What’s New in VS 2017 Demos

Setup

* Turn off Lightweight Solution Load
* Reset any code styles you changed

Installation

Visual Studio 2017 significantly cuts down the time it takes to install, open Visual Studio, and write code in your solution. The new Visual Studio installer gives you the freedom to pick and choose exactly what you want installed.

**Demo:**

1. Launch Installer.
2. You can launch or modify installed version from here.

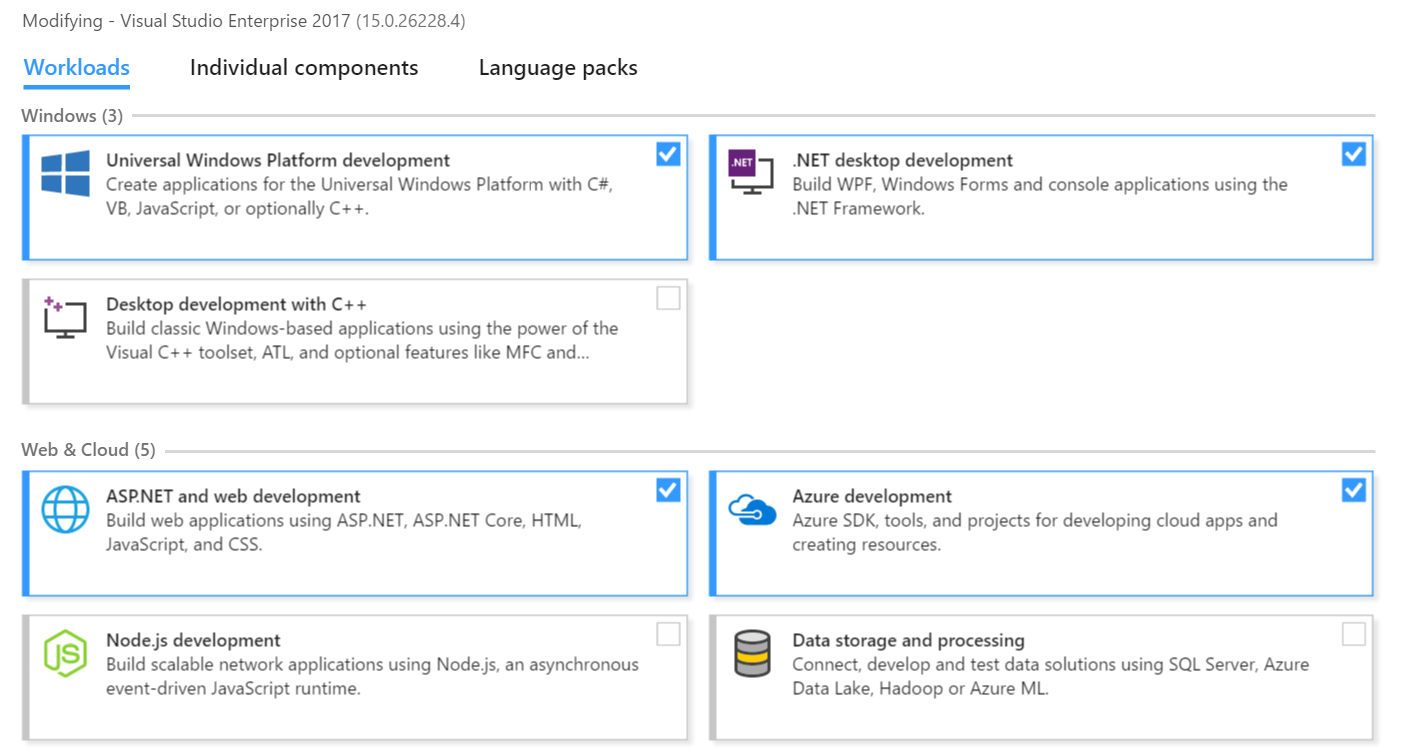
You can install multiple editions side by side. Note that the VS core assemblies are no longer installed to the GAC. They are in Program Files. So having multiple editions takes more disk space. But it also means you can install and uninstall without messing up your other editions.

1. Show Explorer. How much disk space is free? Show folder structure.

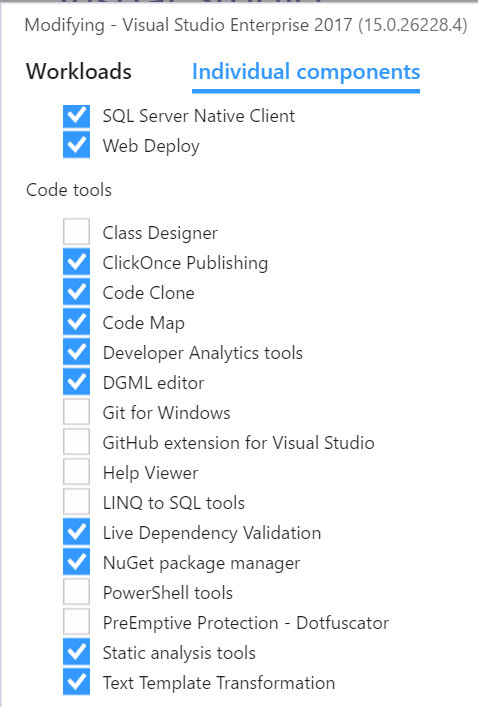
|  |  |
| --- | --- |
|  |  |

1. Install Community

You can choose workloads, which are sets of components that target a specific development scenario.



Make sure you check out the Individual Components tab. There may be things you want that aren’t installed.



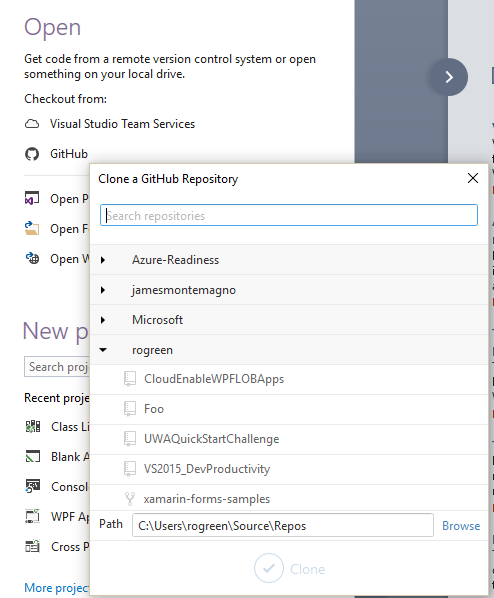
1. Show Explorer. How much disk space is free? Show folder structure.
2. Show that you have 2 installations of VS and they both take up disk space. There is little file sharing across the two at this point. You could update them separately if you wanted.

Note: Visual Studio core assemblies are no longer installed to the Global Assembly Cache (GAC). In order to support the installation of multiple editions, most assemblies required by Visual Studio 2017 now reside in C:\Program Files (x86)\Microsoft Visual Studio\2017\%editionName%\Common7\IDE\PublicAssemblies, where %editionName% is the installed edition (such as Community, Professional, or Enterprise).

Redesigned Start Page

Prioritizes the actions that help you get to code and start working faster

* MRU project list is most prominent
* support for recently opened folders
* MRU list roams across machines if the project is in VSTS or GitHub
* You can pin items in the MRUI
* Create a new project directly from the Start Page with searchable templates and a list of recently used templates.
* Ability to clone repo from TFS or VSTS under open. Ability to clone from GitHub if you install their extension.
* Ability to open folders
* Start Page is now under File.



Solutions

# Lightweight Solution Load

Behind the scenes, Lightweight Solution Load delays loading some projects until you actually need them. This implies that some features, such as code refactoring, code navigation, or inline renaming, might require some extra milliseconds the first time you invoke them. In conclusion, Lightweight Solution Load is a very useful addition, but only with large solutions made of C#/VB projects or with mixed C#/C++ solutions.

