Getting Started with Visual Studio

Visual Studio (VS) is the most advanced integrated development environment (IDE) available today. It is primarily used for Windows development but it also supports development of iOS, Linux and Android apps. It can be overwhelming to new users. This document provides some general help for getting around in Visual Studio, working with code and using the debugger. It is based upon Visual Studio 2015 but the suggestions should be useful in other versions.

Contents

[Getting Around 2](#_Toc471559718)

[Editor Window 2](#_Toc471559719)

[Output Window 3](#_Toc471559720)

[Error Window 3](#_Toc471559721)

[Solutions and Projects 3](#_Toc471559722)

[Projects 3](#_Toc471559723)

[Solutions 4](#_Toc471559724)

[Solution Explorer 4](#_Toc471559725)

[Startup Project 4](#_Toc471559726)

[Working with Files 4](#_Toc471559727)

[Build and Running 5](#_Toc471559728)

[Build vs Rebuild 5](#_Toc471559729)

[Build vs Debug 5](#_Toc471559730)

[Debugging 6](#_Toc471559731)

[Edit and Continue 6](#_Toc471559732)

[Controlling the Debugger 6](#_Toc471559733)

[Breakpoints 6](#_Toc471559734)

[Stepping Through Code 7](#_Toc471559735)

[Current Statement 7](#_Toc471559736)

# Getting Around

VS has a lot of UI elements. It is important to understand some basic terminology.

* Menu – The menu at the top of the screen providing access to the core functionality.
* Editor Window – The tabbed window in the middle of the main window. All open files are represented here. This is also the window where editing files will occur.
* Tool Windows – The various windows along the top, sides and bottom of the main window. They provide access to functionality outside the main editor window.
* Toolbars – The bars across the top of the main window that provide quick access to common commands.
* Dialogs – Windows that appear in response to certain events. These windows generally alert you to some immediate action and must be closed before you may continue.
* Notifications – The area in the top right of the menu bar where notifications will appear related to updates.

VS is highly customizable. You can adjust VS to fit your particular coding style. Some examples of what you can do.

* Toolbars
  + Rearrange, show and hide each toolbar.
  + Create new toolbars and customize the buttons available on them.
* Tool Windows
  + Show or hide tool windows.
  + Dock to any side of the main window.
  + Detach and float anywhere on the screen.
  + Auto-hide (unpin) so the window appears only when it has focus.
  + Combine tool windows into tabbed windows to allow grouping.
* Editor Window
  + Control order of tabs.
  + Pin tabs so they are separate from non-pinned tabs.
  + Detach and float the tab (or set of tabs) in their own window.

VS supports two different visual layouts: design and debug. In design mode the layout of the windows, available toolbars and other features of the UI are optimized for writing code. When debugging your code the layout changes to provide access to UI elements that are relevant for debugging. In many cases you can adjust the layout to fit your needs for debugging. When debugging stops VS will switch back to the design layout.

### Editor Window

The Editor window is where you will be writing code. Some useful information about this window.

1. Each tab represents a different file.
2. The files may or may not be part of your project so pay careful attention to the files you are editing.
3. You can quickly go to the directory containing the current document by right clicking the tab and selecting *Open Containing Folder*.
4. If the contents of a document have been changed but not saved then the tab puts an indicator in the title. Documents are automatically saved
5. All documents in the project are automatically saved when you build.

### Output Window

The Output window is where the IDE puts most messages of importance including build messages, debug messages and general errors. The window separates the messages by category so use the dropdown list in the window to select the category of messages you want to see.

Perhaps the most useful aspect of this window is during builds when errors occur. Each error will generate a well-defined error message. The error message can be double clicked to jump to the appropriate location in your source code. More importantly, errors are shown in the order they are generated so focus on resolving the errors appearing earlier in the window to avoid wasting time on later errors that are caused by earlier ones.

### Error Window

The Error List window appears when you do a build and errors occur. Unlike the Output window errors are shown in an undefined order. Double clicking a row in the window will jump to the appropriate location in your source code.

# Solutions and Projects

### Projects

A project is a set of files and related resources that ultimately can be packaged into a single binary (i.e. an executable, library or DLL). Projects in VS have functionality that VS can call upon including: build, debug, publish and deploy.

The following are some of the many types of projects supported by VS.

* Console applications – Useful for creating utilities and simple UI applications.
* Windows applications – Useful for creating complex UIs and programs for non-experienced users.
* Class libraries – Used to create reusable pieces of code.
* Databases – Used to create databases, reports and other business intelligence solutions.
* Websites – Used to create websites.
* Mobile applications – Used to create mobile applications for Android and iOS

One of the features that makes VS so useful is that the available projects can be extended by third parties to include other types of applications.

