**Sara’s Script**

There are numerous ways to search within the Visual Studio Environment. The most common way is Quick Find. If I press Ctrl+F, you see the Quick Find dialog appear. We know it is Quick Find because it is here, selected in the drop down.

Quick Find is a great all-purpose, general search, but there are other searches that can increase your productivity depending on where you want to search.

In terms of complexity, Quick Find is right in the middle. There is a faster search called Incremental Search. And, there is a more advanced search called Find in Files. Let’s begin by looking at Incremental Search.

**Tip #1 – Incremental Search**

I prefer keeping my eyes as much as possible on my code. Unnecessary dialogs or windows can become distracting to me. Since I know I am going to search within my current document, I’m going to use Incremental Search. It’s great because it doesn’t bring up any windows, and it allows me to continue typing in the editor.

This is my Accessibility Testing Tool called MsaaVerify , which is my personal pet project that you can download off of CodePlex. I am going to run it to show you what it does, since we will be looking at the code quite a bit today. It verifies whether a screen reader will read the properties of common WinForm controls properly. You start by dragging the crosshair to any button. And since an accessibility testing tool has to be accessible, here you see where you can time in the Window Handle. When I press verify, we see that everything passed. Notice that there’s a child count verification. Let’s find the Child Count method (all one word) in the code.

I know it is in the AccessibleObject class, but not sure where. I can press Ctrl+I to begin incremental search. Note how the mouse pointer has changed to binoculars with a downward arrow indicating the direction of the search. Also, notice how the status bar changes to “Incremental Search.”

As I type ‘c’, we see the editor find the first match of the instance of ‘c’. Pressing ‘h’, we see the editor jump to the first occurrence of “ch”. Notice “ch” is highlighted in the status bar. Now if I type the full “ChildCount”, the editor highlights the first occurrence of the ChildCount name, but this is just the method being called.

I know this method isn’t called too frequently, so it must be just a few more occurrances down. I can now hit “Ctrl+I” to tell the editor to search for the next instance of “ChildCount” since this is what is highlighted.

Pressing Ctrl+I takes me to the next occurrence, but it isn’t the method. Pressing it one more time brings me to the actual method.

Let’s say I’m now finished examining this line of code, and want to search back up to where the method was called. I can now do reverse incremental search to go back up. I press Ctrl+Shift+I. Notice how the icon pointer has changed its binoculars to search upwards and the status bar has changed to indicate “reverse incremental search.”

That’s incremental search. A very powerful search when you want to keep your attention on your code while searching the current document. But, if you need to search outside the current document, you need a different search.

**Tip #2 – Find in Files**

Find in Files is the most advanced search in Visual Studio. I can bring up Find in Files by clicking on the Find in Files button next to the Find Combo Box. At first glance, the Find in Files window looks very similar to Quick Find, but there are several important new options that makes Find in Files the most advanced search.

The first option is the ability to choose your search locations. You notice the standard Look in: combo box which is available in Quick Find, where you can specify whether to search in the current document, all open documents, current project, or entire solution. To the right is the new option called “Choose Search Folders.” Pressing this button brings up the Choose Search Folders. Here I can specify where to search on disk. There is little test project available with MsaaVerify, a very inaccessible winform application, to test the MsaaVerify tool itself. I’m going to create a Search Folder for it by navigating to its location on disk. Now, I’ll select the folder I want to search in and add it. I’ll give the search folder a name, press apply, and hit okay. Now we see the “Test Project” search folder in the drop down.

Other options are included, like match case, match whole world, and whether to use wildcards or regular expressions. The new option that doesn’t exist in Quick Find is that you can specify what file type to look under. I just want to see code, so I’ll type in “\*.vb”

The Results options are also unique to Find in Files, where I can specify to save my results in a separate window. In fact, VS gives you two choices where to save results, so that you can have two different search results open at the same time. I’ll leave it in Find Results 1 Window.

I’m going to search for that ChildCount method to see how it is being used and tested in the test project. When I do a search, note how only the one result is found. This is because it isn’t searching anywhere but the Test Project search folder. Notice how the actual line is returned, and we see I’m overriding the GetChildCount. This is because I want to make the test project as inaccessible as possible.

Now I can double-click on the line and it opens the file. Now we can look at the comments that a button is only supposed to have a child count of 0, but to make sure MsaaVerify catches this error, we’ll give it a child count of 5.

Let’s move on to the final tip in the Search section.

