

JavaScript Debugging Reference



By Kayce Basques

Technical Writer for Chrome DevTools

Discover new debugging workflows with this comprehensive reference of Chrome DevTools debugging features.

See [Get Started With Debugging JavaScript In Chrome DevTools](#) to learn the basics of debugging.

Pause code with breakpoints


Set a breakpoint so that you can pause your code in the middle of its execution.

See [Pause Your Code With Breakpoints](#) to learn how to set breakpoints.

Step through code

Once your code is paused, step through it, one line at a time, investigating control flow and property values along the way.

Step over line of code

When paused on a line of code containing a function that's not relevant to the problem you're debugging, click **Step over**  to execute the function without stepping into it.

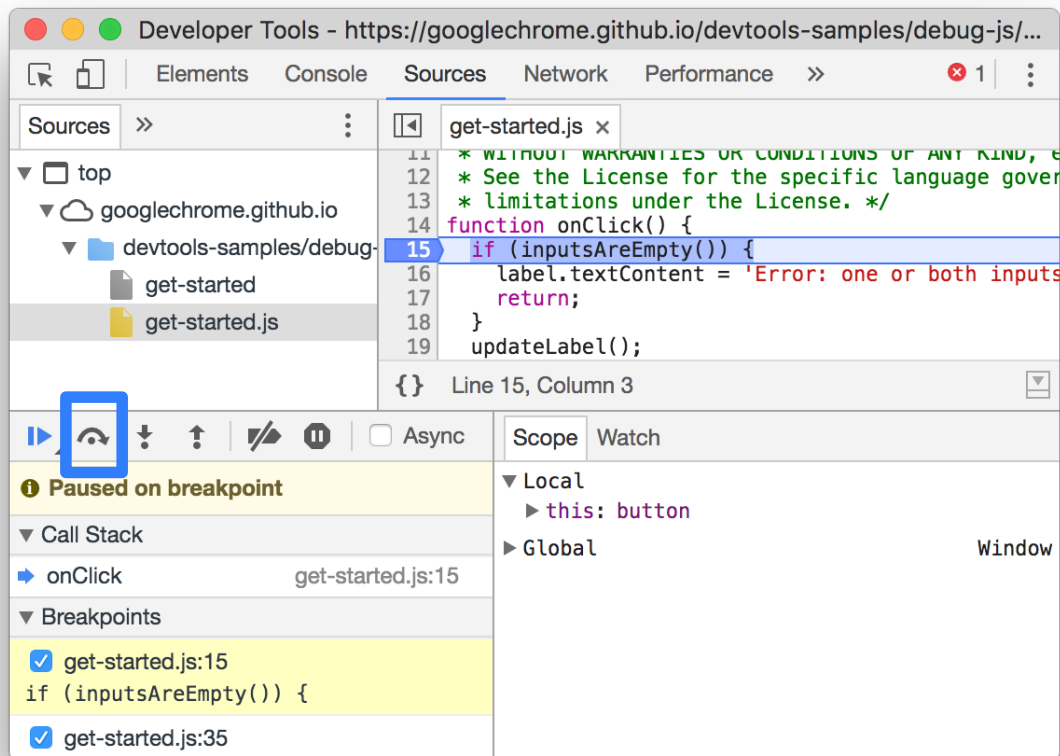



Figure 1. Step over, outlined in blue

For example, suppose you're debugging the following code:

```
function updateHeader() {
  var day = new Date().getDay();
  var name = getName(); // A
  updateName(name); // D
}
function getName() {
  var name = app.first + ' ' + app.last; // B
  return name; // C
}
```

You're paused on A. By pressing **Step over**, DevTools executes all the code in the function that you're stepping over, which is B and C. DevTools then pauses on D.

Step into line of code

When paused on a line of code containing a function call that is related to the problem you're debugging, click **Step into**  to investigate that function further.

