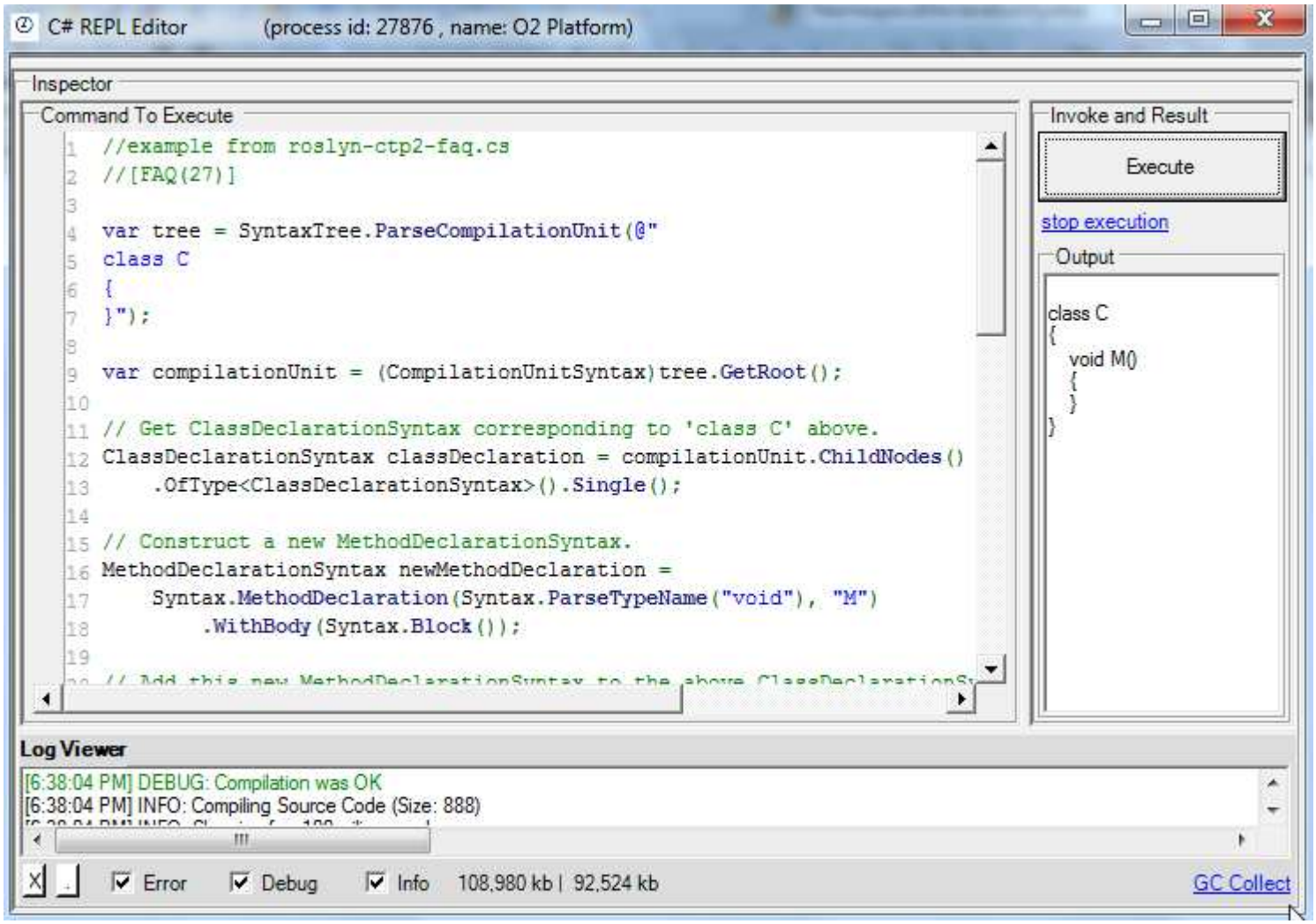
Refactoring AddingMethodToClass

Let's start by executing example 27 (called *AddMethodToClass*)

```
//code from roslyn-ctp2-faq.cs
//     that can be downloaded from http://www.codeplex.com/Download?ProjectName=dlr&DownloadId=386317)
//[FAQ(27)]

var tree = SyntaxTree.ParseCompilationUnit(@"
class C
{
}");

var compilationUnit = (CompilationUnitSyntax)tree.GetRoot();

// Get ClassDeclarationSyntax corresponding to 'class C' above.
ClassDeclarationSyntax classDeclaration = compilationUnit.ChildNodes()
    .OfType<ClassDeclarationSyntax>().Single();

// Construct a new MethodDeclarationSyntax.
MethodDeclarationSyntax newMethodDeclaration =
    Syntax.MethodDeclaration(Syntax.ParseTypeName("void"), "M")
        .WithBody(Syntax.Block());

// Add this new MethodDeclarationSyntax to the above ClassDeclarationSyntax.
ClassDeclarationSyntax newClassDeclaration =
    classDeclaration.AddMembers(newMethodDeclaration);

// Update the CompilationUnitSyntax with the new ClassDeclarationSyntax.
