Debugging concepts

The debugger

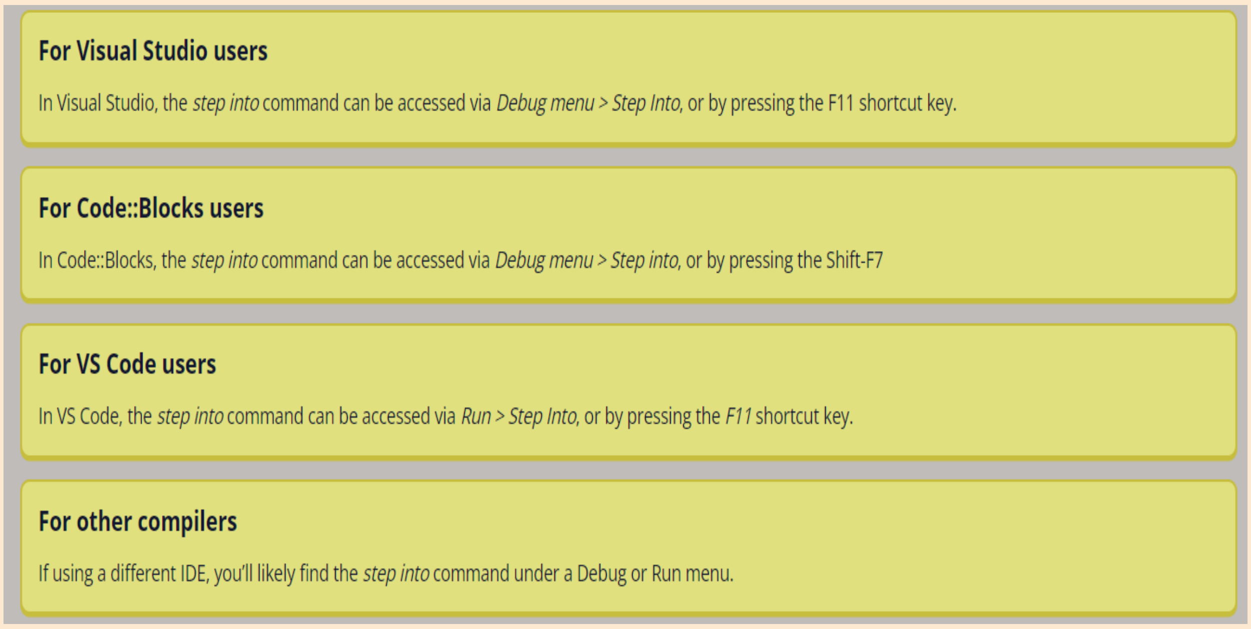
A debugger is a computer program that allows the programmer to control how another program executes and examine the program state while that program is running.

Stepping

Stepping is the name for a set of related debugger features that let us execute (step through) our code statement by statement.

Step into

The step into command executes the next statement in the normal execution path of the program, and then pauses execution of the program so we can examine the program's state using the debugger. If the statement being executed contains a function call, step into causes the program to jump to the top of the function being called, where it will pause.



For Visual Studio users

In Visual Studio, the *step into* command can be accessed via *Debug menu > Step Into*, or by pressing the F11 shortcut key.

For Code::Blocks users

In Code::Blocks, the *step into* command can be accessed via *Debug menu > Step into*, or by pressing the Shift-F7

For VS Code users

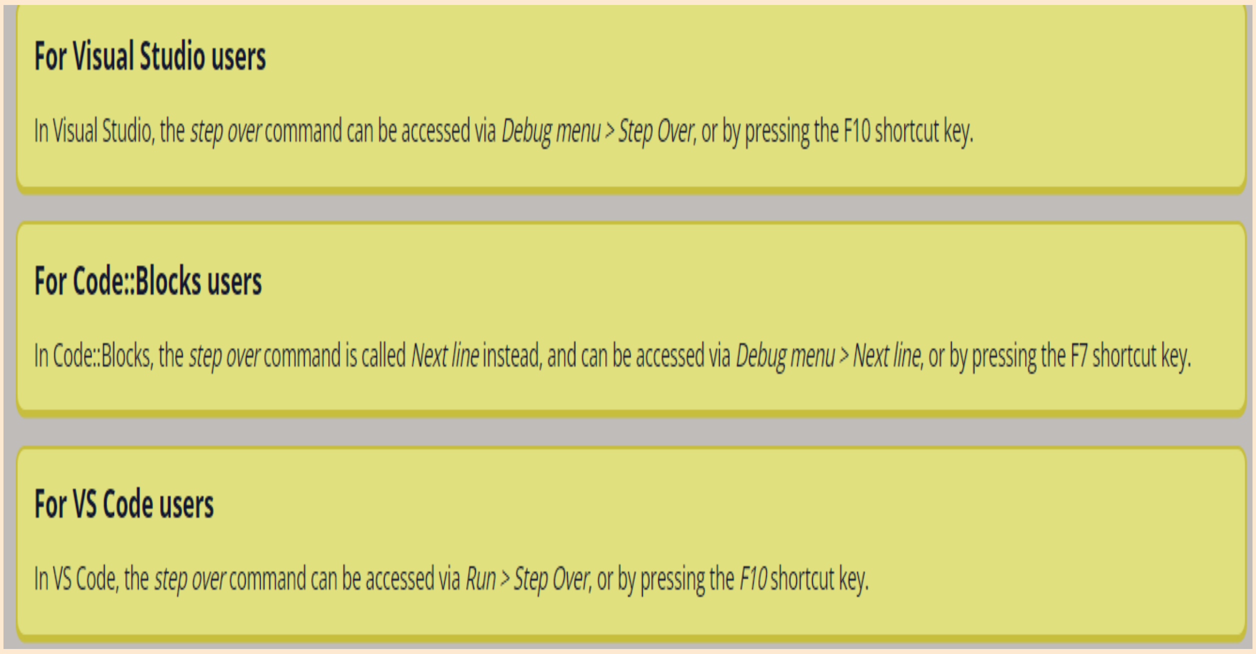
In VS Code, the *step into* command can be accessed via *Run > Step Into*, or by pressing the *F11* shortcut key.