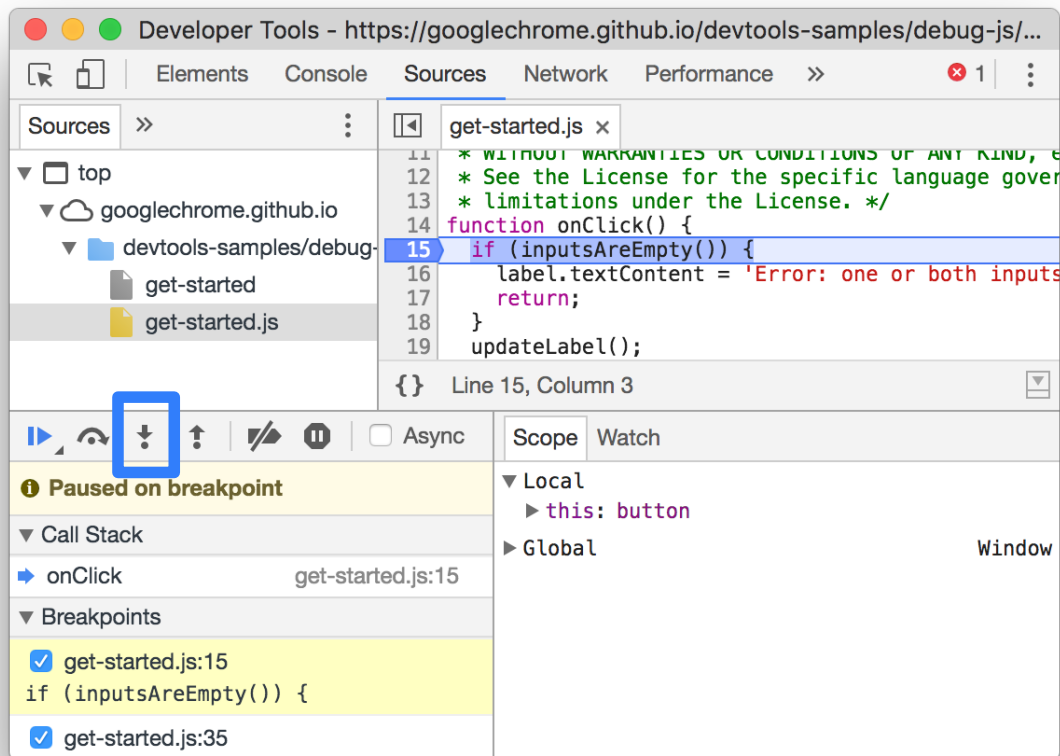



Figure 2. Step into, outlined in blue

For example, suppose you're debugging the following code:

```
function updateHeader() {
  var day = new Date().getDay();
  var name = getName(); // A
  updateName(name);
}
function getName() {
  var name = app.first + ' ' + app.last; // B
  return name;
}
```

You're paused on A. By pressing **Step into**, DevTools executes this line of code, then pauses on B.

Step out of line of code

When paused inside of a function that is not related to the problem you're debugging, click **Step out**  to execute the rest of the function's code.

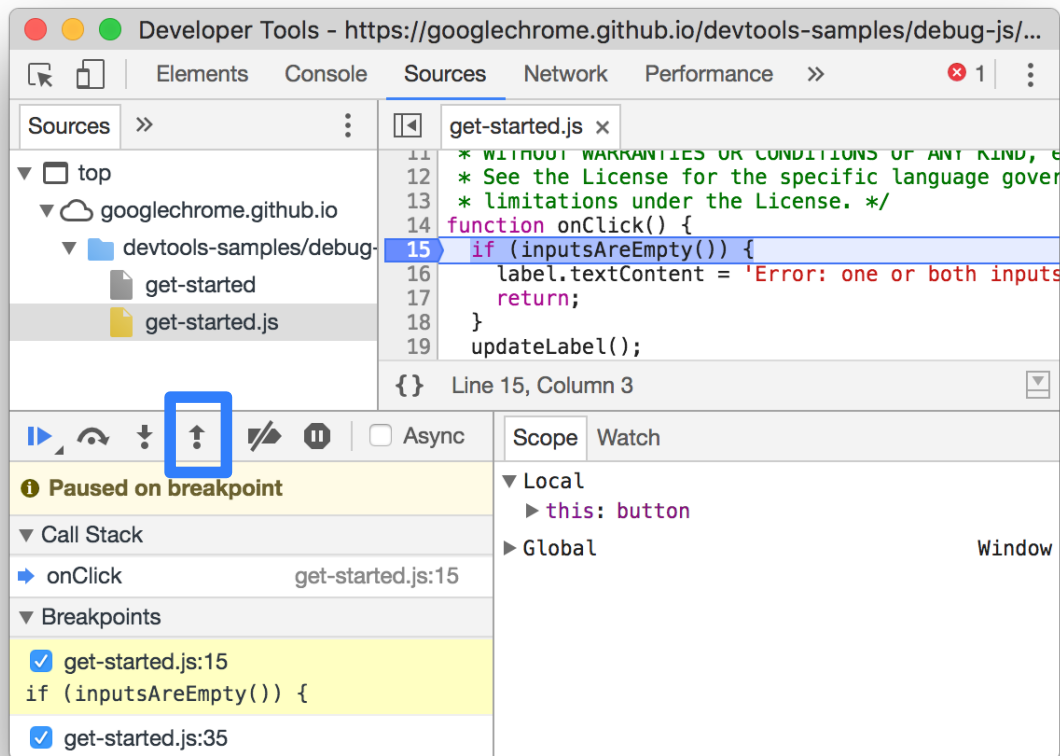


Figure 3. Step out, outlined in blue


For example, suppose you're debugging the following code:

```
function updateHeader() {  
  var day = new Date().getDay();  
  var name = getName();  
  updateName(name); // C  
}  
function getName() {  
  var name = app.first + ' ' + app.last; // A  
  return name; // B  
}
```

You're paused on A. By pressing **Step out**, DevTools executes the rest of the code in `getName()`, which is just B in this example, and then pauses on C.

Run all code up to a certain line

When debugging a long function, there may be a lot of code that is not related to the problem you're debugging.

You *could* step through all the lines, but that can be tedious. You *could* set a line-of-code breakpoint on the line you're interested in and then press **Resume Script Execution** , but there's a faster way.

Right-click the line of code that you're interested in, and select **Continue to here**. DevTools runs all of the code up to that point, and then pauses on that line.