**Demo:**

1. Open Expenses solution.
2. Watch initializing and loading of projects
3. Close solution
4. Turn on Lightweight solution loading
5. Open solution
6. Expand a project to be able to work with it

IDE

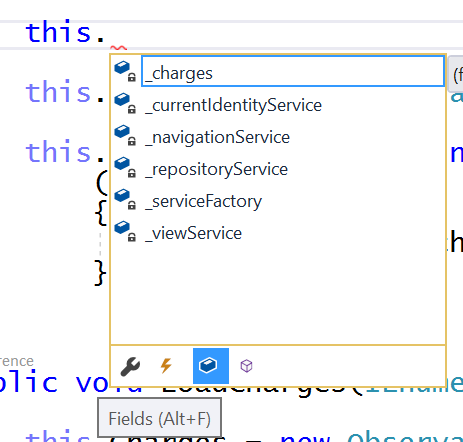
# Code editing

## IntelliSense filtering

You can choose to see only properties, events or methods. Can use shortcut keys Alt+P, Alt+V, Alt+M.

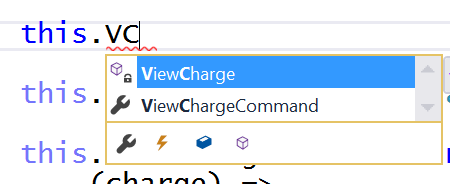
**Demo:**

* Open ChargesViewModel.
* Go to the constructor.



## Match Highlighting

You can use camel case matching to narrow down list of members. Note that is case sensitive, so typing vc below has no effect. Typing VC gives the results below.



Note, you can turn this off in the IntelliSense options if it bugs you.

## Better IntelliSense Behaviors

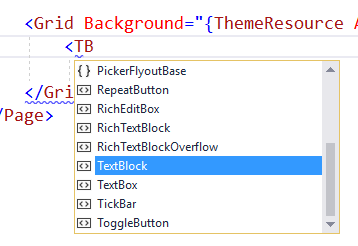
C# and VB have “smart preselection”. This determines the target type you are likely to need, and preselects the items in the IntelliSense list that match that type, speeding your typing flow and removing the burden of figuring out what type you need.

Demo: Type this.c and Charges is highlighted. Now you only have to type a period rather than a tab followed by a period.

<See <https://channel9.msdn.com/events/Connect/2016/149> for more>

XAML now has IntelliSense for namespace completion, lightbulb quick fix for missing namespace, sorting and removing unnecessary namespaces, and rename refactoring support for namespace prefixes. In addition, XAML IntelliSense has been improved to help you bind quickly and correctly when binding events, paths and functions with x:Bind; better filtering so that you see only relevant information; and camel case matching support (for example, “RTB” will complete as “RichTextBox”).

<See <https://channel9.msdn.com/Events/Connect/2016/110> for more>



Opening any Xamarin.Forms XAML document now provides a significantly improved IntelliSense experience in Visual Studio 2017. The new code completion engine supports bindings, custom properties, custom controls, converters, and much more.

# Refactorings

Visual Studio 2017 expands the set of refactorings and fixes to help you maintain a readable code base and catalyze your development workflows. For example, a significant number of developers initially write all their classes, interfaces, and other types in a single file and then extract each type into a file with the matching name later. Visual Studio 2017 expedites this process with the refactoring option “Move Type To Matching File.” Other refactorings you can look forward to include:

* Sync file and type name
* Convert property to method
* Use object initializer
* Convert null-check + throw to use ?? + throw
* Convert string.Format to interpolated string
* Make method synchronous
* Add missing case
* Add braces

Additionally, this release introduces some basic code analysis and fixes for XAML. Using the same lightbulb mechanism in C# and Visual Basic, you can sort and remove unnecessary name­spaces and add missing namespaces in your XAML files.

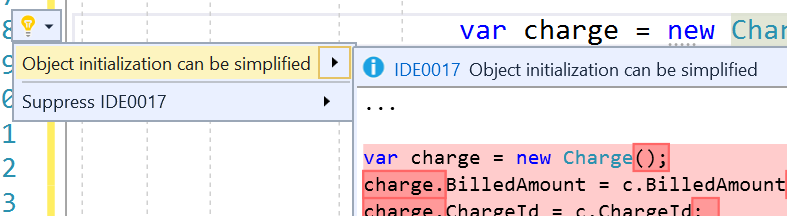
<See <https://msdn.microsoft.com/magazine/mt790181> for more>

**Simplify object initialization**

This replaces an object initialization based on property assignments with another one based on object initializers.

**Demo:**

1. Go to SaveChargesAsync method in ChargesViewModel.
2. Look at the code that declares charge.
3. Notice the dots under new.
4. Look at the tooltip.



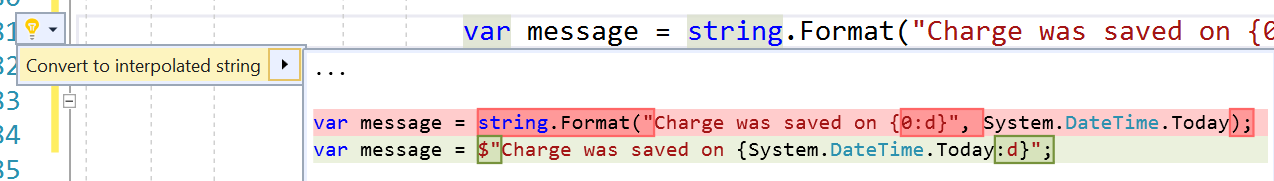
1. Make the change.

**Convert to interpolated string**

You used to have to download a code analyzer to get this. Now it is built in.

**Demo:**

1. Look at the definitions of message.
2. Look at the tooltip.

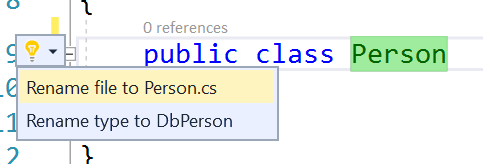


**Synchronizing type name and file name**

This refactoring keeps type and file names in sync.

**Demo:**

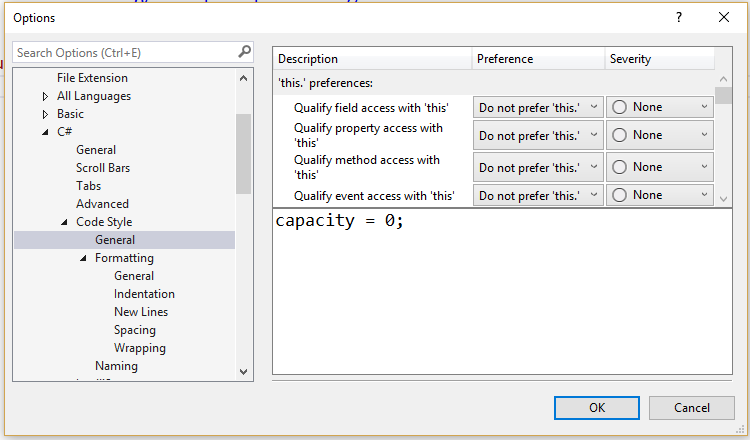
1. Add a Person class to the Models folder.
2. Rename the file to DbPerson and VS prompts you to change the class name.
3. Rename the class back to Person and notice the tooltip.



# Code Styles

We've added/updated style analyzers to help you customize and enforce coding conventions on your team, including:

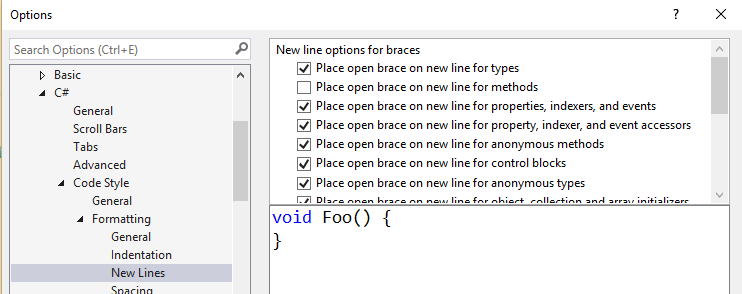
* Naming style rules.
* Use of “var” or explicit types.
* Use of “this.” or “Me.” on member access.
* Missing braces.
* Missing switch case.



