trees the compilers use to understand Visual Basic and C# programs.

Remarks

- The Syntax Tree is a Parse Tree in the context of the Roslyn compiler.

Examples

Getting the Syntax Tree Root from a Document.

If you already have access to your `Document` class from your workspace (Using Workspaces) it is easy to access the root of your Syntax tree.

```
Document document = ... // Get document from workspace or other source

var syntaxRoot = await document.GetSyntaxRootAsync();
```

Traversing the Syntax Tree Using LINQ

You can easily navigate the a Syntax Tree using LINQ. For example it is easy to get all the `ClassDeclarationSyntax` nodes (declared classes), that have a name starting with the letter `A`:

```
var allClassesWithNameStartingWithA = syntaxRoot.DescendantNodes()
    .OfType<ClassDeclarationSyntax>()
    .Where(x => x.Identifier.ToString().StartsWith("A"));
```

Or getting all the classes that have attributes:

```
var allClassesWithAttriutes = syntaxRoot.DescendantNodes()
    .OfType<ClassDeclarationSyntax>()
    .Where(x => x.AttributeLists.Any(y => y.Attributes.Any()));
```

Traversing the Syntax Tree using a CSharpSyntaxWalker

The `CSharpSyntaxWalker` class is out of the box implementation of the Visitor pattern, that we can use to traverse our Syntax Tree. Here is a simple example of a Syntax Walker that collects all the `struct`-s that have a name, starting with the letter `A`:

```
public class StructCollector : CSharpSyntaxWalker
{
    public StructCollector()
    {
        this.Structs = new List<StructDeclarationSyntax>();
    }

    public IList<StructDeclarationSyntax> Structs { get; }

    public override void VisitStructDeclaration(StructDeclarationSyntax node)
    {
        if (node.Identifier.ToString().StartsWith("A"))
        {
            this.Structs.Add(node);
        }
    }
}
```

We can use our SyntaxWalker in the following way:

```
var structCollector = new StructCollector();
structCollector.Visit(syntaxRoot); // Or any other syntax node
Console.WriteLine($"The number of structs that have a name starting with the letter 'A' is
{structCollector.Structs.Count}");
```

Read Syntax Tree online: https://riptutorial.com/roslyn/topic/9765/syntax-tree

Chapter 6: Using Workspaces

Introduction

The workspace is a programmatic representation of the C# hierarchy that consists of a solution, child projects and child documents.

Remarks

- Currently there is no MSBuild workspace that supports a .NET Standard compliant projects. For more information see here.

Examples

Creating an AdhocWorkspace and adding a new project and a file to it.

The idea behind the `AdhocWorkspace` is to create a workspace on the fly.

```
var workspace = new AdhocWorkspace();

string projectName = "HelloWorldProject";
ProjectId projectId = ProjectId.CreateNewId();
VersionStamp versionStamp = VersionStamp.Create();
ProjectInfo helloWorldProject = ProjectInfo.Create(projectId, versionStamp, projectName,
projectName, LanguageNames.CSharp);
SourceText sourceText = SourceText.From("class Program { static void Main() {
System.Console.WriteLine(\"HelloWorld\"); } }");

Project newProject = workspace.AddProject(helloWorldProject);
Document newDocument = workspace.AddDocument(newProject.Id, "Program.cs", sourceText);
```

Creating an MSBuildWorspace, loading a solution and getting all the documents in all that solution

The `MSBuildWorspace` is built around the concept of handling MSBuild solutions (`.sln` files) and their respective projects (`.csproj`, `.vbproj`). Adding new projects and documents to this workspace is not supported.

```
string solutionPath = @"C:\Path\To\Solution\Sample.sln";

MSBuildWorkspace workspace = MSBuildWorkspace.Create();
Solution solution = await workspace.OpenSolutionAsync(nancyApp);

var allDocumentsInSolution = solution.Projects.SelectMany(x => x.Documents);
```

Getting the VisualStudioWorkspace from inside a Visual Studio Extension

In contrast to the other types of workspaces, the `VisualStudioWorkspace`, cannot be created manually. It can be accessed when building a Visual Studio extension.

When inside your extension package project, go to `[YourVSPackage]Package.cs` file. There you can acquire the workspace in two ways:

```
protected override void Initialize()
{
    // Additional code...

    var componentModel = (IComponentModel)this.GetService(typeof(SComponentModel));
    var workspace = componentModel.GetService<VisualStudioWorkspace>();
}
```

Or by using MEF:

```
[Import(typeof(VisualStudioWorkspace))]
public VisualStudioWorkspace ImportedWorkspace { get; set; }
```

A great video tutorial about the `VisualStudioWorkspace`, can be found here.

Read Using Workspaces online: https://riptutorial.com/roslyn/topic/9755/using-workspaces

Credits

S. No	Chapters	Contributors
1	Getting started with roslyn	Community, Teodor Kurtev
2	Analyze source code with Roslyn	andyp, Michael Rätzel, SJP, Stefano d'Antonio
3	Change source code with Roslyn	SJP, Teodor Kurtev
4	Semantic Model	Teodor Kurtev
5	Syntax Tree	Teodor Kurtev
6	Using Workspaces	Teodor Kurtev