For other compilers

If using a different IDE, you'll likely find the *step into* command under a Debug or Run menu.

Step over

Like step into, The step over command executes the next statement in the normal execution path of the program. However, whereas step into will enter function calls and execute them line by line, step over will execute an entire function without stopping and return control to you after the function has been executed.



For Visual Studio users

In Visual Studio, the *step over* command can be accessed via *Debug menu > Step Over*, or by pressing the F10 shortcut key.

For Code::Blocks users

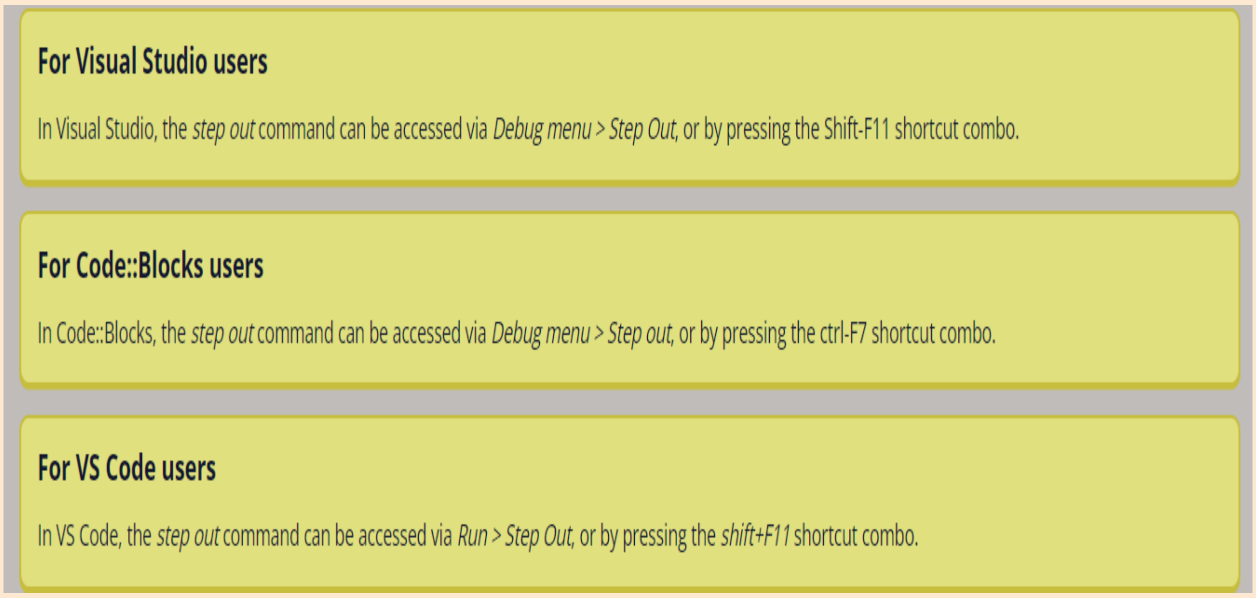
In Code::Blocks, the *step over* command is called *Next line* instead, and can be accessed via *Debug menu > Next line*, or by pressing the F7 shortcut key.

For VS Code users

In VS Code, the *step over* command can be accessed via *Run > Step Over*, or by pressing the *F10* shortcut key.

Step out

Unlike the other two stepping commands, Step out does not just execute the next line of code. Instead, it executes all remaining code in the function currently being executed, and then returns control to you when the function has returned.



For Visual Studio users

In Visual Studio, the *step out* command can be accessed via *Debug menu > Step Out*, or by pressing the Shift-F11 shortcut combo.

For Code::Blocks users

In Code::Blocks, the *step out* command can be accessed via *Debug menu > Step out*, or by pressing the ctrl-F7 shortcut combo.