Review the various options.

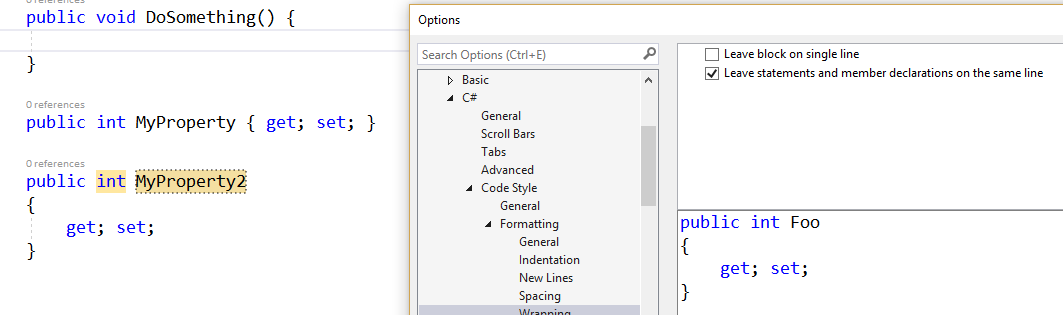
Type public void DoSomething() { } then hit enter and the braces are both on separate lines.

Uncheck that option

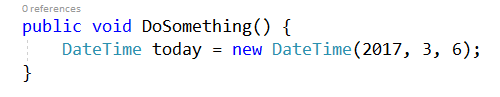


Type public void DoSomethingElse() { } then hit enter and open brace is not on new line.

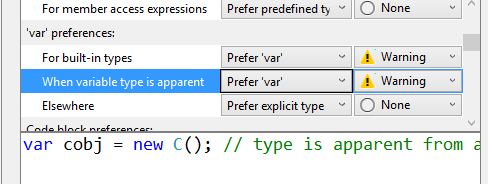
Create a new property and it is all on one line. Change that setting and it is now on multiple lines.

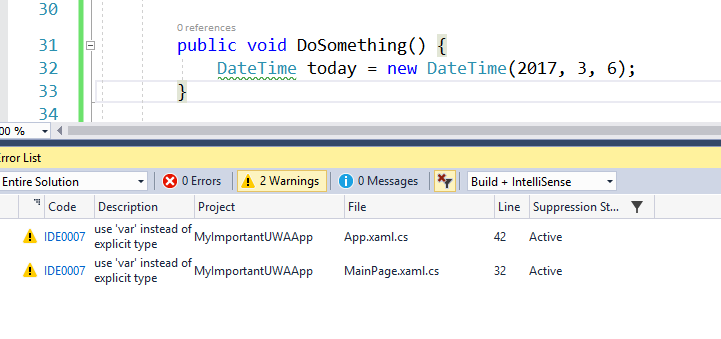


Let’s say you wanted to teach yourself to use var more often. This is the default.

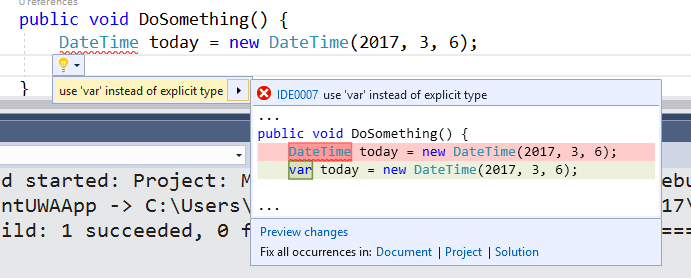


Set a code style so that it is now suggested you use it.

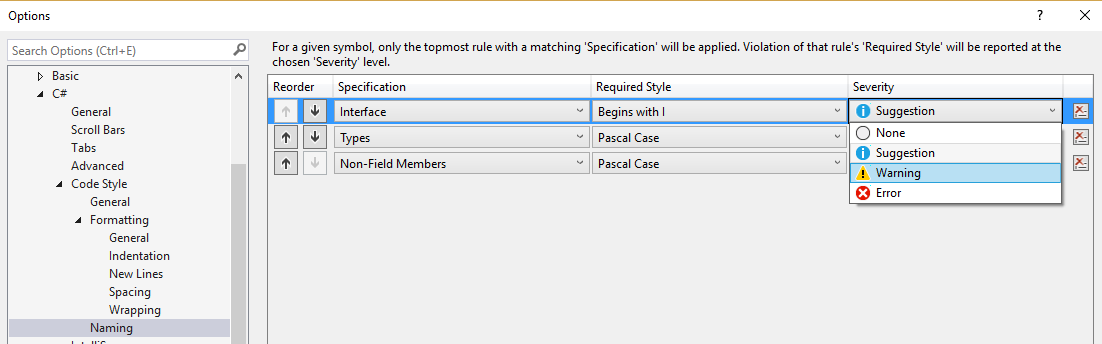




Set it to error to force yourself to do this! 😊



Lots of options in the coding styles



# Coding Convention Support via .editorconfig

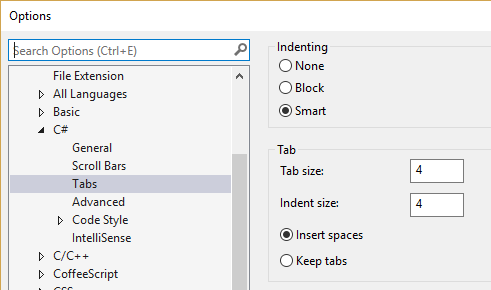
These are all at the IDE level. What if you want to have particular rules for particular projects? You can use edtiorconfig files to achieve that.

To help you achieve consistent formatting on your projects, we’ve introduced built-in support for the public editorconfig convention file format. This lets you define formatting rules that travel with your codebase wherever it goes. This means the code that gets checked in follows the codebase’s conventions and preferences rather than the individual developer’s.

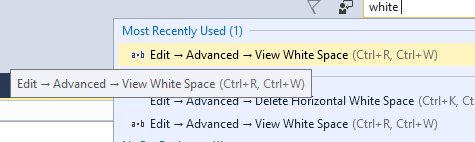
EditorConfig is an open source file format that helps developers configure and enforce formatting and code style conventions to achieve consistent, more readable codebases. EditorConfig files are easily checked into source control and are applied at repository and project levels. EditorConfig conventions override their equivalents in your personal settings, such that the conventions of the codebase take precedence over the individual developer.

**Demo:**

By default, when you hit tab, you add 4 spaces.



Turn on View White Space to see spaces.



Now add another project to your solution. For this project, you want tabs when you press Tab. If you go into options and click Keep tabs, that will apply to both projects.

The way to get it in only one project is to create an editorconfig file.

1. Add a code file named editorconfig to the second project.
2. Add the following code:

root = true

[\*.cs]

indent\_style=tab

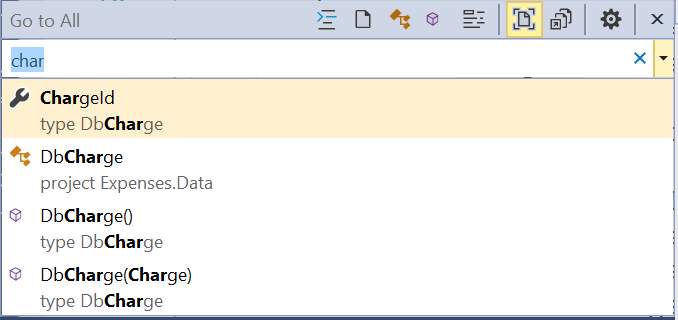
1. Save the file.
2. Close and reopen the solution. Now you have tabs in the second project and spaces everywhere else.

