Meditation on Adding INotifyPropertyChanged to RoslynDom

I was faced with the dual realities of making a broad change to RoslynDom and the fact RoslyDom is supposed to empower exactly this kind of change.

So, obviously, it was time to go off the high dive.

The immediate change: implement a change mechanism, probably INotifyPropertyChanged, but can’t this be flexible?

The big change that was needed along the way: I overlooked how badly RoslynDom supported creating RoslynDom structures from scratch. Wow, that is a major scenario, time to fix it.

# Step 1: Create the test to load the files and make the change

Cruising the directories and loading the files was pretty easy. If you think I was putting the altered changes on top of the originals – I love GitHub, but *are you crazy?* I built into a new directory, checked and copied. Here’s the code for managing files. :

private string inputDirectory = @"..\..\..\RoslynDom";

private string outputDirectory = @"..\..\Walkthrough2\_Updated\_Files";

[TestMethod]

public void Walkthroughs\_2\_load\_files()

{

UpdateFilesInDirectory(inputDirectory, outputDirectory, @"BasesAndBaseHelpers");

UpdateFilesInDirectory(inputDirectory, outputDirectory, @"Implementations");

UpdateFilesInDirectory(inputDirectory, outputDirectory, @"StatementImplementations");

// UpdateFilesInDirectory(inputDirectory, outputDirectory, "");

}

private void UpdateFilesInDirectory(string inputDirectory, string outputDirectory, string subDirectory)

{

var factory = RDomCSharp.Factory;

var inputDir = Path.Combine(inputDirectory, subDirectory);

var outputDir = Path.Combine(outputDirectory, subDirectory);

var files = Directory.GetFiles(inputDir, "\*.cs");

foreach (var fileName in files)

{

UpdateFile(factory, outputDir, fileName);

}

}

OK, now what should that UpdateFile method do?

# Step 2: Alter the base class

I really don’t want to commit to INotifyPropertyChanged. It works, but it sometimes feels so creaky to use events for the simplest notification. So, I decided to isolate notification into the base class. Honestly, this isn’t enough and I’ll be back to work here. This approach doesn’t offer you a simple way to insert a different notification. But I want to postpone that problem, and just implement INotifyPropertyChanged in an isolated manner. Here it is:

public abstract class RDomBase : IDom, INotifyPropertyChanged

{

// Lots of other stuff

protected void SetProperty<T>(ref T field, T value, [CallerMemberName] string name = "")

{

if (!EqualityComparer<T>.Default.Equals(field, value))

{

field = value;

// TODO: Update to null conditionals in C# 6

var handler = PropertyChanged;

if (handler != null)

{

handler(this, new PropertyChangedEventArgs(name));

}

}

}

}

By putting this in the base class, the derived classes need very little change. This code replaces an auto-property for all editable properties:

private string \_name;

public string Name

{

get { return \_name; }

set { SetProperty(ref \_name, value); }

}

# Step 3: Alter the code

The first step is to identify the properties that are candidates for change:

private void AddINotifyPropertyChanged(IClass cl)

{

var notifyingProps = cl.Properties

.Where(x => x.CanSet

&& !x.SetAccessor.Statements.Any()

&& !x.GetAccessor.Statements.Any()

&& x.AccessModifier == AccessModifier.Public);

foreach (var prop in notifyingProps)

{

UpdateProperty(prop);

}

}

Updating each one is very easy with RoslynDom:

private void UpdateProperty(IProperty prop)

{

// Add the field without further checks because the programmer will find and resolve

// things like naming collisions

var parent = prop.Parent as ITypeMemberContainer;

var fieldName = StringUtilities.CamelCase(prop.Name);

var field = new RDomField(fieldName, prop.ReturnType, declaredAccessModifier: AccessModifier.Private);

FixWhitespace(field, prop);

field.Whitespace2Set.Add(new Whitespace2(prop.Whitespace2Set.First().Copy()));

parent.MembersAll.InsertOrMoveBefore(prop, field);

UpdatePropertyGet(prop, fieldName);

UpdatePropertySet(prop, fieldName);

}

For each property, I create a field for the backing field, fix the whitespace, and alter the get and set properties. RoslynDom still needs another generation of whitespace management – you should not need to do any whitespace changes for a change this small – but currently you do:

private void FixWhitespace(RDomField field, IProperty prop)

{

// TODO: This is rather detailed because of featuresnot yet in the whitespace system

var leading = prop.Whitespace2Set[LanguageElement.Public].LeadingWhitespace;

field.Whitespace2Set[LanguageElement.Private] = new Whitespace2(LanguageElement.Private, leading, " ", "");

}

So far, nothing is too hard, but what about changing the code? Is that hard?

The property get needs to create a return statement, which needs an expression. RoslynDom currently uses The .NET Compiler Platform, Roslyn for expressions, because it’s expression parsing is so powerful:

private void UpdatePropertyGet(IProperty prop, string fieldName)

{

var retExpression = RDomCSharp.Factory.ParseExpression(fieldName);

var statement = new RDomReturnStatement(retExpression, true);

prop.GetAccessor.StatementsAll.AddOrMove(statement);

prop.GetAccessor.EnsureNewLineAfter();

}

The property set requires invoking the base class method. Again, created from a string representation of the expression.

private void UpdatePropertySet(IProperty prop, string fieldName)

{

var expression = RDomCSharp.Factory.ParseExpression(string.Format("SetProperty(ref {0}, value)", fieldName));

var statement = new RDomInvocationStatement(expression, true);

prop.SetAccessor.StatementsAll.AddOrMove(statement);

prop.GetAccessor.EnsureNewLineAfter();

}

That’s it. The entire code to implement INotifyPropertyChanged. I will continue to work on making it easier, particularly with whitespace, but that’s not too bad.

# But it’s ugly!

Right, at the moment, when you load the code you should reformat with Ctl-K, D. Just wanted you to remember me via my initials.

# But this didn’t even come close to working in the previous version!

That’s true. All of the RoslynDom classes needed updating to add a constructor that took the editable fields.

# Manage loops