**Tip #3 – Repeat Search.**

**F3**

Now that I know what the inaccessible test app is doing with ChildCount, I want to verify I’m testing it correctly in the Verification method. I open that file, by double-click on it in the solution explorer. Now I need to find that method. I could use any of the previously mentioned searches, but there’s a much faster way. Note how ChildCount is still visible in the Find Combo Box. This is because it was the last thing I searched for. I can now simply press **F3** and volia, I’m right at the ChildCount verification method in the base verification class.

**Ctrl+F3**

I’m one of those people who likes to read with my mouse, like so. I prefer to highlight the text that I need to concentrate on. If you are like me, you’re going to love this tip. And thanks to Chris McGuire, Developer on the Visual Studio team, for providing me with the following tip.

Let’s say I quickly want to jump to the next occurrence of this string Passed. I can just double-clicked it, and knowing me I probably already have the text selected. But if I don’t, I can just place the cursor on the word. Now, I can just press Ctrl+F3, and it’ll search within the current document all instances of Passed. Both F3 and Ctrl+F3 are shortcuts for Quick Finds.

**Editing**

**Tip #4 – Box Selection**

Notice how there are some constants here. Let’s say that I decided to change these from constants to variables. I could go through each line and change them one by one. But wouldn’t it be nice if I could just select the “Const” keyword. If I attempted to do so with just the mouse, I just can’t get the const. I can press and hold down the Alt key to do what is called a box selection. Holding down alt, I can down select straight down and get the Const keywords. Now I can hit delete, and the const is gone. But, these really need to be constants, so I’ll hit Undo through Ctrl+z, and there they are again.

**Tip #5 – Clipboard Ring**.

we’re going to now look at some time saving tips for Cutting, Copying , and Pasting code.

[AccessibleObject – Private Const and Variables]

I always forget this and then whenever I see it again, it’s like getting a present in the mail. Do you ever find yourself cutting multiple lines of code at a time, wishing you could just get back to that previously-cut line? For example, I often find myself thinking I only need to cut one line, but it turns out I need to cut two lines. How do I cut both lines without losing the first line?

Here I see that there’s a public const in the middle of all these private ones in a private region. I need to move this out. So I’m going to go ahead and use Ctrl+x to cut. Now I’m going to put it in its own region. Oh wait, I don’t have a private region. So I’m going to copy the line here using Ctrl+c. Now I realize that I’ve copied over my cut line! Actually I haven’t.

There’s a clipboard ring that keeps track of what you’ve copied. But first, let’s create the Public Region to live under the Private Region. I’m going to Ctrl+v to paste in the region. And finish the end region, by moving to the EOL and hitting enter, and obviously renaming the Private to Public. Now I want to paste in that first line. I can hit Ctrl+Shift+v to cycle through what’s on the clipboard ring and pressing it once gives me the second line. Pressing Ctrl+Shift+v again gives me the first line.

Actually, I don’t like the public region being below the private. I’m going to move the Public region up. I’m going to cut it via Ctrl+x, and now hit Ctrl+v. ARGH!! I accidentally hit Ctrl+x to cut a blank line. Now when I hit Ctrl+v, nothing happens. How many of you has this happened to you when you accidentally cut or copy a blank line?? I can go to Tools Options – Text Editor – All Langauges – General, and **uncheck** the Apply Cut or Copy to blank lines when there is no selection. And press okay.

I’m going to hit undo, to undo the cut, so the region comes back. Now let’s try this again and see what happens. I hit Cut, and accidentally hit Ctrl+c, and now hit Ctrl+v, it’s all good. My line is still there.

**Tip #6 – Toolbox General Tab**

Almost all of us have used the Toolbox at some point in time, but how many have noticed the General tab? This is an extremely useful feature. In fact, it has been my most recently popular tip on Tip of the Day. Let’s say that you need to reuse a certain snippet of code again and again. You may need to have certain comments for your methods or classes, or even at the top of your source code. Let’s say that under the copyright notice, I want to put the URL to CodePlex. What I can do is add

'MsaaVerify is hosted on CodePlex at <http://www.codeplex.com/msaaverify>

But I want to not only add this line of text to other files, but also save it in case I add more files later. What I can do is drag and drop it to the toolbox General tab. Now I can drag and drop into other files. I can also drive from the keyboard by hitting Ctrl+c, pressing Ctrl+Alt+x to reach the toolbox, then Ctrl+v to paste it in. You can also hit enter with the focus on the item in the toolbox to insert it at the cursor position in the editor. You can also just double-click on the item via the mouse to insert it at the cursor position too.

