RoslynDom Update 1.0.10-alpha

After reviewing the changes in 1.0.10-alpha I think trying to expand on the features in the context of the update document is not realistic. The document has some minor updates in the documents folder, and I’m archiving the update document for Updates 1.0.1-alpha to 1.0.10-alpha and starting a new document for 1.0.11-alpha. I’ll highlight the changes here in the context of goals accomplished. This has been an enormous leap and I now know where the last five or six weeks of my life have gone.

# Language independence

There are three vertical slices to the overall RoslynDom.

* The interfaces, which I’ll discuss separately with the unrealistically broad goal of being somewhat platform independent and supplying feature based access to the RoslynDom silo. To the extent you can, ignore the interfaces until I discuss them further as they might give you a headache.
* RoslynDom itself, which is a language independent representation of your source code that is designed from the perspective of .NET and the .NET Compiler Platform, Roslyn. There is significant divergence from the .NET Compiler Platform when it aided the goals
  + Mutability
  + Layout independent format
  + Language independent format
  + Simple access
  + SameIntent support
  + Support for comments and XML documentation as first class citizens
  + Support for compiler directives as first class citizens
* RoslynDomCSharpFactories, C# factories to load RoslynDom and recreate a .NET Compiler Platform, Roslyn, SyntaxTree. Loading and unloading is via the SyntaxTree to allow parsing and consistent structures.

Each language element, such as a method, is represented by a composed interface (IMethod), a language independent RoslynDom class (RDomMethod) and a C# specific factory (RDomMethodTypeMemberFactory). Each factory has CreateFrom… and BuildSyntax… methods. CreateFrom… methods create RoslynDom entities and BuildSyntax… methods recreate syntax elements for the SyntaxTree.

# Dependency Injection

The groundwork for extensibility is in the dependency injection approach to retrieving factories. Since factories instantiate RoslynDom entities and recreate syntax, they are the key player in modifications and extensions. Since they interact to build RoslynDom and SyntaxTree entities, their retrieval is crucial to an extensibility story.

One known extensibility story is a Visual Basic factory. It seems likely possible that user stories for building RoslynDom trees from scratch as an easier way to build SyntaxTrees from scratch will want to tie into extensibility, and therefore a raw helper factory might make sense. I am hoping that more ambitious factories like VB6 can also be created.

There has not yet been any testing of extensibility and more work is required in refactoring the BuildSyntaxHelper methods.

# Comments, Vertical Whitespace and XML Documentation Comments

The .NET Compiler Platform, Roslyn, places all whitespace and all comments into language trivia attached to the first token following where the trivia should appear.

Of necessity I use comments for “public annotations” (in earlier versions) which provide information for RoslynDom clients. This is entirely separate from the private annotations the .NET Compiler Platform, Roslyn provides. Also, for any use you will have, XML Documentation (also called Structured Documentation because the use of XML is compiler dependent) should be available on the language element (such as the class or method) it belongs to. Similar arguments raise directives to being first class citizens.

There are four levels where vertical whitespace and comments can logically occur: file, stem, type and code (method or property). Each of these has a MembersAll property that include comments and vertical whitespace, as well as appropriate code elements. The Members property includes all code elements except comments and vertical whitespace.

Structured documentation is extracted and placed on the corresponding element. At present, you access the XML because breaking this into a true structure is a lower priority because I don’t have user stories.

# Horizontal Whitespace

Earlier versions of RoslynDom were very heavy handed in formatting. This version manages horizontal whitespace. About 25-30% of the code in the factories is now dedicated to managing horizontal whitespace. Three weeks, three redesigns, and a few tears went into this, but the current approach appears solid.

# Report Hierarchy and ToString()

A ReportHierarchy method allows better information about the RoslynDom tree, particularly in the immediate window. More work will go into this, so do not take a dependency on the current structure you don’t want broken.

# Added Statement support

There are six main levels of elements in your code base: file, stem, types, code container, statement and expressions. Previous versions of RoslynDom supported only file, stem, types and code containers. This version supports a variety of statements.

Statements are logically nested in code blocks, particularly the code blocks of conditional (if) and looping statements.

Expressions are minimally supported – RoslynDom uses conditions and assignment expressions without breaking them down or understanding them. I’m not sure whether it ever will. I have compelling user stories for understanding statements and statement parts (see the walkthroughs for one example). I do not yet have compelling user stories for breaking down expressions. If the only user story is intelligent VB.NET/C# conversions, support may be minimal.

# Added Ancestors and Descendants

You can now query the ancestors and descendants of RoslynDom trees. See the walkthroughs for an example of why you might find this interesting.

# Interfaces made non-immutable (mutable)

The RoslynDom tree is mutable. This is because of my intended usage and because I think one of the things an alternative to the .NET Compiler Platform, Roslyn is a mutable alternative. I absolutely agree that the .NET Compiler Platform, Roslyn should be immutable – it’s a compiler structure. However, this pretty much forces a rewriter for any non-trivial changes to the SyntaxTree. I believe there will be scenarios where it’s much easier to load into RoslynDom, do interrogation and mutating in that structure, then output to a new .NET Compiler Platform, Roslyn SyntaxTree.

In my initial vision, the interfaces were immutable (IMethod) and the RoslynDom implementation was mutable (RDomMethod). This proved impractical because of excess casting for mutations. My new vision is that if there’s a need for an immutable set of interfaces, the current set will inherit from the immutable set.

As an implication, and allowing for errors, if something is not-mutable in the interfaces, such as the RDomLists, they aren’t supposed to be changed.

# There’s still a lot of work to do

GitHub lists known issues. The next version or two will be clean up and documentation improvements.

Following that I’ll plug the holes of the most important language features I’m not yet supporting. These include regions, lambdas and async because they are hard, and side cases like destructors and explicit operators.

I want to solidify a single file before I work across multiple files. Multiple file usage will make the underlying model much more useful and allow more interesting interrogation of non-mutated RoslynDom structures.

The biggest help I need right now are user stories, even vague ones, and failing unit tests – particularly if it crashes RoslynDom. Of course, if you’d like to help further please be in touch. If you want to fork the code, it would be lovely to see what you’re doing.

This is still a very early release. Everything is up for change.

I will try to keep the NuGet release from getting as out of date as it has been for the last month.

# 