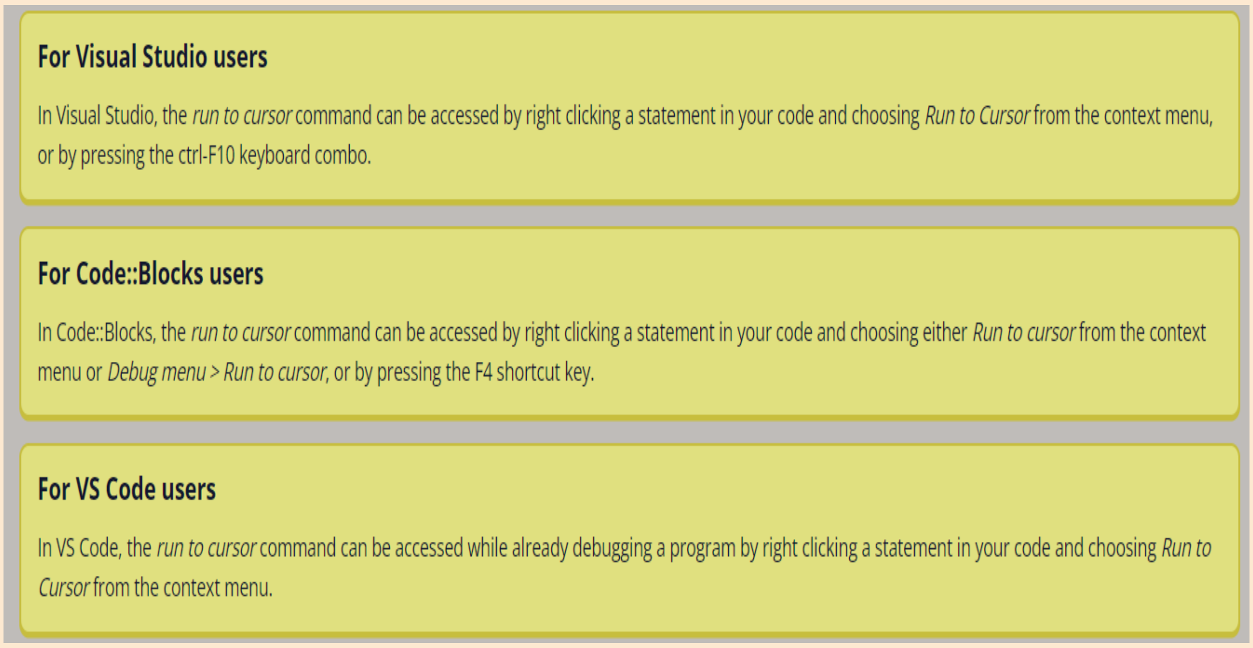
For VS Code users

In VS Code, the *step out* command can be accessed via *Run > Step Out*, or by pressing the *shift+F11* shortcut combo.

Running and breakpoints

Run to cursor

The first useful command is commonly called Run to cursor. This Run to cursor command executes the program until execution reaches the statement selected by your cursor. Then it returns control to you so you can debug starting at that point. This makes for an efficient way to start debugging at a particular point in your code, or if already debugging, to move straight to some place you want to examine further.



For Visual Studio users

In Visual Studio, the *run to cursor* command can be accessed by right clicking a statement in your code and choosing *Run to Cursor* from the context menu, or by pressing the ctrl-F10 keyboard combo.

For Code::Blocks users

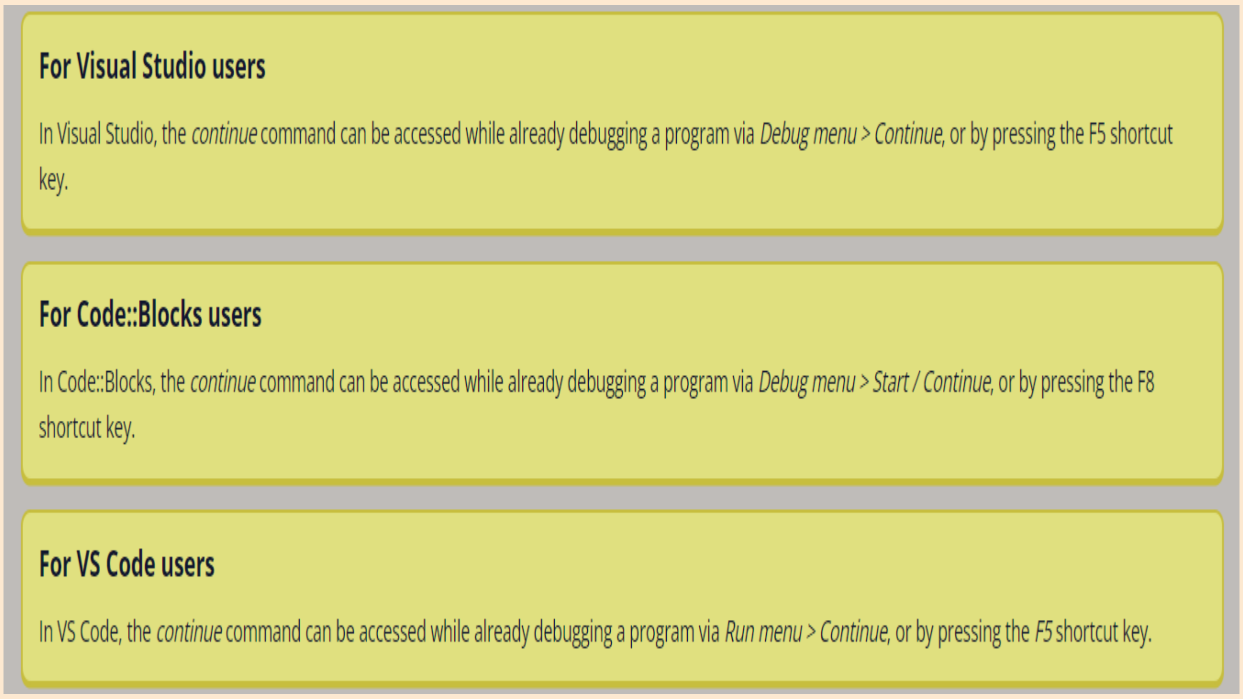
In Code::Blocks, the *run to cursor* command can be accessed by right clicking a statement in your code and choosing either *Run to cursor* from the context menu or *Debug menu > Run to cursor*, or by pressing the F4 shortcut key.