# Navigation

Many of you tell us that you spend much of your time working with large existing bodies of code. Having the ability to quickly and easily navigate your code is immensely powerful and can dramatically improve productivity. Whether you’re drilling into a bug, finding the implications of a refactoring, or just figuring out an unfamiliar codebase, tools that get you around quickly and confidently, and help you know where you are will make a big difference. That’s why we’ve added several new navigation features to help:

## Go To

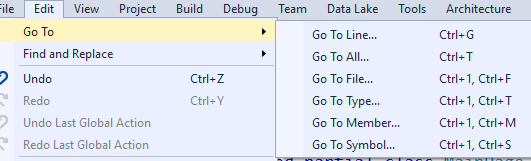
Formerly known as Navigate To. Ctrl-T is a fast, complete search that lets you quickly find files, types, methods, and other kinds of symbols in your code. It gives you a one-stop way to go to any kind of item you need, which is a particular benefit when working in larger code bases.



You can filter by item by clicking or by keyboard shortcut. Notice the letters that show up, eg t or m. These tell you what kind of object you are searching.

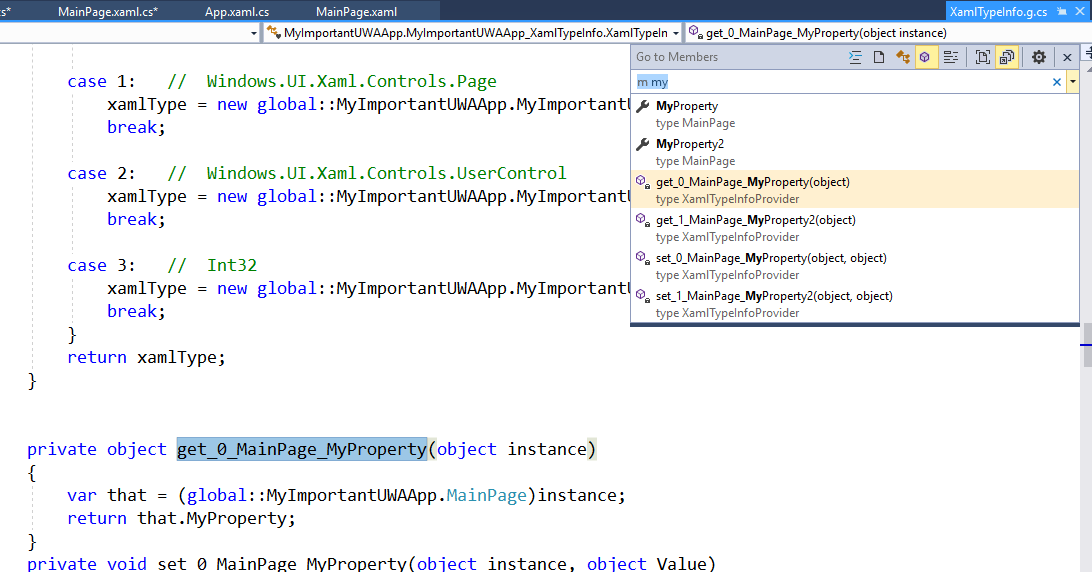
|  |  |
| --- | --- |
|  |  |

There are keyboard shortcuts for each of the filters.

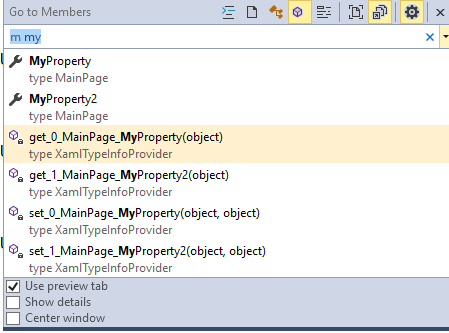


You can type the letter first and that will filter. Eg type t char.

As you scroll, you go to that location if the file is open. Otherwise you see a preview