This is also extremely useful for presentations, where you can store content so that you can quickly add it to the editor.

**Tip #7 – Snippet Tab Tab**

One night I was having dinner with Chris Woodruff and others who were in town taking a certification training exam. One of the developers had asked me to include a tip on Tip of the Day about “snippet tab tab.” I was taken aback, because I had extensively tested this feature during its development and had never heard of the “tab tab” feature. As he explained, I realized he was just inserting snippets through the keyboard, but I never thought of calling it “tab tab.” Pretty cool.

For this tip I’m going to switch to C# for the full effect, since the customer who coined the term was using C#. Actually, I’m only going to use this project for demo’ing purposes, so I don’t really need to save it. I’m going to go to Tools – Options – Projects and Solutions – General - and uncheck Save new projects when created. Now let’s create that C# project. Notice how there’s no information for me to save. VS is going to put this project in a temp folder for me until I decide to save it.

Let’s start exploring “snippet tab tab” by looking at what a snippet is. Starting in Visual Studio 2005, we have pre-written code snippets for you to use, modify, or create your own. You can find the entire collection of them under Tools – Code Snippet Manager. We see we are looking at C# snippets. Let’s look at a simple snippet, like the “for” loop snippet. I will drill down into the C# folder, and look for the word “for”. We see the file path, the shortcut, that it can either be inserted directly into the code or use can use it to surround some existing code, and lastly Microsoft is the author. We can copy the file path for the snippet, and open it directly in the editor via File – open – file. Looking directly at the .snippet file, we see that it is XML, where you can find attributes like the shortcut.

The reason I chose the C# for snippet is because I’m primarily a VB developer, so I always forget C# syntax. Instead of having to pick up a book to remember the syntax, I can just insert the snippet. I can go to Edit – Intellisense – Insert snippet. Here we see the Insert Snippet Window appear. I can click on the C# folder (just like we saw in the Code Snippet Manager) , and select for. We see some helpful tooltip text reminding me that there’s a shortcut. Now I click on the “for” and we see the snippet inserted. Noticed that there are some fields in green. These indicate variables that I can modify. If I don’t want to use ‘i’ as the index, or I can’t if it is already defined in the scope I’m working in, I can just say “ii”, and hit tab to move to the next field. Notice the I’s change to ii’s. I can hit enter now to commit my variable changes.

But let’s see this again using the “tab tab” feature. Recall that the “for” snippet has a “for” shortcut. I can type in “for” and notice how it appears in statement completion. It even gives me a little icon that says, “hey this is a code snippet.” If I hit tab once, this dismisses the statement completion, and if I hit tab again, it inserts the code snippet. Let’s see that again… “as for tab tab.”

Maybe you know your C# syntax. You can still find very interesting built-in snippets. If you need to reverse a for loop, you can insert it within 2 seconds by typing in “forr” noticed the “reverse” in the snippet tooltip, and then “tab tab”.

**Tip # 8 – Command Window**

Remember earlier that we talked about Search, and how there’s a Find Combo Box up here. This combo box will always do a Quick Find. But, did you know the Find Combo Box runs Visual Studio Commands.

This tip comes from a Tip of the Day blog reader who wanted to know whether it was possible to open a file by simply typing in the editor without having to open any dialogs or windows. He had asked because other developer environment allow for this. I had never thought of trying this before, so I was pretty excited to respond to him that it was possible and here are the steps. But first, let’s explore Visual Studio Commands and how they work in the Command Window.

I bring up the command window by going to View -> Other Window -> Command Window. Here I can view and run Visual Studio commands. Let’s explore the File commands by typing “File.” Let’s say I want to run the File.OpenFile command. I can find it in auto-complete, and insert it, and hit enter. And the File.OpenFile dialog appears. There’s another command called “alias” where you can alias a command to another word, just like in the Windows command shell. Let’s alias File.OpenFile to something simplier, like fo. I’ll type in “alias fo File.OpenFile” The status bar now indicates that fo has been aliased. To verify this works, just type in “fo” and we see the Open File dialog appear.

Let’s now go back to the Find Combo Box. From the editor, I can hit Ctrl+D to reach the Find Combo Box, ready to do a search. But I don’t want to search, I want to run commands. To tell VS to run a command instead of searching, insert a ‘>’, like so. Now we can do “> File.OpenFile” We still get intellisense and the command opens the Open File dialog. But we aliased the command, so I can type in “fo”, and even see it in the intellisense. Running “fo” we get the Open File dialog.