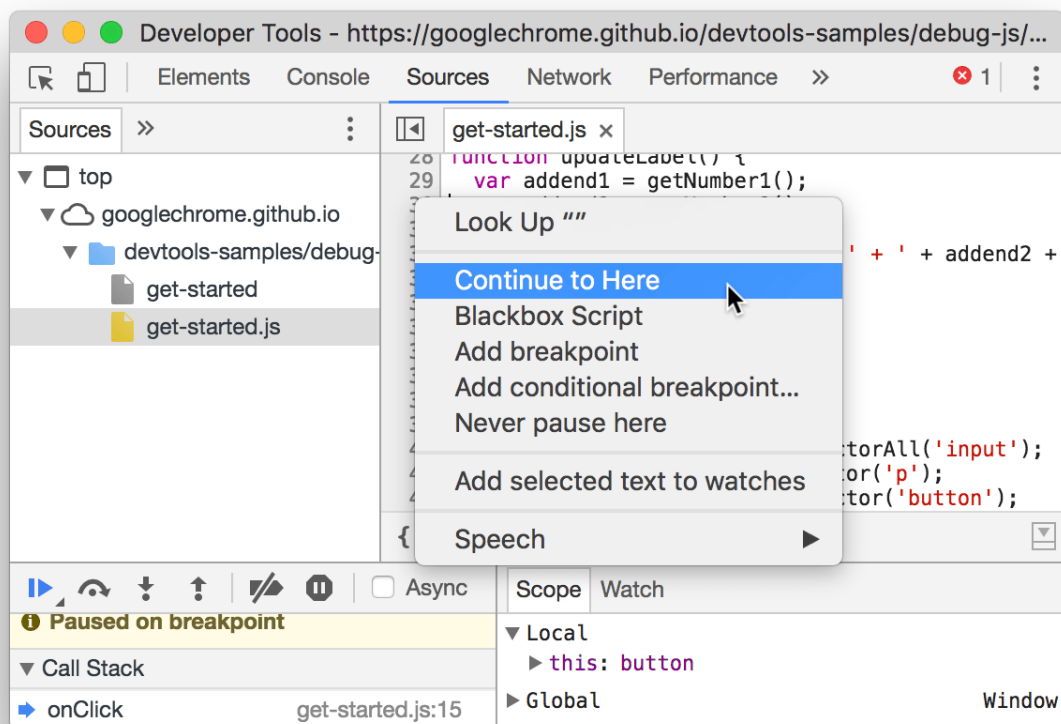


Figure 4. Selecting **Continue to here**

Restart the top function of the call stack

While paused on a line of code, right-click anywhere in the Call Stack pane and select **Restart Frame** to pause on the first line of the top function in your call stack. The top function is the last function that was called.

For example, suppose you're stepping through the following code:

```
function factorial(n) {  
  var product = 0; // B  
  for (var i = 1; i <= n; i++) {  
    product += i;  
  }  
}
```



```
    return product; // A
}
```

You're paused on A. After clicking **Restart Frame**, you'd be paused on B, without ever setting a breakpoint or pressing **Resume script execution**.

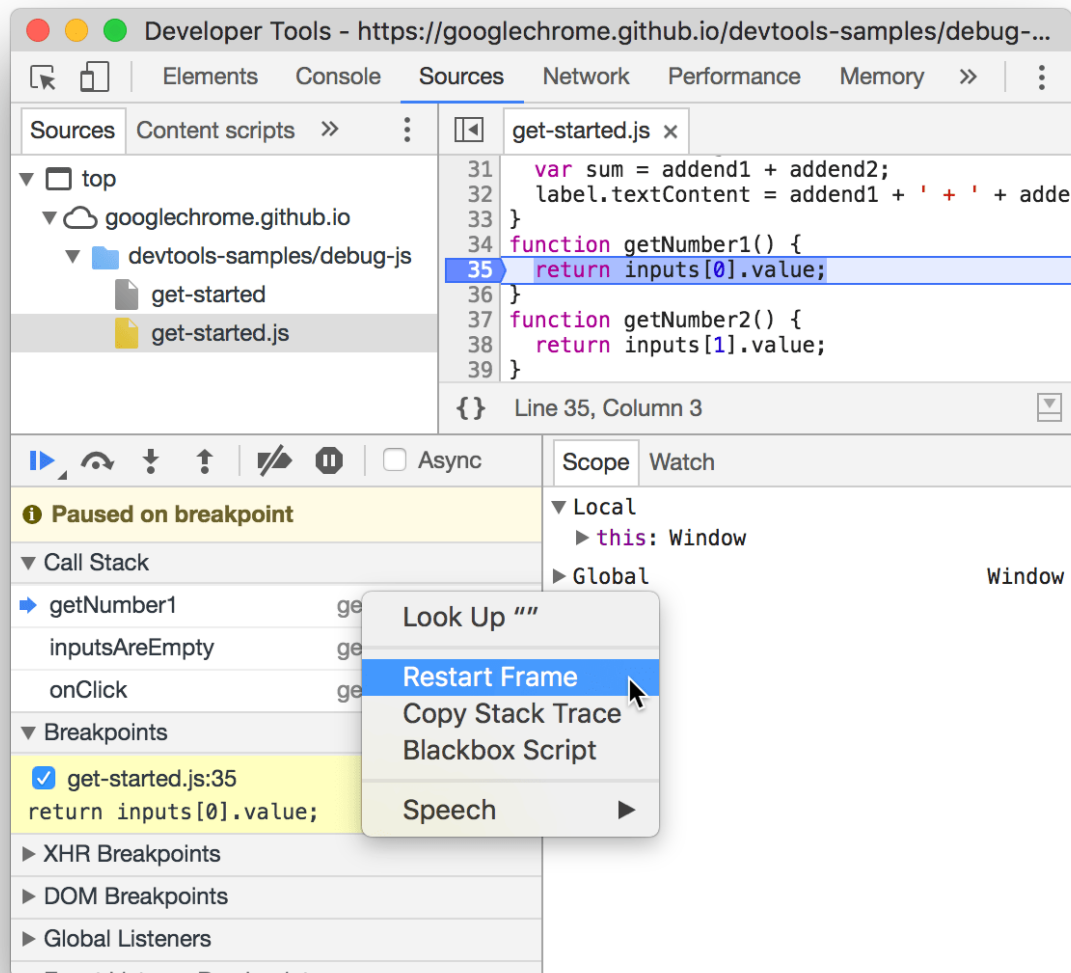



Figure 5. Selecting **Restart Frame**

Resume script execution

To continue your script's execution after a pause, click **Resume Script Execution** . DevTools executes the script up until the next breakpoint, if any.