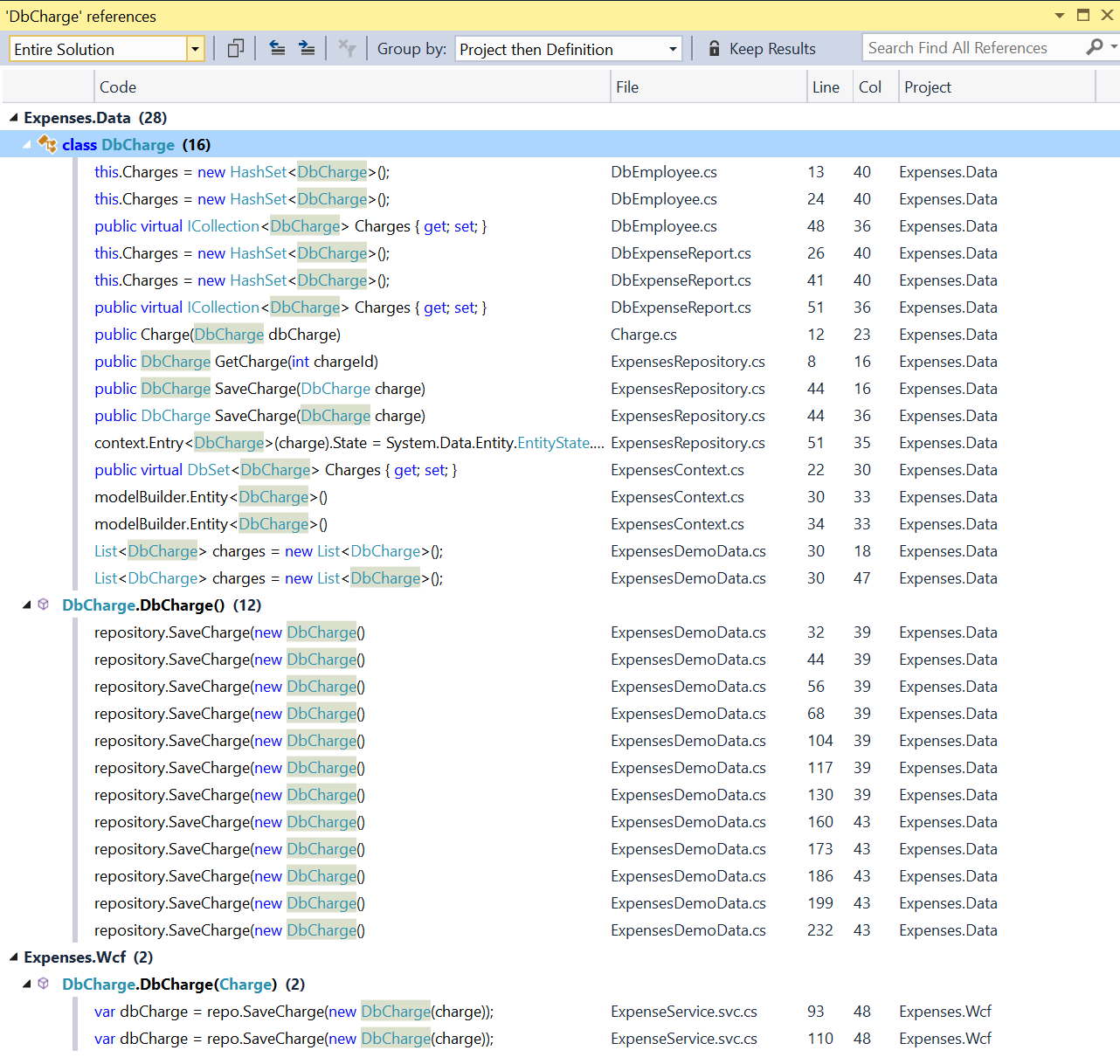
Click the gear to customize



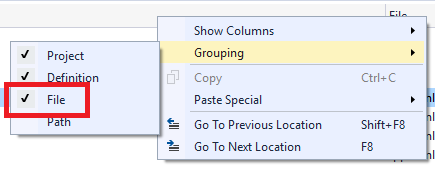
## Find All References

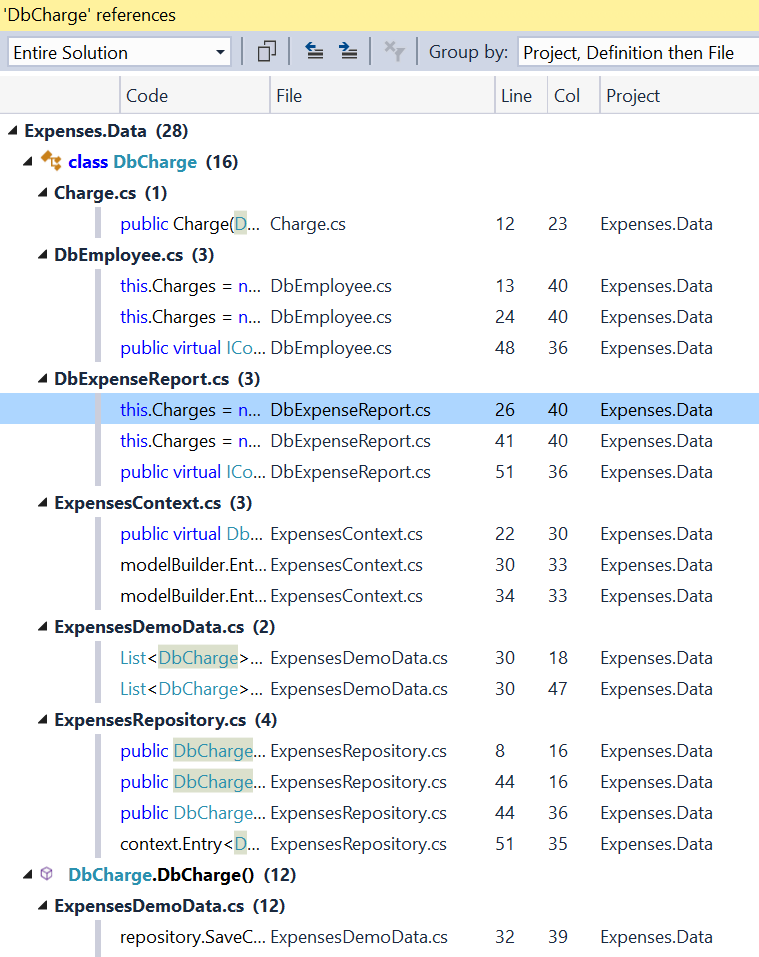
Find All References provides advanced grouping, filtering, sorting, and searching within your results. For some languages, such as C#, the results are colorized just like they are in the editor.

**Demo**: Goto DbCharge class definition.

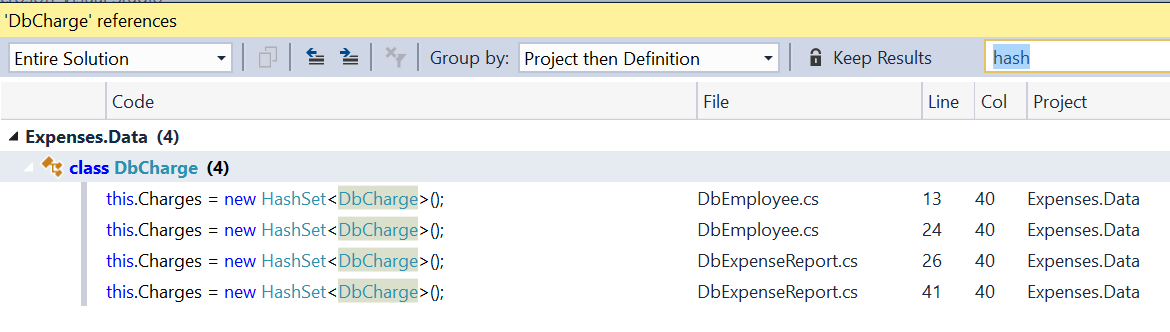


The “Group By” dropdown provides useful preset groupings, and you can even create your own by right clicking within the FAR results and using the Grouping context menu.





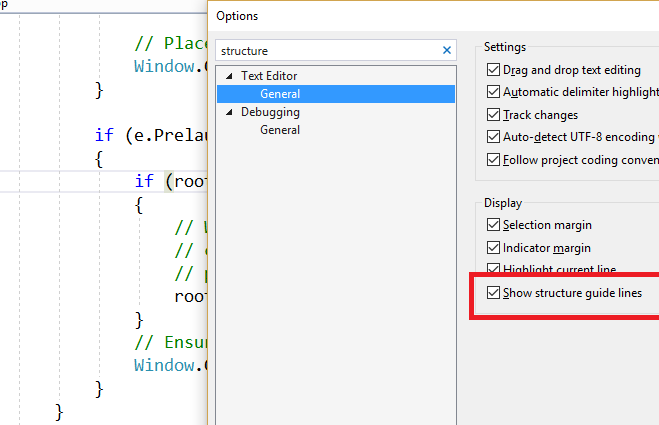
You can search within the results.



Click Keep Results to freeze the results you have. Then you get a brand new window the next time you find all references.

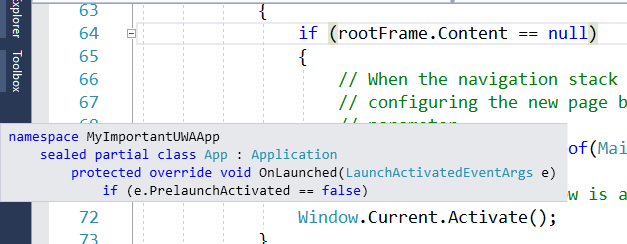
## Structure Visualizer

It’s important to have a clear sense of where you are structurally in your code when reading. To that end, we’ve added a favorite feature from the Productivity Power Tools extension into Visual Studio: Structure Visualizer. This draws structure guide lines (aka indent guides) on your code so you can easily visualize and discover what block of code you’re in at any time without needing to scroll.



Structure Visualizer works for XAML, C# and VB files, and for all the languages supported via TextMate grammars, eg Ruby. You can expect to see more languages add support later. For some languages (like C#, VB), the tooltips are colorized just like the code they reflect too.

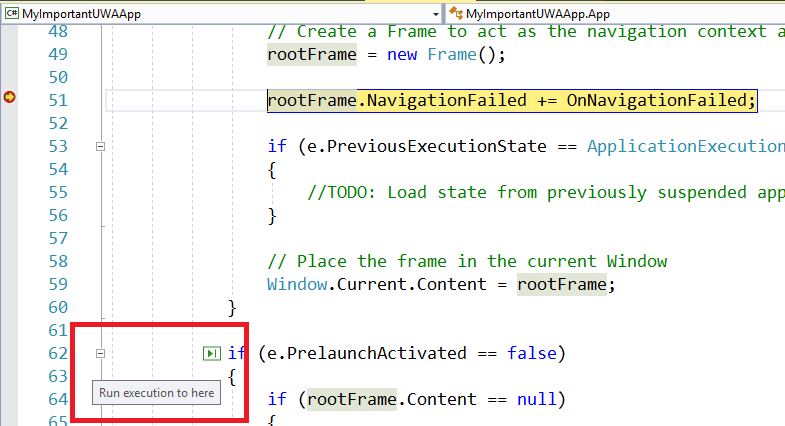
Hovering the lines shows tooltips that let you see the opening of that block and its parents. This is particularly useful when those blocks are scrolled off-screen. The guides are also handy for checking whether your code is at the right indent level!



# Debugging

You have many options when navigating through your code with the debugger in Visual Studio including setting breakpoints, stepping, and using Run to Cursor. In Visual Studio 2017 we have introduced Run to Click, a new way to more easily debug your code – point and click style. You no longer need to set temporary breakpoints or step several times to execute your code and stop on the line you want. You now can get all the benefits of Run to Cursor (Ctrl+Shift+F10) without searching through the context menu or taking your hand off the mouse for a two handed shortcut combination. Run to Click works while debugging in any programming language in VS including C#, VB, C++, and Python.