Let’s get back now to the original question, “how do I open a file by typing in the editor without opening any windows.” Since I’m in the general development settings, I can use Ctrl+/ to jump to the File Combo Box, and notice how VS inserted the ‘<’ for me. I can now type fo, and if I hit a space and begin typing the first letter, I get intellisense for the project. And I’ll select the file and hit enter, and now the file is opened.

Let’s see all of this again. I’m typing in the editor and want to open that Class1.cs. I can just type from the editor “Ctrl+/ fo program2.cs” and now I’m in the editor for Class1.cs”

The response from the blog reader was “I’m officially a Microsoft convert after 7 years of Java development.” Tip of the Day – winning the hearts and minds one tip at a time…

*Now if you’ve been sitting here this entire time thinking, “okay sara, show me something I haven’t seen before. I got one for you.” Not only can you run commands from the Find Combo Box, you can also specify command parameters. For example, if I want to set a breakpoint to a particular function, instead of bringing up the breakpoint window or finding the file in the editor, I can just type the name of the function, like “Main” in this case “note how this isn’t a command, so no ‘<’”, and if I hit F9, the breakpoint is set. I got to show Doug Hodges, a principal architect for Visual Studio this tip.*

**Tip #9 – File Tab Channel**

Now I want to move out of the editor and into more generic aspects of the Visual Studio environment. I’m going to go back to the MsaaVerify project now. As I close this project, notice how I’m prompted to save. Remember we said don’t save new project when they are created. I can now discard the project, and avoiding the ConsoleApplication57 phenonomia that we see on our computers over time.

This row at the top from the tab fin to the ‘x’ close button represents the file tab channel. There are all sorts of wonderful things you can do here. This ‘down arrow” will show any files that are open in the editor, even if they are not visible. For example, let’s resize the tool windows so that a few of the files “fall off” of the file tab channel. Now at least two tabs have fallen off. Notice the icon has even changed to a bar over the arrow. Now if I click on the drop down and choose the accessibleobject file, notice it comes back onto the file tab channel.

Each file tab has some interesting commands. If we right-click on a file tab, we see “close all but this”, which does exactly what it says. It’s especially useful when there are way too many files open and you need to get back in control. The copy full path is just wonderful. It saves you so much time from having to go to the Solution Explorer, pressing F4 to go to the properties window, and trying to copy the file path from the small edit box. The “open containing folder” is equally as useful if you need to directly jump to that file location on disk, instead of having to open Windows Explorer yourself and manually going to the file.

**Tip #10 – Change Keybindings**

One of the first features I ever blogged about was the Ctrl+Tab window. It was amazing to watch on my blog how people either loved it or hated it as they left comments on my blog post. Let’s open a few files to get the full effect. If I hit Ctrl+Tab while holding down the Ctrl key, I get the bird’s eye view on both files and tool windows opened in the editor. When I release, I navigate to my most recently used document in the editor. For those more comfortable with the Visual Studio .NET 2003 Ctrl+tab behavior, which just cycled through all the open documents without popping up the Ctrl+Tab window, you can go to Tools – Options – Environment – Keyboard. In the “shortcut keys” type in Ctrl+tab. We see below that this shortcut is used by the Window.NextDocumentWindowNav. Let’s type this command in up here under “show commands containing.” There are two, one for the 2003 behavior and the other for the Ctrl+Tab window, codenamed IDE Navigator, hence the “nav” at the end of the command.

I’m going to select “Window.NextDocumentWindow” and press “assign” this will assign the Ctrl+tab keystroke to this command, as indicated in the “Shortcuts for the selected command.” Note that Ctrl+F6 is still there and will work also. I’ll press OK to continue.

Now I can cycle through all my open documents.

**Tip #11 – Changing Font Sizes**

Recall earlier how I had a custom settings file ready to go that contained increased environment, statement completion and tooltip sizes. Now I will manually walk you through the process of setting up your environment the same way I have mine setup.

All these options can be found under Tools – Options – Environment – Fonts and Colors. To increase the environment fonts, drop down the “show settings for” combo box and select “Statement Completion”, here you can select to change the Font Style and the Font Size. You can change the tooltip that appears in the editor by dropping down the “show settings for” and selecting editor tooltip. Same options are available for both.

