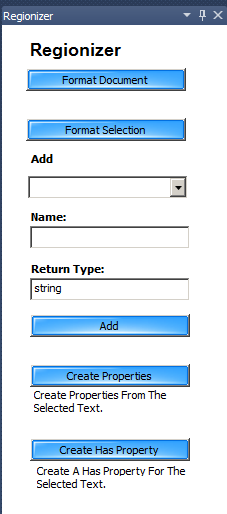
**Regionizer Users Guide**

**Introduction**

****

Screenshot from Regionizer Tool Window docked inside of Visual Studio 2010, similar to where the Solution Explorer, Toolbox or the Properties Window is located.

**What is Regionizer?**

**Regionizer** is a Package for Microsoft Visual Studio 2010 that has many features; this document will introduce you to **Regionizer** and show you how to use the many features that **Regionizer** includes.

****

**Warning! Regionizer** modifies your code file, and while Regionizer has been tested on many types of C# code files and should be safe to use in most cases, you should use Regionizer at your own Risk and verify your project compiles before and after using any of the Regionizer tools.

Remember you can always use Edit Undo until your code is returned to the state before using any of the Regionizer Tools. We cannot be responsible for damage to your project if any code is lost or modified. It is always a good idea to make backups of your code before using Regionizer and we strongly recommend that you abide by this warning.

**C# Document Formatter**



The **C# Document Formatter** formats a C# code so that no matter which member of your team created the file your code will be arraigned consistently within your organization. This adds value to your code especially for companies who need to share their code with customers; presenting very consistent code adds value to your source code and makes your company look much more professional.

The C# Document Formatter formats a C# code file into regions for:

1. Using Statements
2. Private Variables
3. Constructors
4. Events
5. Methods
6. Properties

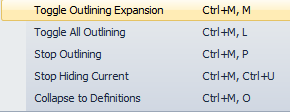
The events, methods and properties are sorted alphabetically, this makes finding code much more consistent and easier to maintain.

The **C# Document Formatter** is the cure for ***spaghetti code.*** Many programmers coding styles are very sloppy. I have had managers and bosses that are brilliant programmers, but they spend zero time on formatting and arranging code. They will create a private variable here, a method next, and event next a property next and then another method, without any concern as to how the file is arranged. When the culprit is your boss there is not much you can say, until now. **Regionizer** will transform even the sloppiest code into a very neat, well organized and easy to read document.

**Tip:**

**Regionizer** is for Regionaholics, so hopefully you will enjoy the fact that methods, properties and events are surrounded by regions that can be collapsed / expanded as needed.

If you have team members that do not like code collapsed into regions, Visual Studio provides a method for expanding all regions; under the Edit menu select Edit > Outlining > Toggle All Outlining.



**Format Selection**



The **Format Selection Button** will cut the selected text and paste it into the correct location inside the appropriate region for ***Events, Methods or Properties***.

The typical use for **Format Selection** is as follows:

Your document has been formatted but you have used one of the refactoring tools of Visual Studio such as Generate > Method Stub. This method will determine the type of code you have selected and cut and paste the code into the alphabetically sorted location. Alternatively you could use the Format Document button, but this takes longer on large code files.

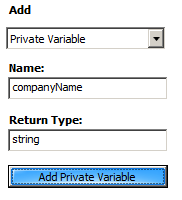
**Remember the Format Selection button only works on Events, Methods or Properties.**

I did not spend any time making **Regionizer** “Idiot Proof”; this is a tool meant to be used by professional developers and you should be smart enough to know when to click which buttons. If you click a button when a C# code file is not the Active Document, you will get errors or unexpected results so use this tool with care as the tool expects you to be able to determine the appropriate state of Visual Studio before you click any buttons.

There are some tips on how to cut the text that you will pick up quiet easily; if you select a blank line or two with the text that needs formatting, **Regionizer** will erase the unneeded line thus saving you from having to delete the extra line after the text cut and paste. With practice you will learn what to select to save the time of the extra line deletion as it is annoying to do over and over.

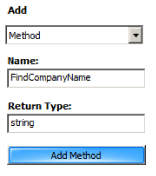
**Insert Code Elements**

**Add Private Variable**



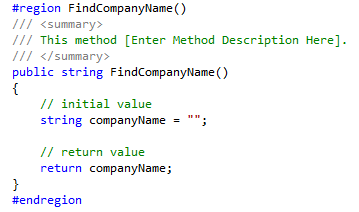
This feature was built as a test, but it does actually work so I left it in. This can be useful for creating multiple private variables of the same Return Type as you only have to change the name for each private variable. It is probably just as fast to just type this private variable yourself, but use it if you desire.