When stopped in the debugger, simply click the icon that appears next to the line of code your mouse is over. Your code will run and stop on that line the next time it is hit in your code path. This is called Run to Click. Essentially, it removes the need for developers to constantly add, hit and remove temporary breakpoints by combining all these actions into one click.



This is especially useful if you naturally have one hand on the mouse while debugging to inspect variables with data tips in your code. You can quickly Run to Click a line of code, inspect the variables on that line, and then continue debugging all while keeping focus and attention in that location. Run to Click between the same method, different methods, and within loops!

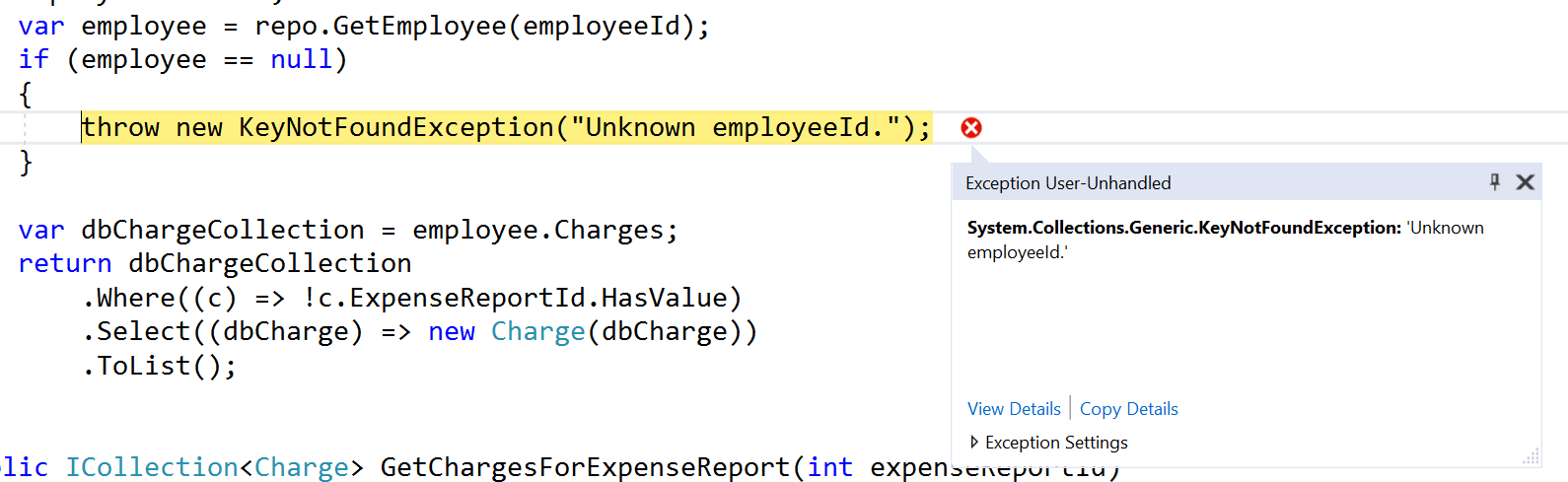
You can turn off Run to Click from the checkbox Debug/Options/Enable Run to Click.

### Exception Helper

Exception Helper is a pop-up that appears when an unhandled or thrown exception occurs. This has a new look and improved behavoir.

Demo:

1. Open GetOutstandingChargesForEmployee in ExpenseService.svc.cs.
2. Set the employeeId to a large number.
3. Run the app and get an exception.



Let’s summarize what’s new:

* The Exception Helper will appear whether you are debugging managed or unmanaged code.
* When an unhandled exception occurs, the entire line of code is highlighted. This improves code readability. An exception error icon will help you understand why the execution was interrupted.
* The Exception Helper pop-up is smaller, nonmodal, and less distractive.
* At a glance, the pop-up shows only the exception type, the error message, and whether the exception was thrown or unhandled.
* The pop-up immediately shows any inner exceptions—without the need of additional dialogs.
* In the Exception Settings group, you can specify if the debugger must break the execution when the exception is thrown, and you can exclude specific modules from breaking the execution.
* If you click the Open Exception Settings hyperlink, the exception information will be shown inside the QuickWatch dialog (see Figure 68). This makes it easy to investigate the exception details and to reevaluate an expression.
* You can click the Edit Conditions hyperlink to specify when the execution should be broken by including or excluding specific modules (see Figure 69). This is the only case in which you interact with a modal dialog from the Exception Helper.

With its new look and behavior, the Exception Helper improves productivity by providing all the information you need while letting you focus on the code.

# Live Unit Testing

We are very proud to introduce a new feature in Visual Studio 2017 called Live Unit Testing ! This feature will make it easy for you to maintain quality and test coverage during rapid development and take your productivity to a whole new level. Imagine you are fixing a bug in a code base which you may not be completely familiar with. With Live Unit Testing you can know right away, as you are making edits to fix the bug, that you did not break any other part of the system. Getting this feedback, in-situ, as you type will give you extra confidence, make you more productive and why not, even enjoying fixing bugs and writing unit tests!

Live Unit Testing automatically runs the impacted unit tests in the background as you edit code, and visualizes the results and code coverage live, in the editor, in real-time. In addition to giving feedback on the impact that your changes had on the existing tests, you also get immediate feedback on whether the new code you added is already covered by one or more existing tests. This will gently remind you to write unit tests as you are making bug fixes or adding features. You will be on your way to the promised land where there is no test debt in your code base!

Live Unit Testing is present in the Enterprise edition of Visual Studio 2017 and it’s available for C# and VB projects that target the .NET Framework. It uses VB and C# compilers to instrument the code at compile time. Next, it runs unit tests on the instrumented code to generate data which it analyzes to understand which tests are covering which lines of code. It then uses this data to run just those tests that were impacted by the given edit providing immediate feedback on the results in the editor itself. As more edits are made or more tests are added or removed, it continuously updates the data which is used to identify the impacted tests.

Live Unit Testing in Visual Studio works with three popular unit testing frameworks; namely, MSTest, xUnit and NUnit. For MSTest you will need MSTest.TestAdapter 1.1.4-preview and MSTest.TestFramework 1.0.5-preview (or higher versions).

Demo:

1. Open LiveUnitTestingDemo solution.
2. Notice the following code in Program.cs:

public static class MyMath

{

public static int Add(int value1, int value2) {

return value1 + value2;

}

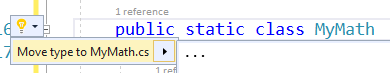
public static int Sub(int value1, int value2) {

return value1 - value2;

}

}

1. Notice the refactoring to move this into its own file.



1. Right-click MyMath and select Create Unit Tests. Keep the defaults.
2. Modify the AddTest code to the following:

public void AddTest() {

var result = MyMath.Add(2, 5);

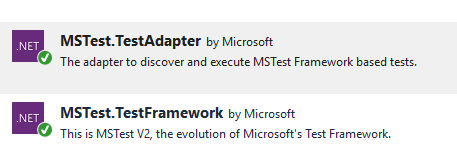
Assert.IsTrue(7 == result);

}

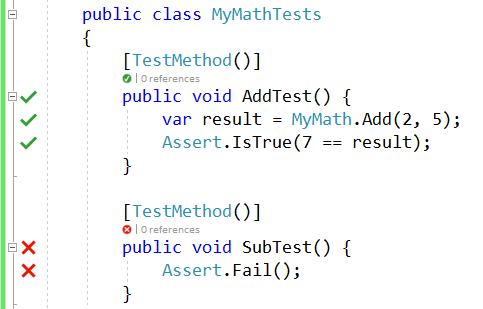
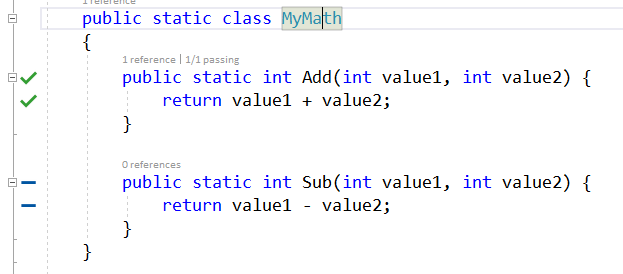
1. Build.
2. Test|Windows|Test Explorer.
3. You can run the tests from here. One passes and one doesn’t.
4. Select Test|Live Unit Testing|Start.
5. Change the Add code and break the test.

return value1 + value2 + 7;

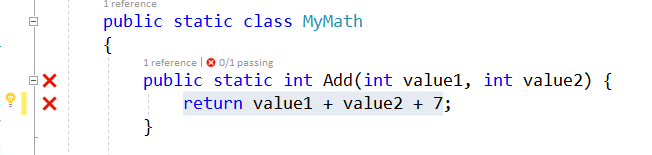
1. Nothing tells you that you just wrote broken code. You have to rerun the tests to know.
2. Change the code back and rerun the tests.
3. Add the 2 NuGet packages



1. Select Test|Live Unit Testing|Start.



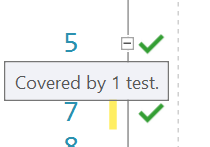
1. Change the Add code and break the test.