CompilationUnitSyntax newCompilationUnit =
    compilationUnit.ReplaceNode(classDeclaration, newClassDeclaration);

// Format the new CompilationUnitSyntax.
newCompilationUnit = (CompilationUnitSyntax)newCompilationUnit.Format().GetFormattedRoot();
return newCompilationUnit.GetFullText();

//using Roslyn.Compilers;
//using Roslyn.Compilers.Common;
//using Roslyn.Compilers.CSharp;
//using Roslyn.Scripting;
//using Roslyn.Scripting.CSharp;
//using Roslyn.Services;

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2Ref:Roslyn.Services.dll
//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll
```



The objective is to refactor the code using O2's Roslyn APIs (and extension methods)

starting with getting the AstTree we can replace the exiting code with an Extention Method (in comments is the code replaced)

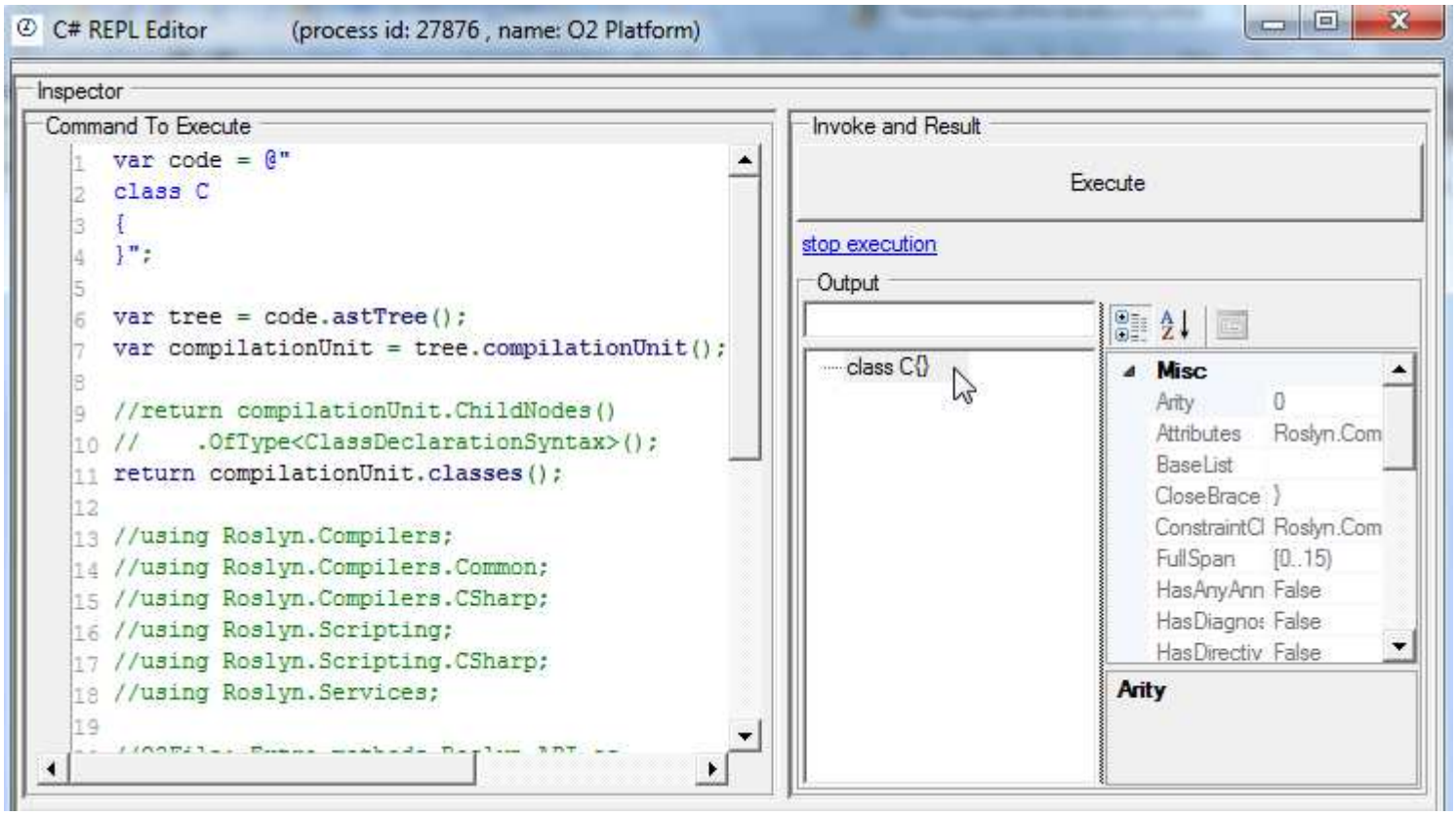
```
/*var tree = SyntaxTree.ParseCompilationUnit(@"
class C
{
}");
*/

var code = @"
class C
{
}";


var tree = code.astTree();
```