Each project is stored in the file system under its own folder. Multiple projects cannot share the same folder. Within the project’s folder are the files that make up the project and any output from building the project.

As projects get bigger it often becomes useful to break them up into a series of separate projects. This simplifies development and helps support reuse. Unfortunately this adds complexity as each project may be in a different state (ex. Building, debugging, etc). To simplify the UI, all projects must be contained in a solution.

### Solutions

A solution is a collection of projects. A solution, by itself, does nothing and produces nothing. It is simply a useful grouping of projects. For example, instead of building each project you can simply build the solution and all projects within it are built automatically. Even in a single project environment, a solution is still present. On disk the solution is represented as a folder with a single .SLN file in it. Projects contained within the solution appear as subfolders of the solution.

NOTE: Folder structures are critical to VS. Do not every change the folder structure of a solution or any of its projects. Doing so will cause serious problems. When copying projects from one machine to another it is mandatory that you copy the entire solution folder and not just the projects and/or files you’re using.

Projects within a solution are still independent of each other. Setting up dependencies between projects requires additional steps that depend upon the project type and language being used.

In most cases your interactions with solutions and projects will occur in Solution Explorer.

### Solution Explorer

When you are in Solution Explorer you should notice a top level node that simply contains a name (Out of the box this may not be visible so go to *Tools\Options -> Projects and Solutions\General* -> *Always show solution)*. This is the solution. Any functionality exposed here impacts all the projects. Under the solution is a node for each project. The project name should match what you enter when you created it. Any functionality performed on the project node only impacts that node.

Solution Explorer is a virtual view of the file system. A solution is the root directory where your solution file resides. You can get to this folder quickly by right clicking the solution node and selecting *Open Folder in File Explorer*. Each project has its own subfolder under the solution. You can quickly get to that folder by using the same command but on the project node.

### Startup Project

Because a solution may have any number of projects they are all considered equal. In the case of debugging, it is necessary to identify the project that should be started. This is known as the startup project.

Whenever you start the debugger it needs to know which project(s) to run. The debugger can only run executable projects (i.e. console applications). By default it will run a single project known as the startup project. The startup project is generally the last project added to the solution that is executable.

The startup project is highlighted in Solution Explorer. To change the startup project (you cannot be debugging at the time) then right click the appropriate project and select *Set as Startup Project*.

### Working with Files

Projects are made up of files but not every file you open in VS may be part of a project. Unfortunately there is no way to tell if an open file is part of the project or not. Changing a file that is not part of a project will have no impact on the project itself.

To help avoid issues with files the following is strongly recommended.

1. Never open a file or project directly, always either double click the .SLN file in the file system or open Visual Studio and use the Start page to open the recent solution.
2. Do not work with the files or projects directly inside the file system. Always use Solution Explorer to open, edit, and rename them.
3. To open a file in a project double click it in Solution Explorer.
4. If you are unsure whether an open file is in the project then double click the file in Solution Explorer. If it is the correct file then you will return to the same editor window. If it is not then a new window will open.

# Build and Running

Building your code is the process of converting it from source code to runnable code. This requires the compiler to run. Any changes to any files will be saved prior to the build running. If you are currently debugging then the debugger must be stopped as well.

You can build a single file or project but you’ll almost always choose Build Solution from the menu or toolbar. This command will rebuild any changes that have occurred. Errors will appear in the Error List and Output windows. If any errors occur then the build will fail and no binary will be created.

NOTE: When building a project it is possible for the build to fail because of a locked file. This most often happens when a solution is stored on a network or removable drive. If a file is locked then a compiler error occurs. In this case the only solution is to restart VS.

### Build vs Rebuild

There are two similar commands: Build and Rebuild. Build only builds files and projects that have changed. As such it is the quickest option with the least amount of IO. This is the default choice you should make.

Rebuild will recompile everything whether it has changed or not. This is useful in cases where it appears something has gotten corrupted or you made significant changes to the project structure or contents. This is basically a test build to ensure everything compiles. It should be used rarely.

### Build vs Debug

Building is the process of converting your source code to binary code. Debugging is the process of running your code in the debugger. Build must occur before debug. If the build fails for any reason then debugging does not make sense. Since you must build before you debug the IDE will automatically build your code if it is out of date when you select to debug. This is a time saver but can introduce unexpected issues if you are not careful.