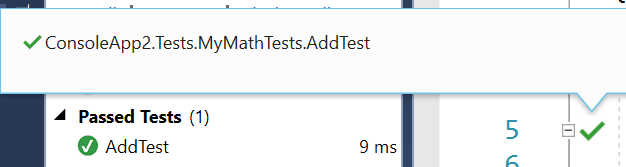
Note: Visual Studio 2017 can associate Live Unit Testing to your code only if unit tests have a reference to an object.

You can pause, stop and restart. You might pause if you are in the middle of writing a lot of code or refactoring and you know your tests will be broken for awhile.

Hover over the glyph to see how many tests are covered for each line of code.

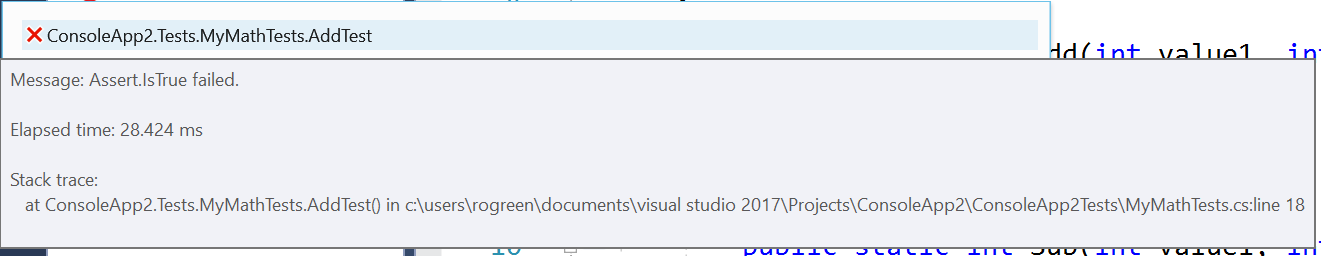


Click on the glyph to see the tests.

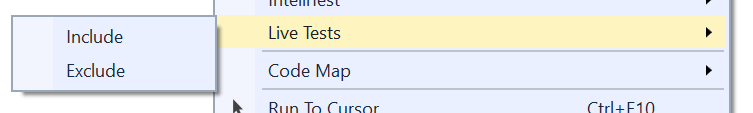


Click on a test to navigate to it.

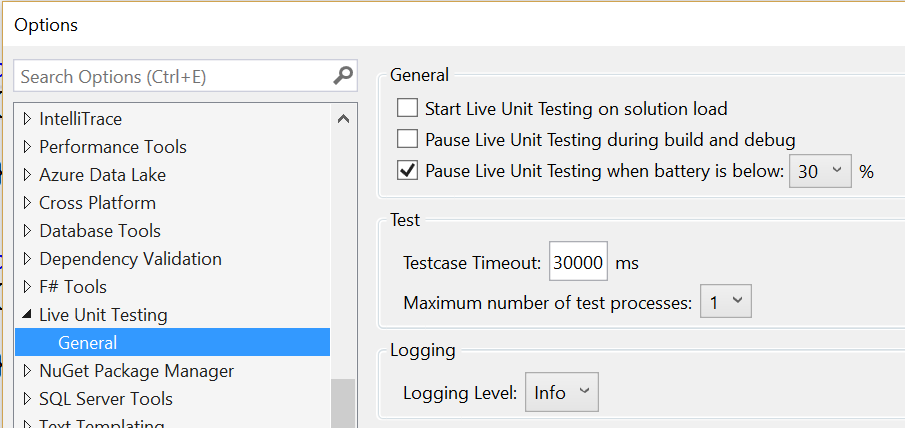
Hover over a failed test and you get more info on why the test failed.



Right-click in a method and you can include or exclude it from Live Unit Testing.



Configure this in the Options dialog.



C# 7.0

# Out variables

In older versions of C#, using out parameters isn’t as fluid as we’d like. Before you can call a method with out parameters you first have to declare variables to pass to it. Since you typically aren’t initializing these variables (they are going to be overwritten by the method after all), you also cannot use var to declare them, but need to specify the full type:

In C# 7.0 we have added *out variables*; the ability to declare a variable right at the point where it is passed as an out argument:

# Improvements to literals

C# 7.0 allows \_ to occur as a digit separator inside number literals:

var d = 123\_456;

var x = 0xAB\_CD\_EF;

You can put them wherever you want between digits, to improve readability. They have no effect on the value.

Also, C# 7.0 introduces binary literals, so that you can specify bit patterns directly instead of having to know hexadecimal notation by heart.

# Tuples

It is common to want to return more than one value from a method. The options available today are less than optimal. To do better at this, C# 7.0 adds tuple types and tuple literals.

Demo:

1. Start with the following code:

static void SimpleTuple()

{

int[] numbers = { 1, 2, 4, 8, 16, 32 };

var t = Tally(numbers);

}

static (int, int) Tally(int[] values)

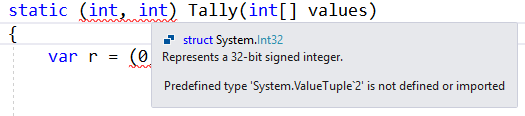
{

var r = (0, 0);

return r;

}

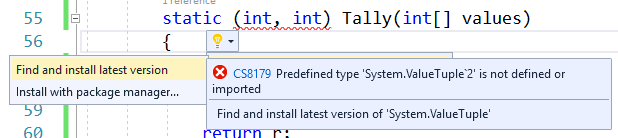
1. You get an error

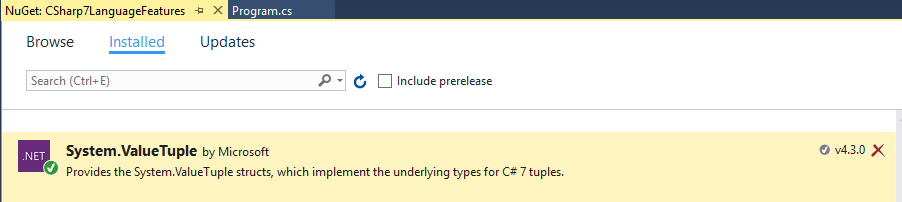


Tuples rely on a family of underlying generic struct types called ValueTuple<...>. If you target a Framework that doesn’t yet include those types, you can instead pick them up from NuGet.

You need the System.ValueTuple package. You can add it yourself or you can let VS add it for you.

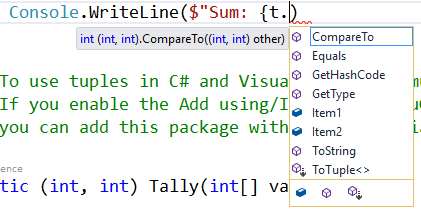
1. Go to Tools|Options|Text Editor|C#|Advanced and select Suggest usings for types in NuGet packages.