**Add Event or Method**

****

Events or Methods are essentially treated the same, the only difference as events are identified by certain items of text such as the presence of **object sender** or **EventArgs e**; it is possible for a method to get labeled as an event if you have a method named GetBigEventDate(); I have just learned to leave the method in the Events Region as it does not hurt anything to be there.

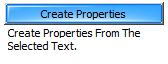
When you click the Add Method button the new method is inserted in the correct location (alphabetically within the Methods Region).



Notice the method summary is inserted and is waiting for a description; it is always a best practice to describe a method and the more information here some HELP generators look at method descriptions to build help file or other documentation files.

There are two features built into the Method creator that are time savers and I think you will appreciate it. If your method starts with Find or Get, a private variable is created for you as shown above by dropping the Find or Get text and creating a lower case variable (first character) of the remainder of the method name.

**Create Properties from Selection**

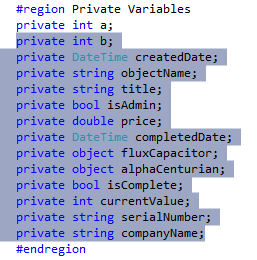
****

This feature I love and I am pretty sure you will think this is as cool as I do.

Refactoring multiple private variables into properties is a very time consuming task considering this is performed several times per day, every day so any time saved is money saved.

Using the standard refactoring tools of Visual Studio, the refactoring tools will create the properties for you, but Visual Studio places the newly created property directly below the private variable that is being refactored into a property. After creating all of your properties you also must cut and paste the newly created code into the properties region (if you are a Regionaholic like me) and you also must delete any blank lines created from the refactoring and all in all this takes quite some time.

Now with Regionizer you can select multiple private variables as shown below and click the create properties button.



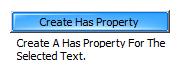
**Voila!** Your properties are created and inserted into the **properties region** in alphabetical order.

This feature alone justifies the cost of **Regionizer** and I think you will enjoy this.

Note: For this feature to work you must have a **Properties Region** in your code file.

I have tested creating up to 15 private variables at once, it should work on an unlimited number of private variables, but it does take longer the more variables you have as the code must be reparsed after each insert to get the correct alphabetical order of the newly inserted code file.

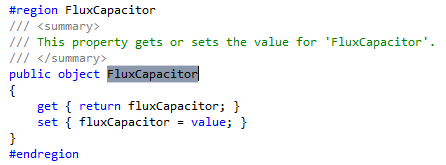
**Has Property Creator**

****

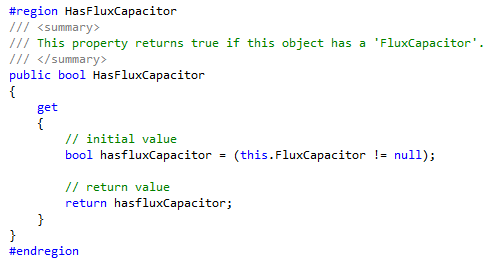
The **Create Has Property** button is another item I find very useful. What this button does is it creates a read only property that returns true if the object you have selected exists.

**Example:** I will demonstrate a sample usage of the **Create Has Property** button.

First you select the name of a property in your C# code file as shown below:



Now click the **Create Has Property** button and Regionizer will insert the newly created read only property as shown here:



**Regionizer** is smart enough so that if the property highlighted is a String, a test of String.IsNullOrEmpty is ran or if the return type of the property is an int or a double a test for greater than zero is performed.

You may be asking why you need to create a property that tests for null when you can just test if the object is not equal to null as needed in your code.

The answer is that by creating a property to test for null, the property shows up in Intellisense, so it is much faster when you refer to an object multiple times throughout your code.

If you have any questions please email [corbynichols@gmail.com](mailto:corbynichols@gmail.com) and I will do my best to respond as fast as possible.

If you have trouble with a particular code file and would like me to investigate you may send the file and a brief description of the problem and I will see if I can solve the problem quickly or if a major revision is required to solve the problem.

I hope you enjoy **Regionizer** as much as I do.

Corby Nichols

**Appendix**

**Known Issues**

Nested Classes (classes inside of other classes) are not supported yet (coming soon).

Delegates inside of a namespace are parsed with a Document Format, but delegates inside of a class are not supported yet (coming soon).

Structures inside a namespace of inside of a class are not support yet (coming soon).

Comments not associated with a code element (method, property, event, etc.) could be lost.

A class has a Code Notes collection of orphan comments, but currently the Document Formatter does not capture orphan comments (future enhancement).

Please report any issues to the email address above.