For VS Code users

In VS Code, the *run to cursor* command can be accessed while already debugging a program by right clicking a statement in your code and choosing *Run to Cursor* from the context menu.

Continue

The continue debug command simply continues running the program as per normal, either until the program terminates, or until something triggers control to return back to you again such as a breakpoint.



For Visual Studio users

In Visual Studio, the *continue* command can be accessed while already debugging a program via *Debug menu > Continue*, or by pressing the F5 shortcut key.

For Code::Blocks users

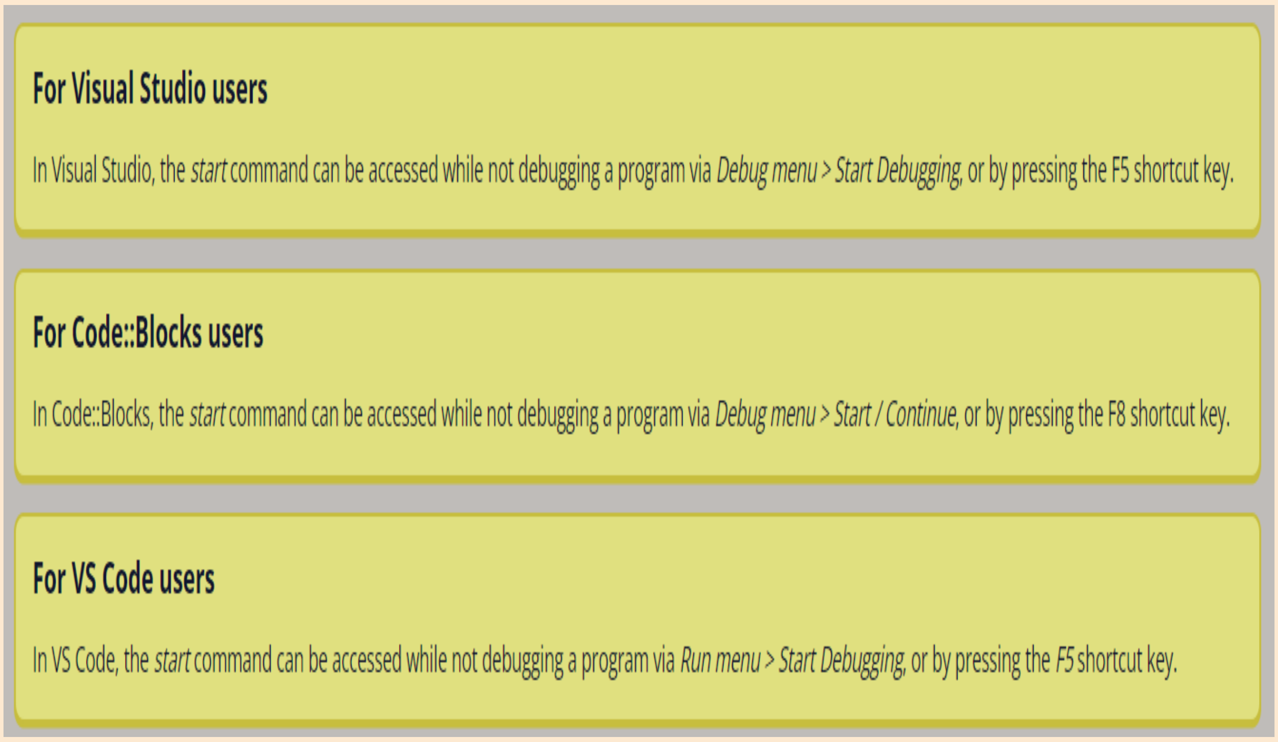
In Code::Blocks, the *continue* command can be accessed while already debugging a program via *Debug menu > Start / Continue*, or by pressing the F8 shortcut key.

For VS Code users

In VS Code, the *continue* command can be accessed while already debugging a program via *Run menu > Continue*, or by pressing the *F5* shortcut key.