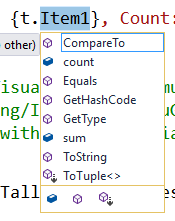
1. Add the following line. Notice you get IntelliSense on the items of the tuple.

Console.WriteLine($"Sum: {t.Item1}, Count: {t.Item2}");



1. Run the method.
2. You don’t need to use Item1, Item2, etc. You can give these names. Then you get IntelliSense. This is called a deconstruction declaration.

static (int sum, int count) Tally(int[] values)



1. You can also use names inside the method.

var r = (tupleSum:0, tupleCount:0);

1. You don’t need to store the tuple results in a variable. You can consume it directly.

var (sum, count)= Tally(numbers);

Console.WriteLine($"Sum: {sum}, Count: {count}");

1. Now implement the logic in the Tally method:

foreach (var v in values)

{

r = (r.tupleSum + v, r.tupleCount + 1);

}

1. Run the code.
2. Tuples are mutable.

r.tupleSum += v;

r.tupleCount ++;

# Local Functions

Sometimes a helper function only makes sense inside of a single method that uses it. You can now declare such functions inside other function bodies as a local function. Parameters and local variables from the enclosing scope are available inside of a local function, just as they are in lambda expressions.

Say I want to abstract out the way I calculate the sum and count into an Add method but I want to modify r directly. I can do that by declaring Add inside the Tally method. This is known as a local function.

# Pattern Matching

C# 7.0 introduces the notion of patterns, which, abstractly speaking, are syntactic elements that can test that a value has a certain “shape”, and extract information from the value when it does.

Examples of patterns in C# 7.0 are:

* Constant patterns of the form c (where c is a constant expression in C#), which test that the input is equal to c
* Type patterns of the form T x (where T is a type and x is an identifier), which test that the input has type T, and if so, extracts the value of the input into a fresh variable x of type T
* Var patterns of the form var x (where x is an identifier), which always match, and simply put the value of the input into a fresh variable x with the same type as the input.

In C# 7.0 we are enhancing two existing language constructs with patterns:

* is expressions can now have a pattern on the right hand side, instead of just a type
* case clauses in switch statements can now match on patterns, not just constant values

There are several things to note about this newly extended switch statement:

* The order of case clauses now matters: Just like catch clauses, the case clauses are no longer necessarily disjoint, and the first one that matches gets picked. It’s therefore important that the square case comes before the rectangle case above. Also, just like with catch clauses, the compiler will help you by flagging obvious cases that can never be reached. Before this you couldn’t ever tell the order of evaluation, so this is not a breaking change of behavior.
* The default clause is always evaluated last: Even though the null case above comes last, it will be checked before the default clause is picked. This is for compatibility with existing switch semantics. However, good practice would usually have you put the default clause at the end.
* The null clause at the end is not unreachable: This is because type patterns follow the example of the current is expression and do not match null. This ensures that null values aren’t accidentally snapped up by whichever type pattern happens to come first; you have to be more explicit about how to handle them (or leave them for the default clause).

XAML

These apply to both WPF and UWP, but not Xamarin.Forms.

# Edit and Continue

VS 2015 had the ability to modify XAML and see the changes while the app was running, but once you closed the app, your changes were all gone. In VS 2017, you can edit XAML while the application is running. No need to stop the app, make changes and run again.

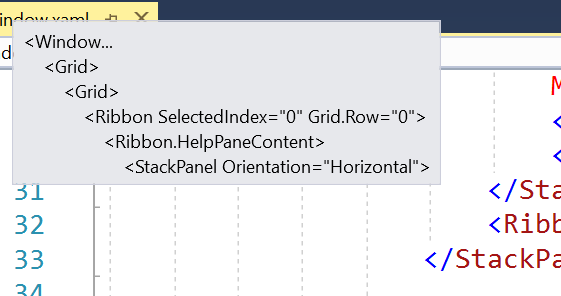
Demo:

1. Run the Expenses app.
2. Open MainWindow.xaml.
3. Find the RibbonButton with the Label of View Outstanding.
4. Change it to View Outstanding Charges.
5. Change it to View Charges.
6. Undo your changes.
7. Change the ImageSource to NewCharge.
8. Set it back.
9. Change the RibbonTab Header to Charges.

You are not limited to editing controls’ or panels’ property values—you can also add new controls, new panels, and everything you might need to improve your user interface. You can combine using this feature with the Live Visual Tree and Live Property Explorer windows to get an enhanced experience.

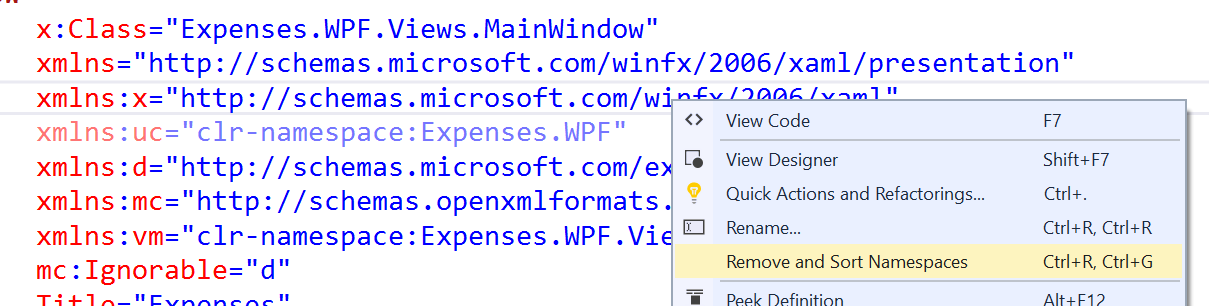
Other improvements

XAML now has Style Guidelines

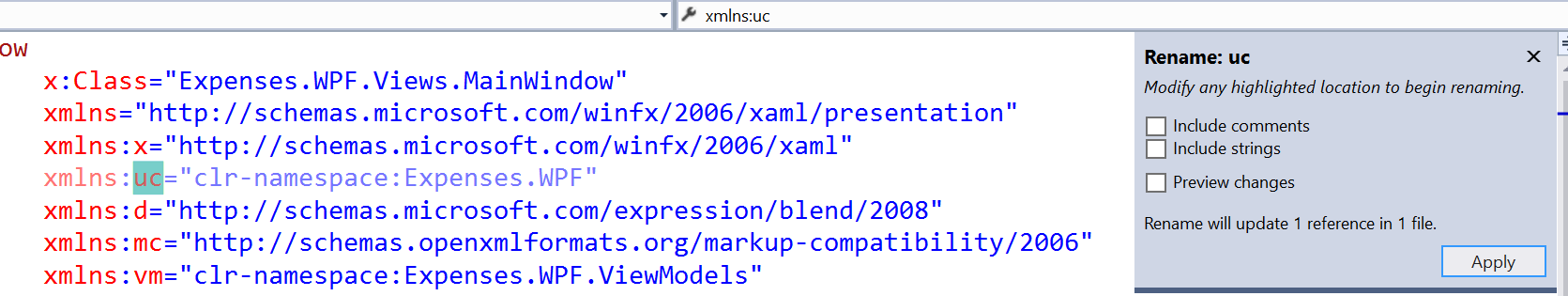


IntelliSense Filtering

Remove and Sort namespaces



Inline namespace rename



.NET Core

* Create .NET Core Web app project. Notice ability to enable Docker support.
* Right-click on project and Edit .csproj
* Add a NuGet package and see it gets added to csproj
* Delete from csproj and it gets deleted from project
* Add back to csproj and it gets added back to project
* Open project folder in File Explorer
* Create new .cs file
* It auto gets added to the project

Explore Service Capabilities. Right-click that node and Add Service Capability. This is update over Add Connected Service option that was in VS 2015.

Extensions

Not that VSIX extensions now get installed after you exit VS, rather than during and then telling you to restart.

Demo: You can use the GitHub extension in Community edition or install a new extension.

Visual Studio Preview

<https://www.visualstudio.com/vs/preview/>