then the Compilation unit (in comments is the code replaced)

```
var tree = code.astTree();
//var compilationUnit = (CompilationUnitSyntax)tree.GetRoot();
var compilationUnit = tree.compilationUnit();
```

getting all classes in compilation Unit (in comments is the code replaced)

```
var tree = code.astTree();
var compilationUnit = tree.compilationUnit();

//return compilationUnit.ChildNodes()
//    .OfType<ClassDeclarationSyntax>();
return compilationUnit.classes();
```



getting the first class declaration (in comments is the code replaced)

```
var tree = code.astTree();
var compilationUnit = tree.compilationUnit();

//ClassDeclarationSyntax classDeclaration = compilationUnit.ChildNodes()
//    .OfType<ClassDeclarationSyntax>().Single();
var classDeclaration = compilationUnit.classes().first();

return classDeclaration;
```

now lets simplify the method creation, which currently looks like this

```
MethodDeclarationSyntax newMethodDeclaration =
    Syntax.MethodDeclaration(Syntax.ParseTypeName("void"), "M")
        .WithBody(Syntax.Block());
```

create TypeSyntax using a string extension method

```
//var name = Syntax.ParseTypeName("void")
var name = "void".parse_TypeName();

MethodDeclarationSyntax newMethodDeclaration =
    Syntax.MethodDeclaration(name, "M")
        .WithBody(Syntax.Block());
```

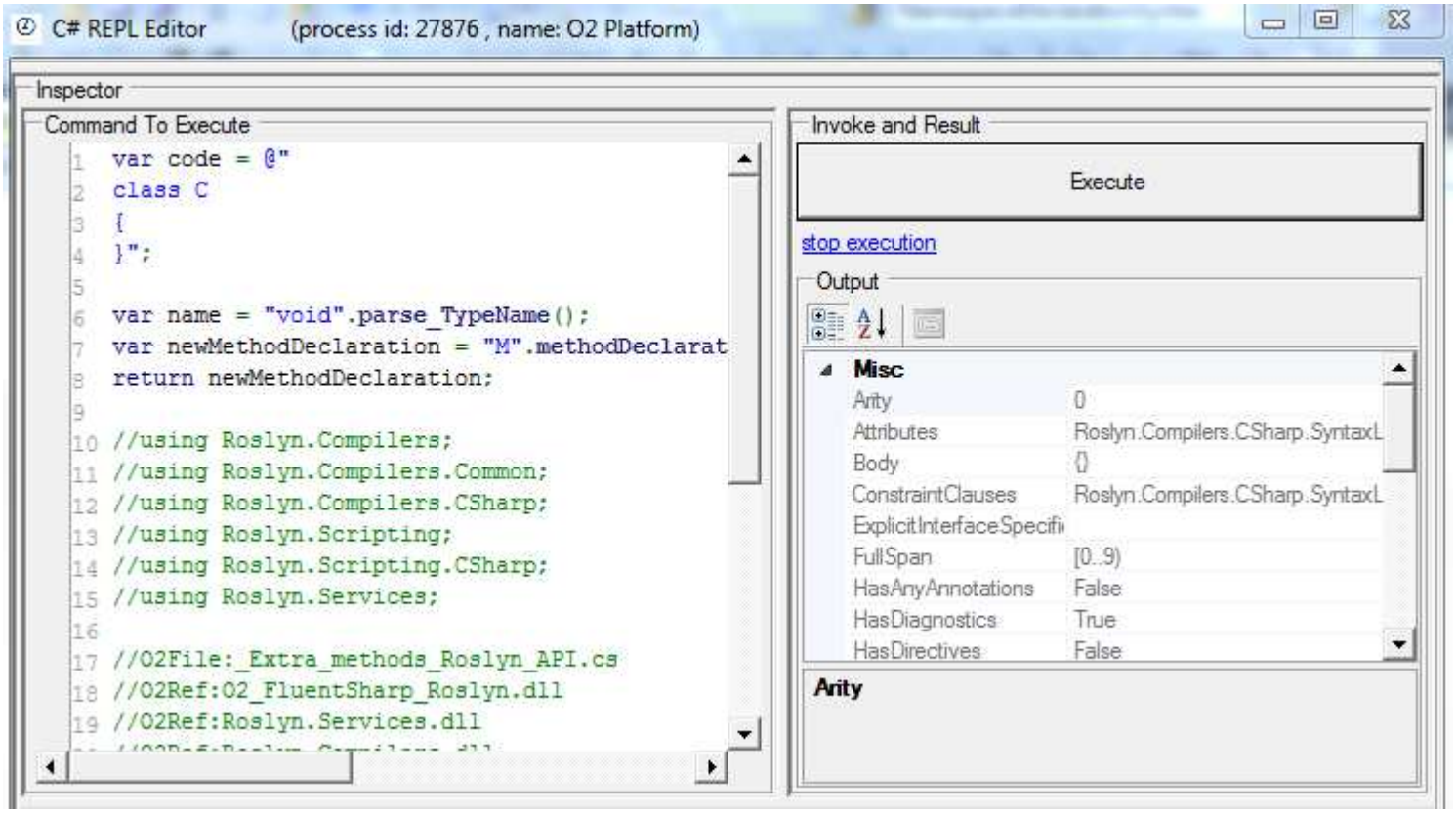
creating the method declaration from a string and return value

```
var name = "void".parse_TypeName();


var newMethodDeclaration = "M".methodDeclaration(name).WithBody(Syntax.Block());
```

```
return newMethodDeclaration;
```