Start

The continue command has a twin brother named start. The start command performs the same action as continue, just starting from the beginning of the program. It can only be invoked when not already in a debug session.



For Visual Studio users

In Visual Studio, the *start* command can be accessed while not debugging a program via *Debug menu > Start Debugging*, or by pressing the F5 shortcut key.

For Code::Blocks users

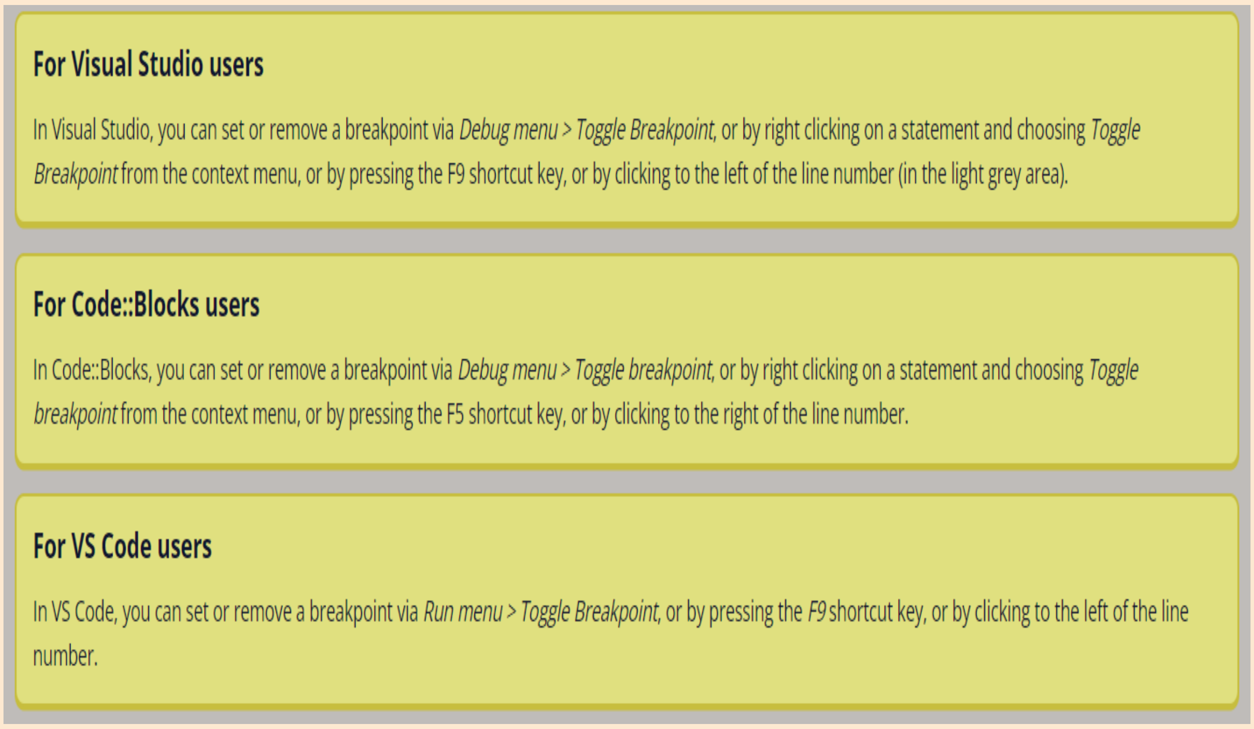
In Code::Blocks, the *start* command can be accessed while not debugging a program via *Debug menu > Start / Continue*, or by pressing the F8 shortcut key.

For VS Code users

In VS Code, the *start* command can be accessed while not debugging a program via *Run menu > Start Debugging*, or by pressing the *F5* shortcut key.

Breakpoints

A breakpoint is a special marker that tells the debugger to stop execution of the program at the breakpoint when running in debug mode.



For Visual Studio users

In Visual Studio, you can set or remove a breakpoint via *Debug menu > Toggle Breakpoint*, or by right clicking on a statement and choosing *Toggle Breakpoint* from the context menu, or by pressing the F9 shortcut key, or by clicking to the left of the line number (in the light grey area).