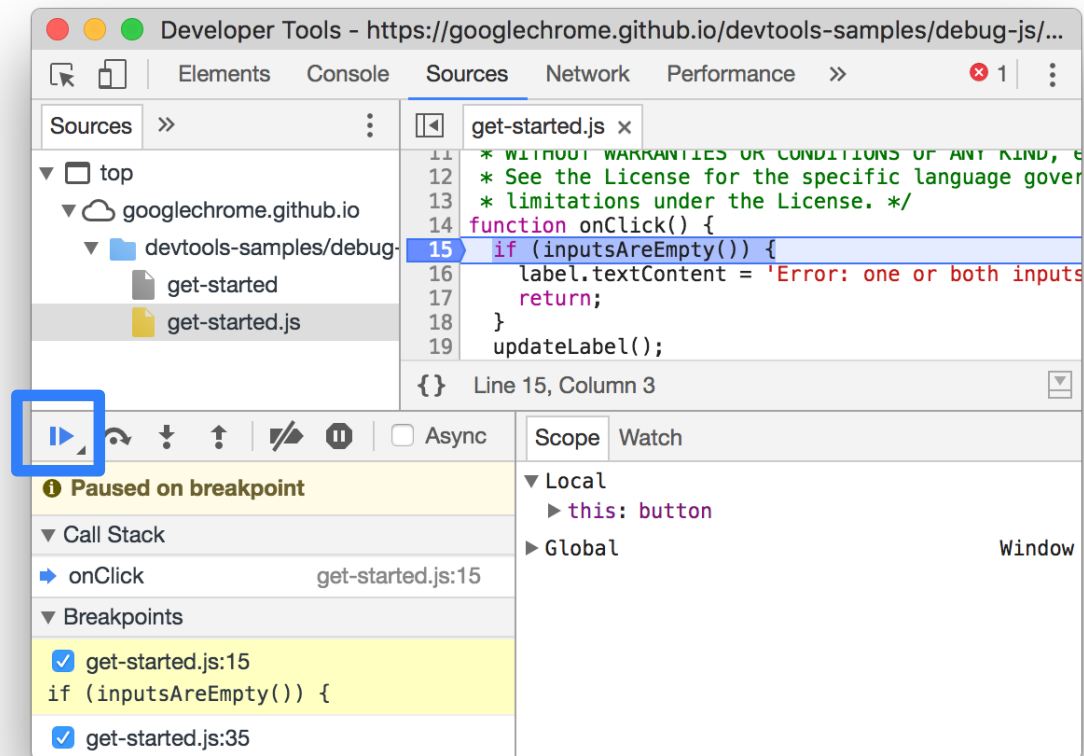




Figure 6. Resume script execution, outlined in blue

Force script execution

To ignore all breakpoints and force your script to resume execution, click and hold **Resume Script Execution**  and then select **Force script execution** .

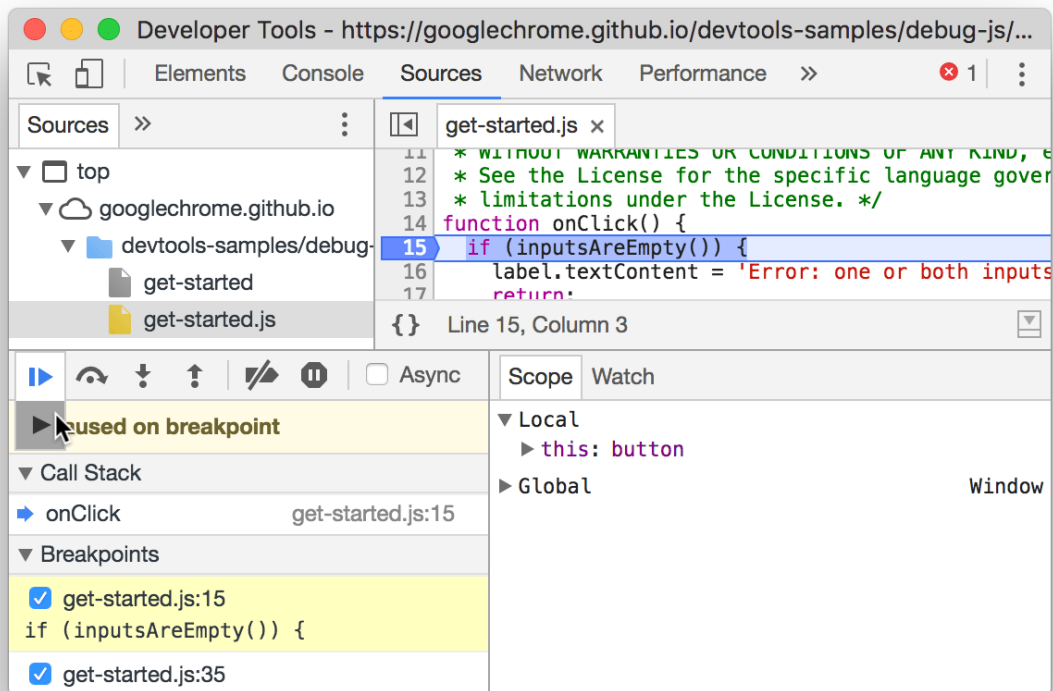


Figure 7. Selecting **Force script execution**

Change thread context

When working with web workers or service workers, click on a context listed in the Threads pane to switch to that context. The blue arrow icon represents which context is currently selected.