adding a body

```
var name = "void".parse_TypeName();
var newMethodDeclaration = "M".methodDeclaration(name).add_Body();
return newMethodDeclaration;
```



we can do the *parse_TypeName* call inside the methodDeclaration extensionMethod

```
var newMethodDeclaration = "M".methodDeclaration("void").add_Body();
return newMethodDeclaration;
```

we can assume that when return value is specified it is void

```
var newMethodDeclaration = "M".methodDeclaration().add_Body();
return newMethodDeclaration;
```

we can also assume that by default methods have a body

```
var newMethodDeclaration = "M".methodDeclaration();

return newMethodDeclaration.formatedCode();    //creates:   void M()
                                                            {
                                                            }
```

here are number of examples of how the methodDeclaration extensionMethod can be used:

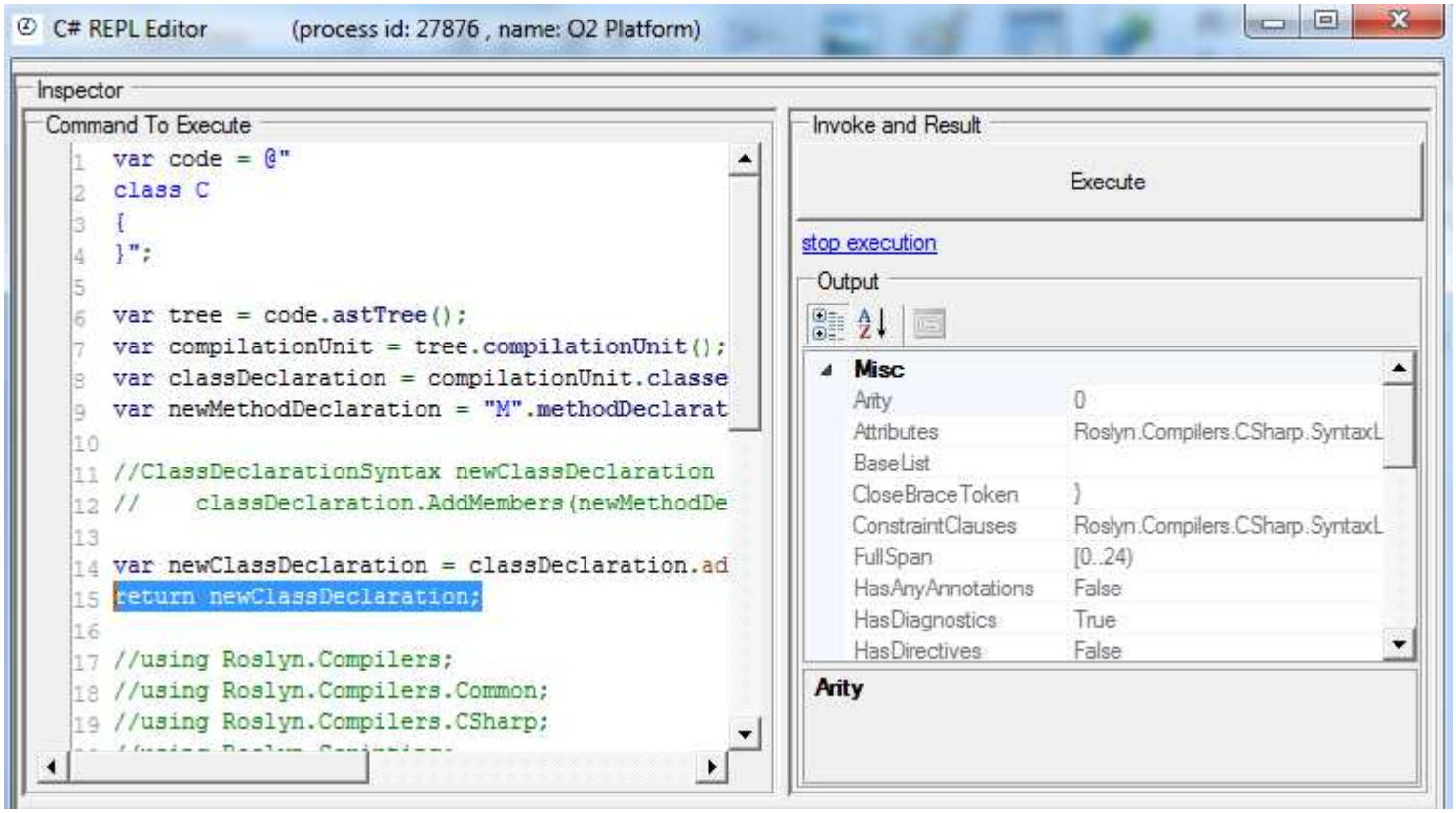
```
"M".methodDeclaration("string").formatedCode();                    //creates:  string M() {}
"M".methodDeclaration("void").formatedCode();                      //creates:  void M() {}
"M".methodDeclaration().formatedCode();                            //creates:  void M() {}
"GiveMeAnInt".methodDeclaration("int").formatedCode();             //creates:  int32 GiveMeAnInt() {}
"WithNoMethodBody".methodDeclaration("String",false).formatedCode();  //creates: String WithNoMethodBody
()
```

back to the main script we're refactoring (in comments is the code replaced)

```
var tree = code.astTree();
var compilationUnit = tree.compilationUnit();
var classDeclaration = compilationUnit.classes().first();

/*MethodDeclarationSyntax newMethodDeclaration =
    Syntax.MethodDeclaration(Syntax.ParseTypeName("void"), "M")
        .WithBody(Syntax.Block());*/