For Code::Blocks users

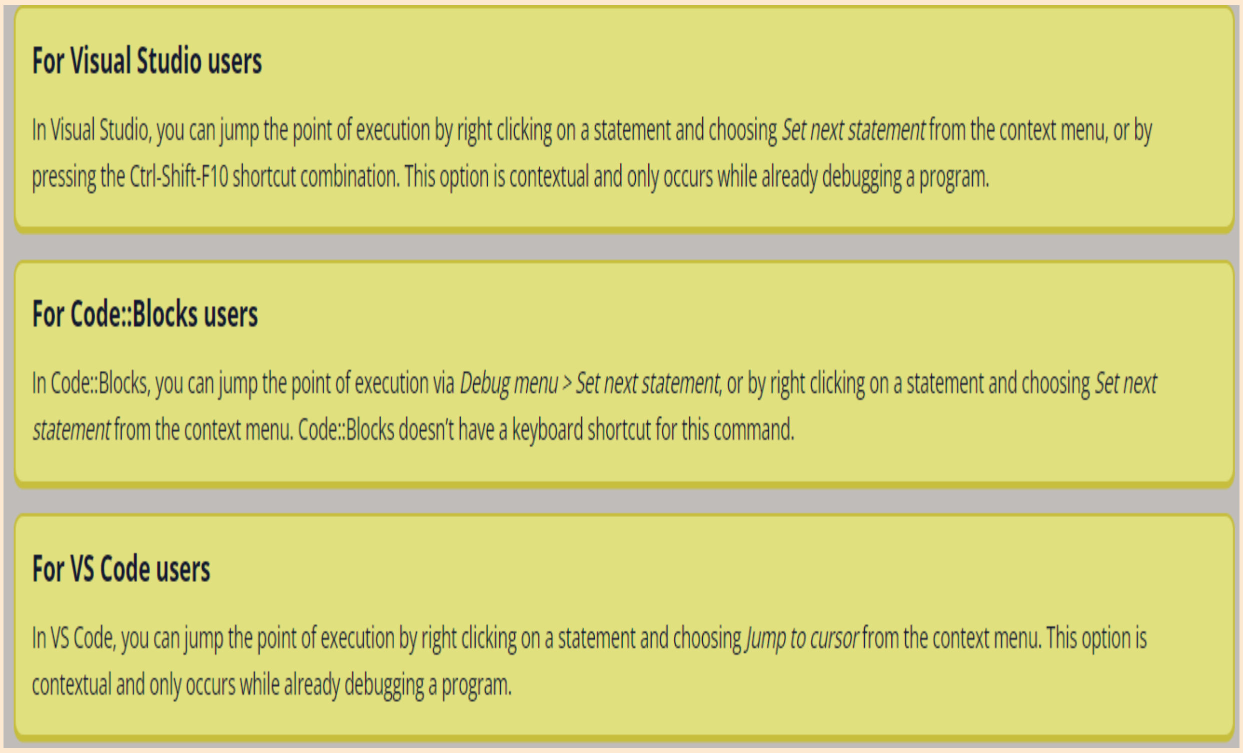
In Code::Blocks, you can set or remove a breakpoint via *Debug menu > Toggle breakpoint*, or by right clicking on a statement and choosing *Toggle breakpoint* from the context menu, or by pressing the F5 shortcut key, or by clicking to the right of the line number.

For VS Code users

In VS Code, you can set or remove a breakpoint via *Run menu > Toggle Breakpoint*, or by pressing the *F9* shortcut key, or by clicking to the left of the line number.

Set next statement

The set next statement command allows us to change the point of execution to some other statement (sometimes informally called jumping). This can be used to jump the point of execution forwards and skip some code that would otherwise execute, or backwards and have something that already executed run again.



For Visual Studio users

In Visual Studio, you can jump the point of execution by right clicking on a statement and choosing *Set next statement* from the context menu, or by pressing the Ctrl-Shift-F10 shortcut combination. This option is contextual and only occurs while already debugging a program.

For Code::Blocks users

In Code::Blocks, you can jump the point of execution via *Debug menu > Set next statement*, or by right clicking on a statement and choosing *Set next statement* from the context menu. Code::Blocks doesn't have a keyboard shortcut for this command.

For VS Code users

In VS Code, you can jump the point of execution by right clicking on a statement and choosing *Jump to cursor* from the context menu. This option is contextual and only occurs while already debugging a program.