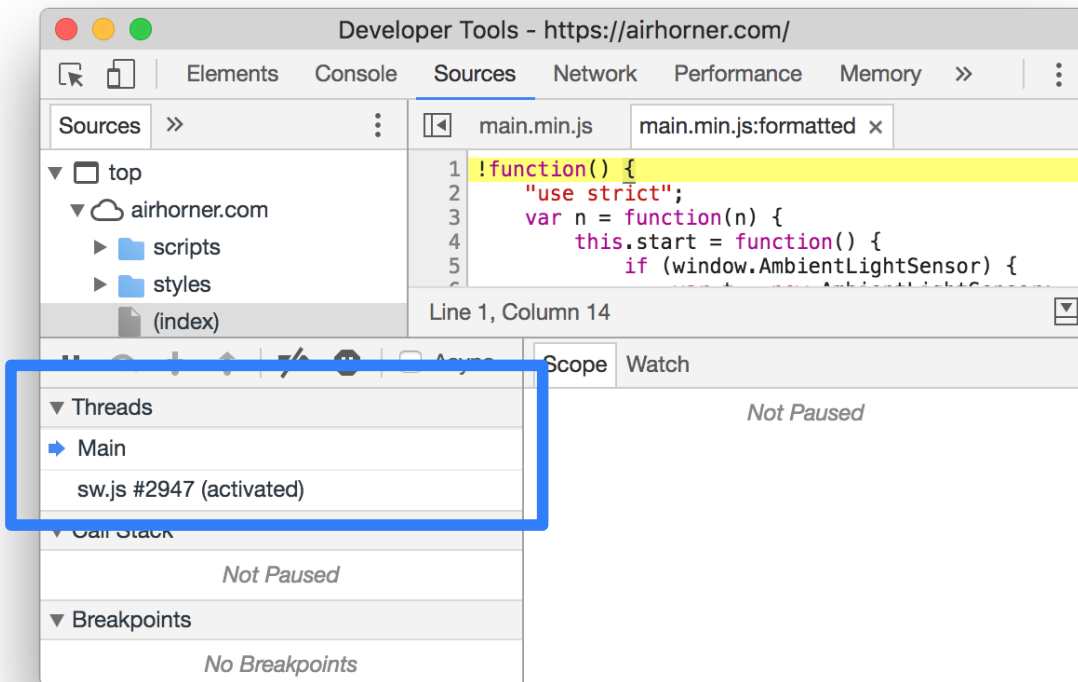


Figure 8. The Threads pane, outlined in blue

For example, suppose that you're paused on a breakpoint in both your main script and your service worker script. You want to view the local and global properties for the service worker context, but the Sources panel is showing the main script context. By clicking on the service worker entry in the Threads pane, you'd be able to switch to that context.

View and edit local, closure, and global properties

While paused on a line of code, use the Scope pane to view and edit the values of properties and variables in the local, closure, and global scopes.

- Double-click a property value to change it.
- Non-enumerable properties are greyed out.

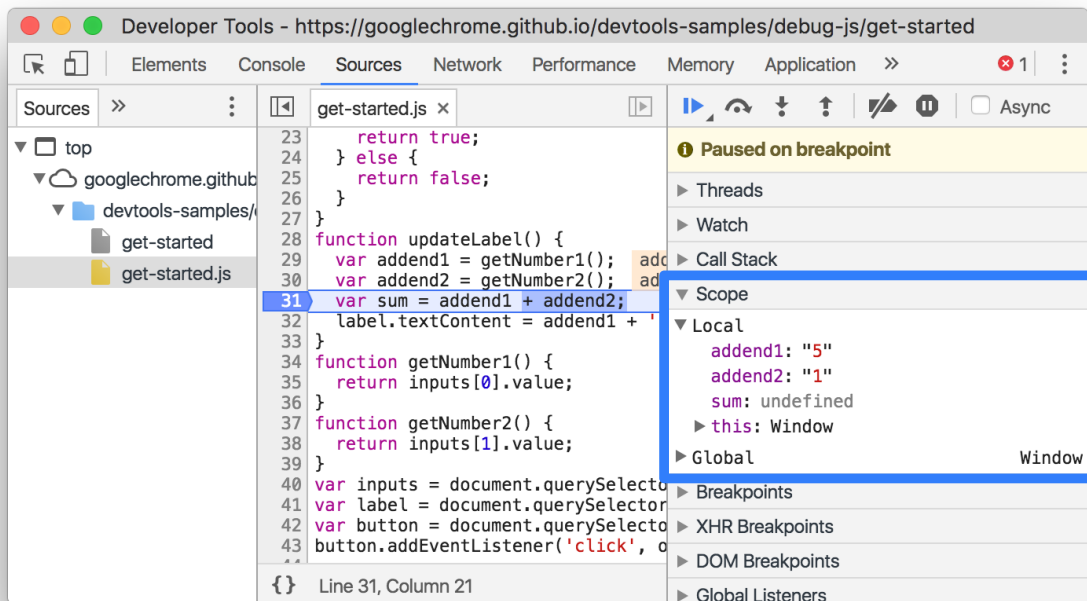


Figure 9. The Scope pane, outlined in blue

View the current call stack

While paused on a line of code, use the Call Stack pane to view the call stack that got you to this point.

If you're working with async code, check the **Async** checkbox to enable async call stacks.

Click on an entry to jump to the line of code where that function was called. The blue arrow icon represents which function DevTools is currently highlighting.