var newMethodDeclaration = "M".methodDeclaration();
```

Adding a method to a class (in comments is the code replaced):

```
var tree = code.astTree();
var compilationUnit = tree.compilationUnit();
var classDeclaration = compilationUnit.classes().first();
var newMethodDeclaration = "M".methodDeclaration();

//ClassDeclarationSyntax newClassDeclaration =
//    classDeclaration.AddMembers(newMethodDeclaration);

var newClassDeclaration = classDeclaration.add(newMethodDeclaration);
return newClassDeclaration;
```



this can be further simplified by an *add_Method* which receives as parameter the method name (and optionally the return type)

```
var newClassDeclaration = classDeclaration.add_Method("M").add_Method("B", "string");
return newClassDeclaration.formatedCode();
/*the formated code will be:

class C
{
    void M()
    {
    }

    string B()
    {
    }
}
*/
```

replacing the ClassDeclarations (in comments is the code replaced)

```
// Update the CompilationUnitSyntax with the new ClassDeclarationSyntax.
//CompilationUnitSyntax newCompilationUnit =
//    compilationUnit.ReplaceNode(classDeclaration, newClassDeclaration);

var newCompilationUnit = compilationUnit.replace(classDeclaration, newClassDeclaration);
```

getting the compilationUnit from one of the Syntax nodes

```
return classDeclaration.parent<CompilationUnitSyntax>();
return classDeclaration.compilationUnit();
return newClassDeclaration.compilationUnit();

return newCompilationUnit.classes()
                         .first()
                         .compilationUnit();
return newCompilationUnit.classes().first()
                         .methods().first()
                         .compilationUnit();
```

since we can get the compilation Unit from the class to replace, we can replace a class like this:
(in comments is the code replaced)