If the build fails then VS will display a message asking if you want to continue anyway. The correct answer is ALWAYS no. However there is an option to continue anyway in which case it will run the previously built version, if any. Even more frustrating though is the option to remember your selection. If you select this box then you will no longer get the dialog if the build fails and it will always debug the previous version. This can lead to confusion as the code does not match what you’re debugging.

It is strongly recommended that you always build explicitly first and then debug your code. There is no difference in speed for this and avoids you selecting the wrong option.

# Debugging

Debugging is the process of running your code in the debugger sandbox to identify issues. The debugger is a very special process that has complete access to your code and can do things that normally are not allowed. As such the debugger can identify issues in your code that may not be visible outside the debugger. You should ALWAYS run your code with the debugger. Do not use the Start without Debugging option.

Running under the debugger will cause any runtime errors to pause your program and jump you to the line of code that is causing the issue. This is incredibly useful for finding issues. However the debugger does modify your code’s behavior so do not assume that the behavior you see in the debugger is the same behavior an end user would see.

### Edit and Continue

Normally, while debugging your code, you cannot edit it. To edit code you would need to stop the debugger. VS supports the concept of edit and continue. This allows you to make changes to your code while stopped in the debugger. When the debugger continues the changes are compiled and applied to the debugging code. The debugger will prompt you whether you want to use this feature or stop, recompile and then run your program again. This is designed to save time for code that requires significant setup time. But it doesn’t work for all changes and may break your code. Use it with caution.

### Controlling the Debugger

The debugger is a complicated program but you do have some control over its behavior. To start debugging you’ll use the Start or Debug command. This begins executing your startup project within the debugger. Until a debug command is reached then your program will execute normally. The following are some common commands.

* Break
* Unhandled error
* Step
* Pause
* Stop

### Breakpoints

While executing code it is often useful to stop your program so you can view the state of variables and other information. While you can use logging statements this is not how modern debugging is done. Instead you may insert breakpoints into your code. A breakpoint is an indicator to the debugger that when a line is reached, pause the program and allow you to view what is going on.

A program can have any number of breakpoints. Breakpoints only impact the debugger and are not part of the compiled code. You can toggle a breakpoint by selecting a line of code and pressing F9 (default) or by clicking in the left margin of the editor. If you are not currently debugging then a red symbol appears on the line. If you are debugging already (or when you start) the symbol may move if it is not on an executable line.

In general you will “set” a breakpoint on a line of cod where you want to view the state of the system. Once the breakpoint is “hit” you will analyze the variables and state to ensure everything is working as you expected. You will then continue the debugger. You can add/remove breakpoints as needed.

One word of note about breakpoints. The debugger breaks into your code before the line it is applied to executes. This is your chance to verify state before the line executes.

### Stepping Through Code

Once you hit a breakpoint in your code it is often useful to evaluate the next set of lines one by one. You can do that by putting a breakpoint on each line but the preferred approach is to step through your code.

Stepping is like setting a breakpoint on the next line, continuing the program until it hits the breakpoint and then removing it. The advantage is that the debugger does this for you. You can use stepping to walk through your code line by line and break into the debugger each time. Like breakpoints, the debugger stops before the line is executed.

Stepping is probably the most commonly used debugger command. There are 3 different versions of the command that you can use.

* Step Over – This command executes the current line and moves to the next one. If there are any functions or expressions on the line then it executes them automatically. This is the command that you will use most often.
* Step Into – This command behaves like Step Over except when on a function call. For function calls this command will step into the function being called. This command is used when you need to step into a function being called to verify its state. If there are no function calls on the current line then it behaves like Step Over.
* Step Out – This command is useful only when inside a function you wrote. It causes the debugger to run the remaining statements in the function and then stop on the line that called this function. This command is useful for when you accidentally stepped into a function you didn’t want. It quickly gets you back out.

### Current Statement

In modern programs there can be only one logic command running at a time (ignoring threading). The debugger indicates the line to be executed next by a yellow arrow in the left margin. Unless you are stopped at a breakpoint you won’t see this because the code runs to fast. Use the arrow to identify which statement will be executed when the code continues running. If you cannot find what the next line is then right click the editor and select the Show Next Statement command to jump to it.

An advanced debugging command is to change the line that is being executed. You can left-click drag the yellow arrow up or down to change what the next statement is. Care must be taken to ensure you don’t invalidate your program’s behavior. Additionally the debugger will not allow any changes that violate the core language requirements (such as jumping out of a function). This command is most useful for repeating a line of code that didn’t behave the way you expected the first time.