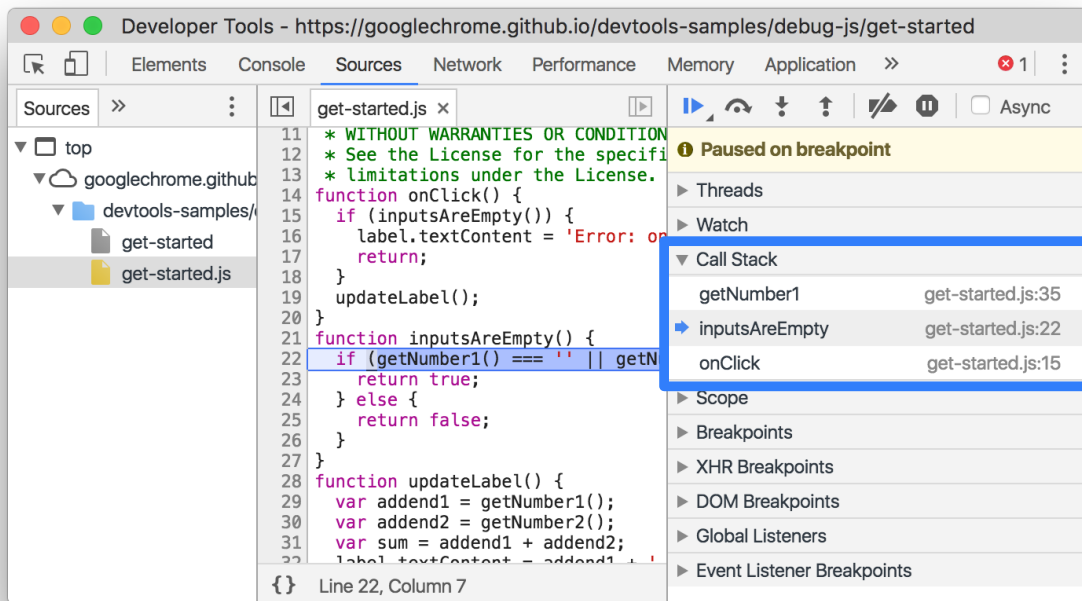


Figure 10. The Call Stack pane, outlined in blue

Note: When not paused on a line of code, the Call Stack pane is empty.

Copy stack trace

Right-click anywhere in the Call Stack pane and select **Copy stack trace** to copy the current call stack to the clipboard.

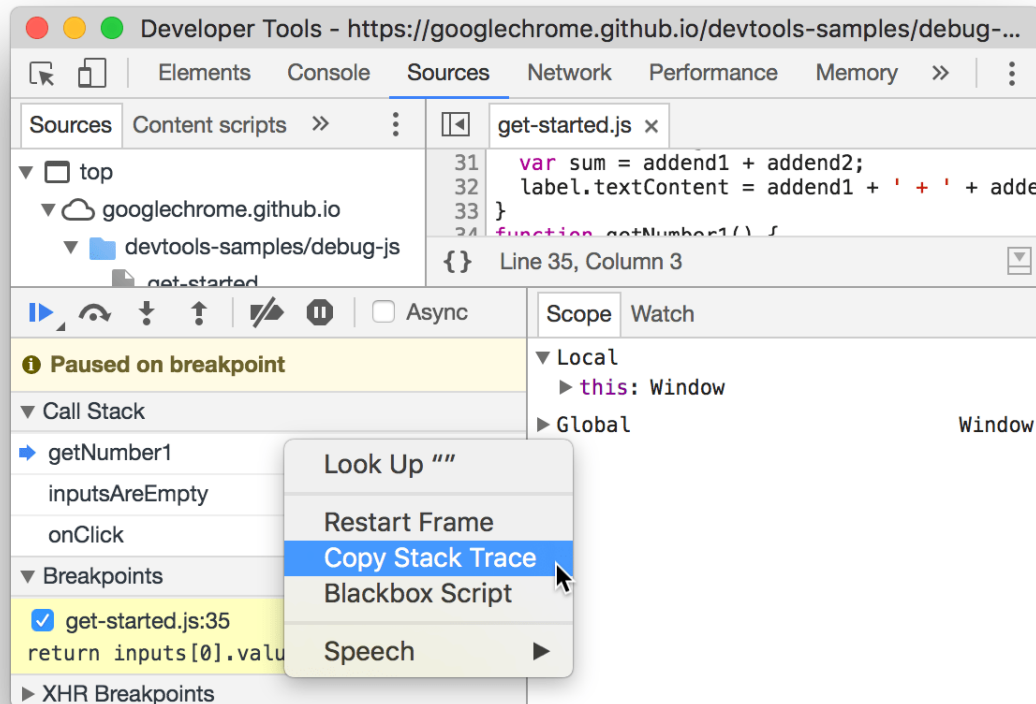


Figure 11. Selecting Copy Stack Trace

Below is an example of the output:

```
getNumber1 (get-started.js:35)
inputsAreEmpty (get-started.js:22)
onClick (get-started.js:15)
```



Ignore a script or pattern of scripts

Blackbox a script when you want to ignore that script while debugging. When blackboxed, a script is obscured in the Call Stack pane, and you never step into the script's functions when you step through your code.

For example, suppose you're stepping through this code:

```
function animate() {
  prepare();
  lib.doFancyStuff(); // A
  render();
}
```



A is a third-party library that you trust. If you're confident that the problem you're debugging is not related to the third-party library, then it makes sense to blackbox the script.