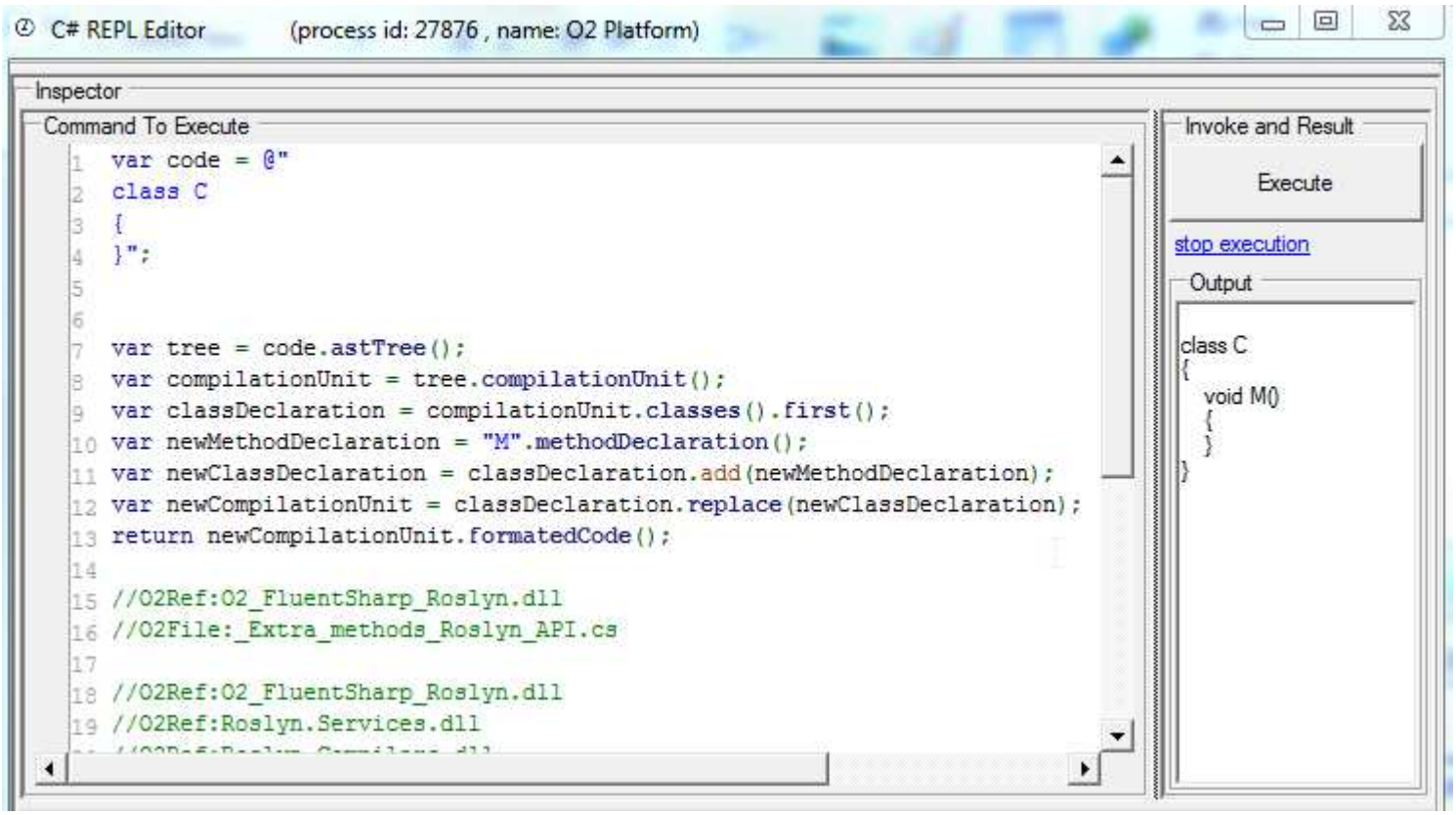
```
//CompilationUnitSyntax newCompilationUnit =
//    compilationUnit.ReplaceNode(classDeclaration, newClassDeclaration);

var newCompilationUnit = classDeclaration.replace(newClassDeclaration);
```

get the formatedCode (in comments is the code replaced)

```
//newCompilationUnit = (CompilationUnitSyntax)newCompilationUnit.Format().GetFormattedRoot();
//return newCompilationUnit.GetFullText();
return newCompilationUnit.formatedCode();
```