For the environment font, you can find it also in the “show settings for”, and you can select a font and font size. **This is a new VS 2008 feature.**

To reset back to what’s in your environment development settings, you can press the “use defaults”. This will set the environment fonts back to their defaults for the general development settings. Note that the Use Default applies to just what’s selected in the combo box. For example, if we pressed “Use Defaults” here for the environment font, the statement completion and tooltip sizes would remain the same.

**Tip #12 – Window Layouts**

The next aspect of the general Visual Studio environment you will learn more about is window layouts. There are four window layouts: Design mode (which I’m in right now), debug mode (while running or debugging), full screen mode (which I’ll demonstrate in a second), and the file mode (the mode Visual Studio goes into when you open a file from the command line). Any customizations made in one mode will persist in that mode, and only in that mode, meaning I can customize my design environment differently than my debug environment, and so forth.

Full Screen Mode is the “Start Slide Show” that’s in Power Point, meaning that it not only In many presentations, you just want to focus on a deep dive of the code, and not all the features of Visual Studio. Obviously this talk is an exception, but let’s see just the editor. I go to View – Full Screen mode.

What’s happened? We see that all of the tool windows are gone and all the command bars are gone,a and even the title bar and Windows Taskbar are gone. We also see the Full Screen View button up here at the top. I’m now in the window layout called full screen mode. It’s just a focus on the editor. And I can toggle out of it by hitting the Full Screen View button, but before I do that, I want to make a few customizations. Let’s say that I really want to show a tool window here. I can bring up solution explorer through View – Solution Explorer. And let’s add a command bar. I can go to Tools – Customize, and add the Text Editor. Now I have the Text Editor and the Solution Explorer showing.

Now I’ll go back to design mode by clicking the button. Notice how the Windows Taskbar comes back, the tool windows come back, and the title bar comes back. If I go into Full Screen Mode again, we noticed that the Text Editor command bar and the Solution Explorer have persisted. These customizations will persist across Visual Studio sessions and can be saved in your settings file through Tools – Import / Export Settings.

**Tip #13 – Miscellaneous files project**

Recall how the Test Project for MsaaVerify lives outside the MsaaVerify project. I don’t want to add the Test Project to my solution because there’s no need. But sometimes I want to keep track of those external files so that I don’t have to reopen them every time i close Visual Studio or close the file itself.

I go to Tools – Options – Environment – Documents – and under the Show Miscellaneous Files in Solution Explorer. Since I want these files to come with me, I’ll mark to save at least 5 of them.

Now I’m going to open up MyPushButton.vb, a very inaccessible push button. Notice the “miscellaneous files” project that’s been created. I can close the document in the editor, and the file remains in the solution explorer. I can simply double-click to get it back. I can reopen the solution and see how the file comes along with me.

I remember finding this feature extremely useful when I wrote my test cases. Each test case had numerous references to the overall automation framework. Whenever I wanted to step into the automation framework, many many files would be opened that I would later reference. But whenever I closed Visual Studio or closed those files from the editor, they would go away. Using this feature, I could carry them with me in the solution explorer and never have to worry about “where does that live again? Or, what was that file again that was so important.” Now I have the files I need with me.

**Tip #14 – Opening VS to what you want**

Ever just launch Visual Studio in the mornings and wish it would load your project for you, so you could just jump right in where you left off the night, or even the session, before? You can, by going to Tools – Options – Startup, and under the At Startup combo box, you can customize it to show your VS web browser home page, the new project dialog, the open file dialog, an empty environment, or the last loaded solution. In the general development settings, the default is to show the start page.

Here’s I’ll select to show the last loaded solution. Note how I have my accessibility tool opened, and the MyPushButton active. Now I’ll restart VS. I loved this feature when I was a SDET when I would write automation.

**Tip #15 - Tracepoints**

Tracepoints are a new Visual Studio 2008 feature. Consider for a moment how many of you debug your applications by putting in Console.WriteLine or PrintF statements? Then after you are finished debugging, you have to go back and remove those statements.

Tracepoints are built in “console.writeline” or “printf” statements that will log for you, without every modifying your code. Let’s say I need to log the (x,y) coordinates whenever I drag the crosshairs in my accessibility tool. I’m going to search for “handlemouse” using Quick Find and when I draw the rectangle around the control, I’ll insert a tracepoint to get the x,y values.