Blackbox a script from the Editor pane

To blackbox a script from the Editor pane:

1. Open the file.
2. Right-click anywhere.
3. Select **Blackbox script**.

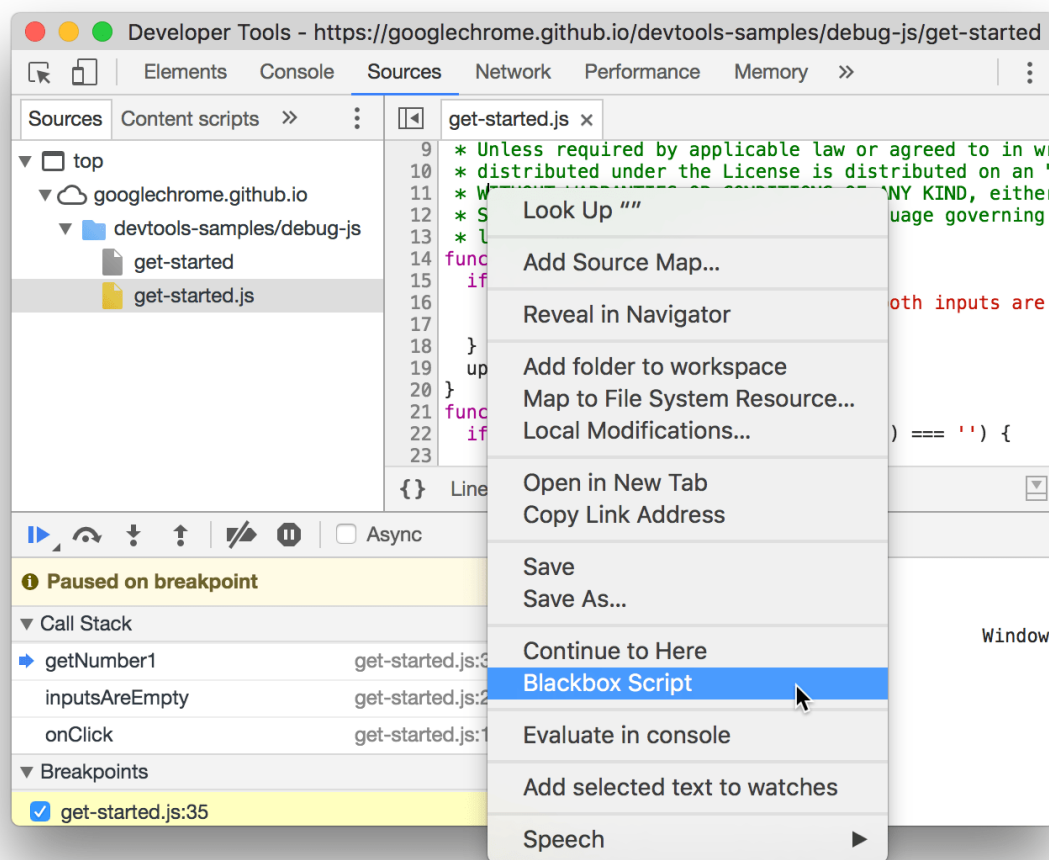


Figure 12. Blackboxing a script from the Editor pane

Blackbox a script from the Call Stack pane

To blackbox a script from the Call Stack pane:

1. Right-click on a function from the script.

2. Select **Blackbox** script.

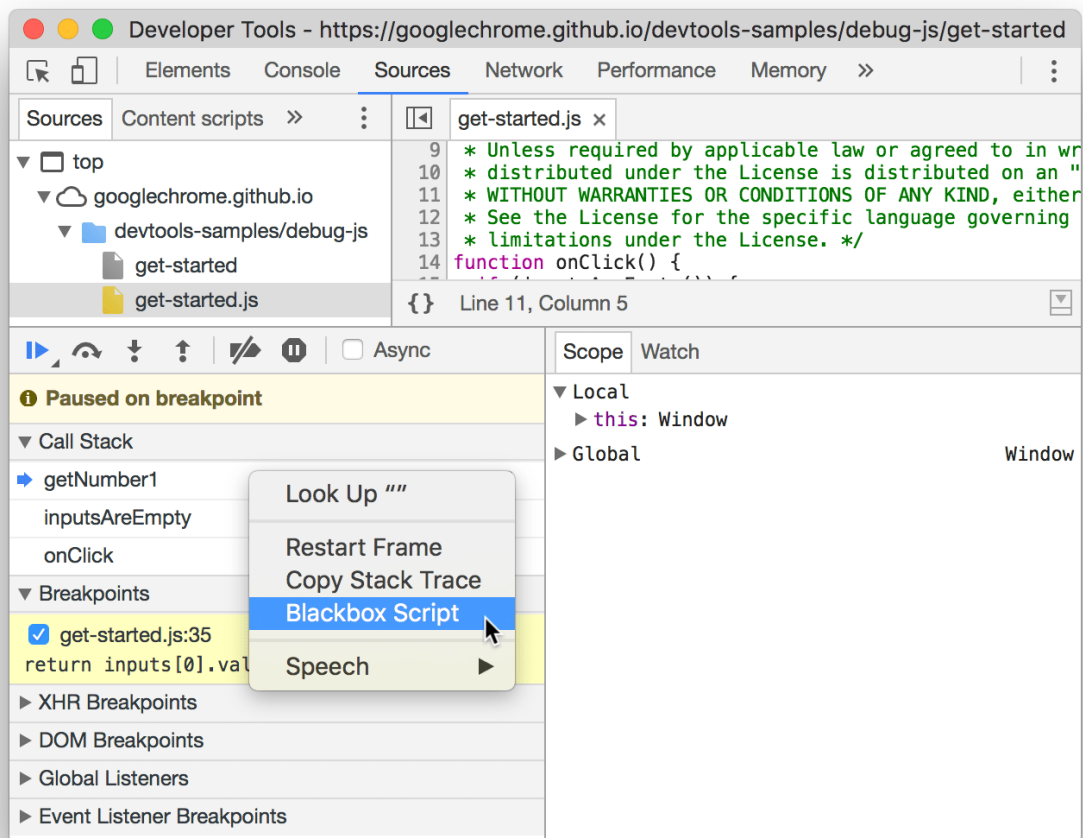


Figure 13. Blackboxing a script from the Call Stack pane

Blackbox a script from Settings

To blackbox a single script or pattern of scripts from Settings:

1. Open Settings.
2. Go to the **Blackboxing** tab.
3. Click **Add pattern**.
4. Enter the script name or a regex pattern of script names to blackbox.
5. Click **Add**.