here is the original code using the ExtensionMethods (from O2's Rosyln API):

```
var code = @"
class C
{
}";


var tree = code.astTree();
var compilationUnit = tree.compilationUnit();
var classDeclaration = compilationUnit.classes().first();
var newMethodDeclaration = "M".methodDeclaration();
var newClassDeclaration = classDeclaration.add(newMethodDeclaration);
var newCompilationUnit = classDeclaration.replace(newClassDeclaration);
return newCompilationUnit.formatedCode();

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2File:_Extra_methods_Roslyn_API.cs

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2Ref:Roslyn.Services.dll
//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll
```



which can be further simplified to:

```
var code = @"
class C
{
}";


var classDeclaration =  code.astTree()
                            .compilationUnit()
                           .classes().first();
var newMethodDeclaration = "M".methodDeclaration();
var newClassDeclaration = classDeclaration.add(newMethodDeclaration);
var newCompilationUnit = classDeclaration.replace(newClassDeclaration);
return newCompilationUnit.formatedCode();
```

and to

```
var code = @"
class C
{
}";


var classDeclaration =  code.astTree()
                            .compilationUnit()
                            .classes().first();
var newClassDeclaration = classDeclaration.add("M".methodDeclaration());
var newCompilationUnit = classDeclaration.replace(newClassDeclaration);
return newCompilationUnit.formatedCode();
```

and to:

```
var code = @"
class C
{
}";


var classDeclaration =  code.astTree()
                            .compilationUnit()
                            .classes().first();
var newCompilationUnit = classDeclaration.replace(classDeclaration.add("M".methodDeclaration()));
```

```
return newCompilationUnit.formatedCode();
```

and to:

```
var code = @"
class C
{
}";


var classDeclaration =  code.astTree()
                            .compilationUnit()
                            .classes().first();

classDeclaration.replace(classDeclaration.add("M".methodDeclaration()))
                .formatedCode();
```

Sometimes we can refactor too much :) This is probably easier to read:

```
var code = @"class C {}";

var classDeclaration =  code.astTree()
                            .compilationUnit()
                            .classes().first();

var newClassDeclaration = classDeclaration.add("M".methodDeclaration());

return classDeclaration.replaceWith(newClassDeclaration).formatedCode();

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2File:_Extra_methods_Roslyn_API.cs

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll
```

for reference here is the original code (using Roslyn APIs)

```
var tree = SyntaxTree.ParseCompilationUnit(@"
class C
{
}");
```

```
var compilationUnit = (CompilationUnitSyntax)tree.GetRoot();

// Get ClassDeclarationSyntax corresponding to 'class C' above.
ClassDeclarationSyntax classDeclaration = compilationUnit.ChildNodes()
    .OfType<ClassDeclarationSyntax>().Single();

// Construct a new MethodDeclarationSyntax.
MethodDeclarationSyntax newMethodDeclaration =
    Syntax.MethodDeclaration(Syntax.ParseTypeName("void"), "M")
        .WithBody(Syntax.Block());

// Add this new MethodDeclarationSyntax to the above ClassDeclarationSyntax.
ClassDeclarationSyntax newClassDeclaration =
    classDeclaration.AddMembers(newMethodDeclaration);

// Update the CompilationUnitSyntax with the new ClassDeclarationSyntax.
CompilationUnitSyntax newCompilationUnit =
    compilationUnit.ReplaceNode(classDeclaration, newClassDeclaration);

// Format the new CompilationUnitSyntax.
newCompilationUnit = (CompilationUnitSyntax)newCompilationUnit.Format().GetFormattedRoot();
return newCompilationUnit.GetFullText();

//using Roslyn.Compilers;
//using Roslyn.Compilers.Common;
//using Roslyn.Compilers.CSharp;
//using Roslyn.Scripting;
//using Roslyn.Scripting.CSharp;
//using Roslyn.Services;

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2Ref:Roslyn.Services.dll
//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll
```

and the Code using O2's Roslyn API (with the same comments)

```
var code = @"
class C
{
}";

// Get ClassDeclarationSyntax corresponding to 'class C' above.
var classDeclaration =  code.astTree()
                            .compilationUnit()
                            .classes().first();

// Construct a new MethodDeclarationSyntax.
// Add this new MethodDeclarationSyntax to the above ClassDeclarationSyntax.

var newClassDeclaration = classDeclaration.add("M".methodDeclaration());

// Update the CompilationUnitSyntax with the new ClassDeclarationSyntax.
 // Format the new CompilationUnitSyntax.
return classDeclaration.replaceWith(newClassDeclaration).formatedCode();

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2File:_Extra_methods_Roslyn_API.cs

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2Ref:Roslyn.Services.dll
//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll
```