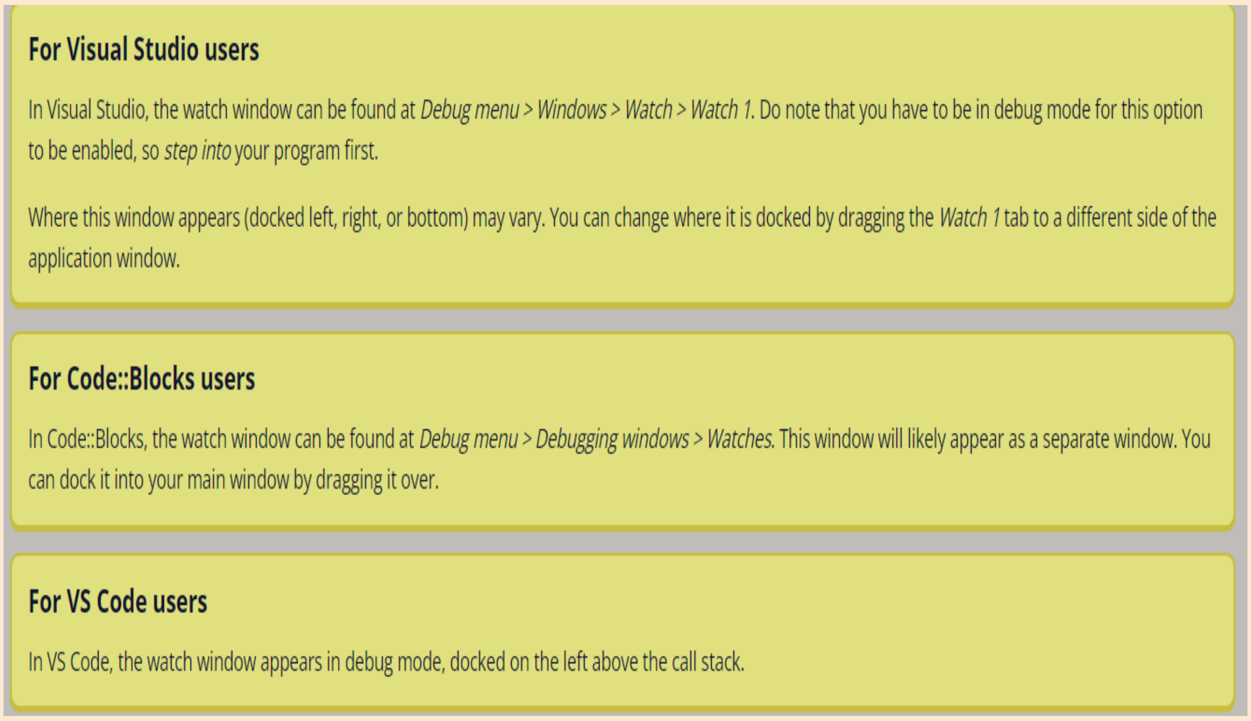
Watching variables

Watching variables

Watching a variable is the process of inspecting the value of a variable while the program is executing in debug mode.

The watch window

The watch window is a window where you can add variables you would like to continually inspect, and these variables will be updated as you step through your program.



For Visual Studio users

In Visual Studio, the watch window can be found at *Debug menu > Windows > Watch > Watch 1*. Do note that you have to be in debug mode for this option to be enabled, so *step into* your program first.

Where this window appears (docked left, right, or bottom) may vary. You can change where it is docked by dragging the *Watch 1* tab to a different side of the application window.

For Code::Blocks users

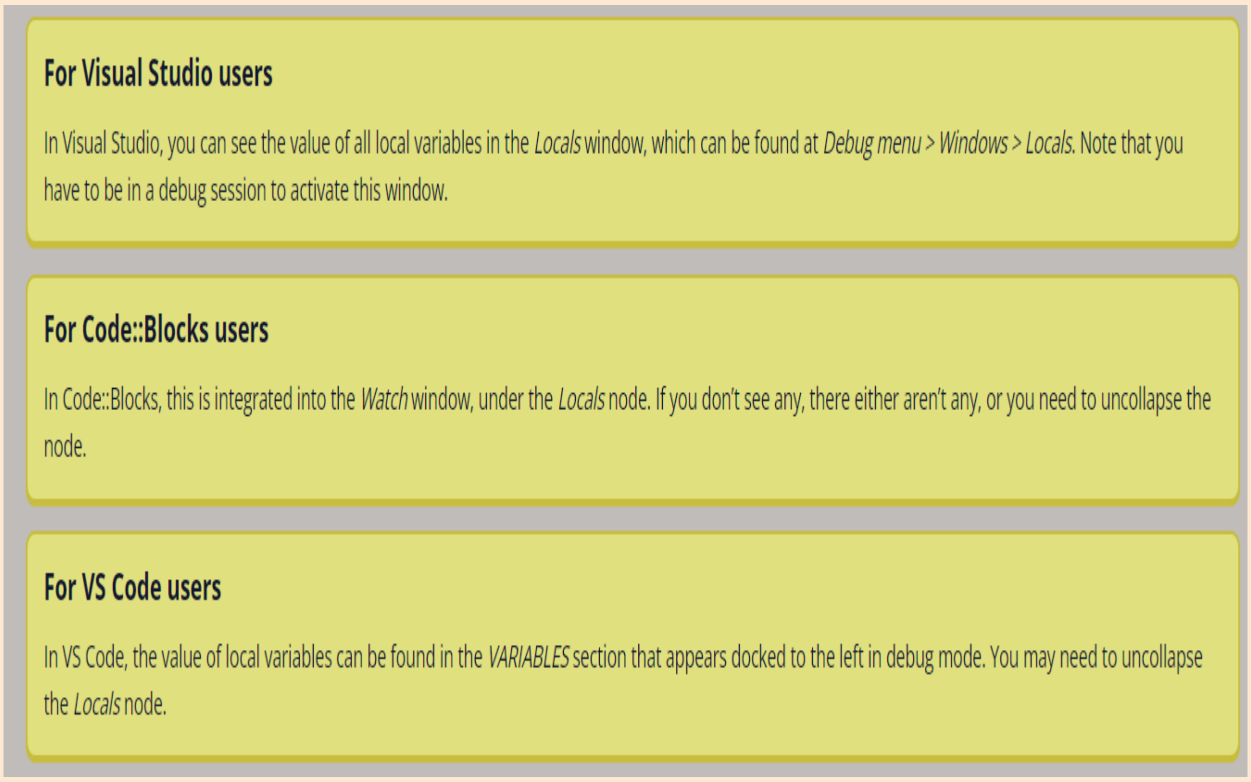
In Code::Blocks, the watch window can be found at *Debug menu > Debugging windows > Watches*. This window will likely appear as a separate window. You can dock it into your main window by dragging it over.