Right clicking on the line, I’ll select Breakpoint – Insert tracepoint. You see that by default VS will suggest logging the function name and thread name and id. I just need the variables logged, so I’ll enter "({x}, {y})" to get the nice x,y coordinates. Note the icon. Instead of the circular breakpoint icon, we get a diamond.

Now when I run MsaaVerify, and drag the crosshairs around, I will be logging tracepoints. Now that I’ve probably logged a few, I’ll quit the application, and check out the tracepoints we got. I’ll go to the Output Window’s Debug pane, as indicated here, and you see the coordinates were recorded, and I never had to edit my code.

**Tip #16 – Remove Unused Using Statements**

For this tip, I’ll create a new C# project. Note how it still says ConsoleApplication1, instead of ConsoleApplication2. I love having control of my environment.

Back in my software testing days, I wished we had this feature, because we would always start our test cases from either copying a previous test case or from the generic test case template. From time to time, we needed to remove unused using statements, but we would have to go to each using statement and comment it out, and hit compile. If we got errors, we knew we were using it. We would have to do it for each and every using statement. It got old, fast.

There are three using statements, but I’m pretty sure this small console app only needs the System using statement. I’ll right click to bring up the context menu, and choose the “Organize Usings” and click the “remove unused usings”. We notice that two of the three using statements have disappeared, saving me significant time from having to uncomment, built, and rebuild each line.

**Tip # 18 – Keep the console window open**

Charlie Calvert, the C# Community PM who did the Tuesday version of this talk, did a dry run at a user group in Bellingham, WA. The group lead who hosted the event took us out to dinner afterwards and provided us with the following tip.

Similar to the tracepoints, you find yourself having to edit code to get Visual Studio behave a certain way. Another such example is with the console window. How many of us have had to type in Console.Read() at the end of our application to keep the console window open. For example, if I were to run this simple winform app, the console window immediately disappears. So, I’ll type in Console.Read() just to keep it open. Now the console app waits for me to enter a keystroke before terminating.

I strongly dislike having to edit my code in order to get a particular debug behavior. It runs the risk of having the code checked-in or forgotten about. The solution is to hold down the Ctrl key when you hit F5. This does a “Start without Debugging.” Now, we see the console application display “press any key to continue…”

**Tip #19 – DataTips and Watch Window**

I’m going to modify the contents of the application to contain a string to store the message in.

One of the absolute can’t live without features for debugging is the watch window. I’m going to start a debugging session by setting a breakpoint on the End Main } and hitting F5. To the bottom left-hand corner, we see the Watch window, called watch 1. You can have up to 4 different watch windows. Now I can either type msg into the edit box, or drag and drop msg into the watch window.

Now, the value column is actually editable. I’m going to hit tab to navigate to the value column and start typing a new value “go sara go”, the battle cry of the Tip of the Day series. Notice how the value has changed to red, indicating it is a new value in use. Now if I drag the current statement up one line to repeat the console.writeline and press F10 to execute current statement, we see

Again, I personally like spending my time in the Editor, and try to avoid going to tool windows unless I have to. There’s a feature called DataTips that you can do the same changing of values, but without leaving the editor.

I’ll set the breakpoint on the Console.Writeline statement and hit F5. Now if I hover over the “msg” I get the datatip. I can now click on the value and type in “go sara go” and press enter. Note how the watch window reflects the change in red.

**Tip #20 – Immediate Window**

When I tested the generic Tool Windows functionality in Visual Studio, I had to do a lot of math and formula calculations. The immediate window was extremely helpful to calculate expressions without having to edit my code at all.

Let’s remove these two lines by typing in Ctrl+Shift+L, which deletes the line.

Let’s type in a simple calculation.

Now when we run, with the breakpoint on the end main, we’ll bring up the immediate window. It is found under Debug – Windows – Immediate. I can type things like “1+1” and get 2. I can create temporary variables, like int I = 1 + 1; I can now check the contents of a variable by doing ?i and seeing 2 returned.

Now let’s look at the wages formula. I may realize that 200 is too low and I’ve made a mistake in the rate. It needed to be 6. I can then in the immediate window type in rate = 6. Doing a ?rate confirms. Now if I rerun the expression in the editor, and go back to the immediate window, I can see the new value.

Also, all the VS commands are available when you hit the “>”

**Tip #21 – Recording macros**

*Need a good editing scenario…*

Great way to learn more about the VS object model.

Check out my 133 line contribution to Visual Studio located in the Macro IDE under Accessibility.