ExtensionMethods (most) added during the creation of this script:

```
// This file is part of the OWASP O2 Platform (http://www.owasp.org/index.php/OWASP_O2_Platform) and is
```

```
released under the Apache 2.0 License (http://www.apache.org/licenses/LICENSE-2.0)
using System;
using System.Linq;
using System.Collections.Generic;
using O2.Kernel.ExtensionMethods;
using O2.DotNetWrappers.ExtensionMethods;
using Roslyn.Compilers.Common;
using Roslyn.Compilers.CSharp;
using Roslyn.Compilers;

//O2Ref:O2_FluentSharp_Roslyn.dll
//O2Ref:Roslyn.Compilers.dll
//O2Ref:Roslyn.Compilers.CSharp.dll

namespace O2.XRules.Database.APIs
{
	public static class _Extra_methods_Roslyn_API
	{
		public static SyntaxTree astTree(this string code)
		{
			return code.tree();
		}

		public static bool hasErrors(this SyntaxTree tree)
		{
			return tree.errors().size() > 0;
		}


		public static bool hasErrors(this CommonCompilation compilation)
		{
			return compilation.errors().size() > 0;
		}

		public static Compilation compiler(this SyntaxTree tree)
		{
			return tree.compiler(7.randomLetters());
		}

		public static CompilationUnitSyntax compilationUnit(this SyntaxTree tree)
		{
			return (CompilationUnitSyntax)tree.GetRoot();
		}

		public static CompilationUnitSyntax compilationUnit(this SyntaxNode syntaxNode)
		{
			return syntaxNode.parent<CompilationUnitSyntax>();
		}


		public static CompilationUnitSyntax replace(this CompilationUnitSyntax compilationUnit,
ClassDeclarationSyntax classA, ClassDeclarationSyntax classB)
		{
			return compilationUnit.ReplaceNode(classA, classB);
		}

		public static CompilationUnitSyntax replace(this ClassDeclarationSyntax classA,
ClassDeclarationSyntax classB)
		{
			return classA.compilationUnit().ReplaceNode(classA, classB);
		}

		public static CompilationUnitSyntax replaceWith(this ClassDeclarationSyntax classA,
ClassDeclarationSyntax classB)
		{
			return classA.replace(classB);
		}


		public static List<ClassDeclarationSyntax> classes(this CompilationUnitSyntax
compilationUnit)
		{
			return compilationUnit.ChildNodes()
                                      .OfType<ClassDeclarationSyntax>().toList();
		}

		public static List<MethodDeclarationSyntax> methods(this ClassDeclarationSyntax
classDeclarationSyntax)
		{
			return classDeclarationSyntax.ChildNodes()
```

```
                                                .OfType<MethodDeclarationSyntax>().toList
();
            }

            public static TypeSyntax parse_TypeName(this string name)
            {
                  return Syntax.ParseTypeName(name ?? "void");
            }


            public static MethodDeclarationSyntax methodDeclaration(this string methodName, string
returnType = null, bool addEmptyBody = true)
            {
                  return methodName.methodDeclaration(returnType.parse_TypeName(), addEmptyBody);
            }

            public static MethodDeclarationSyntax methodDeclaration(this string methodName, TypeSyntax
returnType, bool addEmptyBody = true)
            {
                  var methodDeclaration = Syntax.MethodDeclaration(returnType, methodName);
                  if (addEmptyBody)
                        return methodDeclaration.add_Body();
                  return methodDeclaration;
            }

            public static MethodDeclarationSyntax add_Body(this MethodDeclarationSyntax
methodDeclaration)
            {
                  return methodDeclaration.WithBody(Syntax.Block());
            }

            public static ClassDeclarationSyntax add(this ClassDeclarationSyntax classDeclaration,
MethodDeclarationSyntax methodDeclaration)
            {
                  return classDeclaration.AddMembers(methodDeclaration);
            }

            public static ClassDeclarationSyntax add_Method(this ClassDeclarationSyntax
classDeclaration, string methodName, string returnType = null)
            {
                  return classDeclaration.AddMembers(methodName.methodDeclaration(returnType));

            }

            public static T parent<T>(this SyntaxNode syntaxNode)
                  where T : SyntaxNode
            {
                  if (syntaxNode.notNull())
                  {
                        if (syntaxNode.Parent is T)
                              return (T)syntaxNode.Parent;
                        return syntaxNode.Parent.parent<T>();
                  }
                  return default(T);
            }
      }

 }
```