For VS Code users

In VS Code, the watch window appears in debug mode, docked on the left above the call stack.

Local watches

Because inspecting the value of local variables inside a function is common while debugging, many debuggers will offer some way to quickly watch the value of all local variables in scope.



For Visual Studio users

In Visual Studio, you can see the value of all local variables in the *Locals* window, which can be found at *Debug menu > Windows > Locals*. Note that you have to be in a debug session to activate this window.

For Code::Blocks users

In Code::Blocks, this is integrated into the *Watch* window, under the *Locals* node. If you don't see any, there either aren't any, or you need to uncollapse the node.

For VS Code users

In VS Code, the value of local variables can be found in the *VARIABLES* section that appears docked to the left in debug mode. You may need to uncollapse the *Locals* node.

The call stack

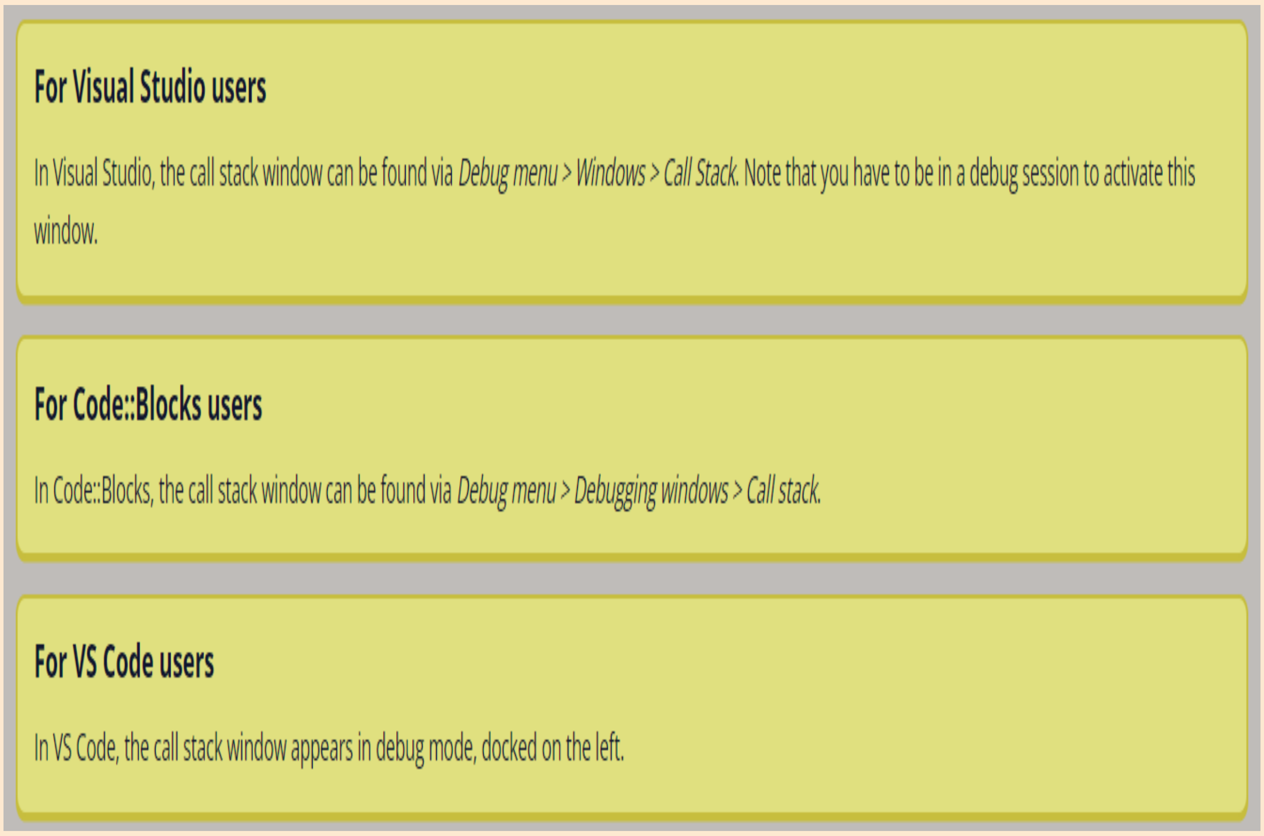
The call stack

The call stack is a list of all the active functions that have been called to get to the current point of execution. The call stack includes an entry for each function called, as well as which line of code will be returned to when the function returns.

Whenever a new function is called, that function is added to the top of the call stack. When the current function returns to the caller, it is removed from the top of the call stack, and control returns to the function just below it.

Call stack window

The call stack window is a debugger window that shows the current call stack.



For Visual Studio users

In Visual Studio, the call stack window can be found via *Debug menu > Windows > Call Stack*. Note that you have to be in a debug session to activate this window.

For Code::Blocks users

In Code::Blocks, the call stack window can be found via *Debug menu > Debugging windows > Call stack*.

For VS Code users

In VS Code, the call stack window appears in debug mode, docked on the left.

References:

1. Google.com
2. Learn C++