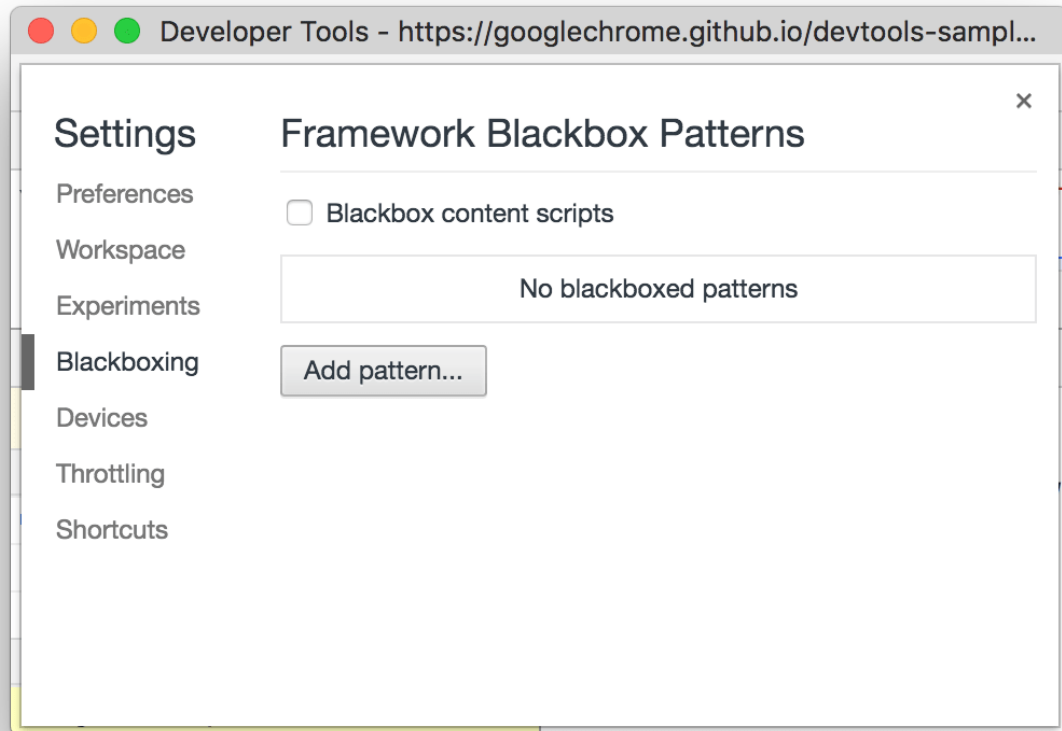


Figure 14. Blackboxing a script from Settings

Run snippets of debug code from any page

If you find yourself running the same debug code in the Console over and over, consider Snippets. Snippets are executable scripts that you author, store, and run within DevTools.

See [Run Snippets of Code From Any Page](#) to learn more.

Watch the values of custom JavaScript expressions

Use the Watch pane to watch the values of custom expressions. You can watch any valid JavaScript expression.

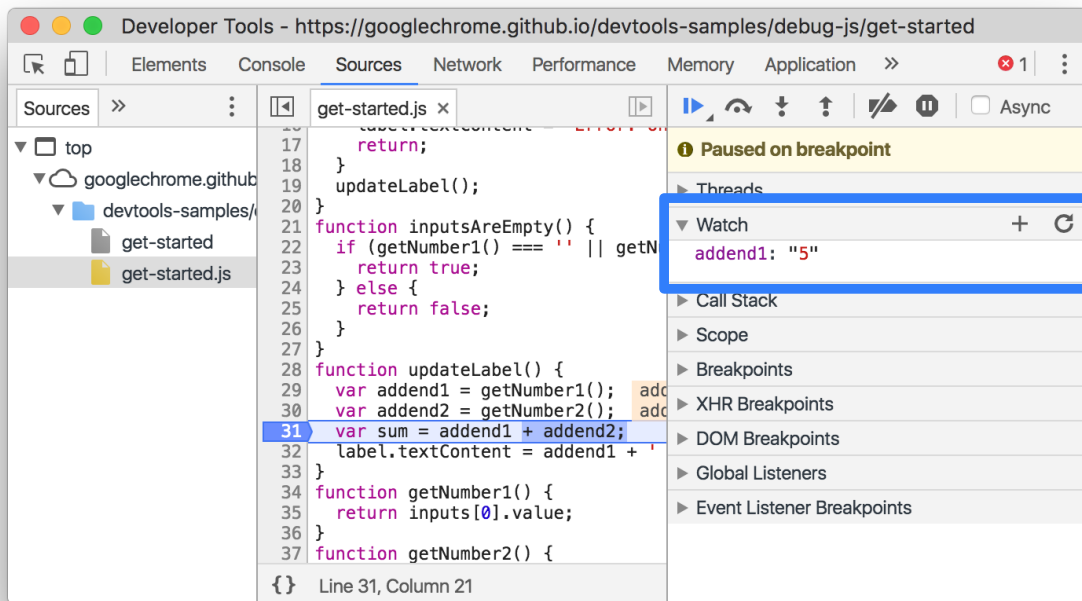






Figure 15. The Watch pane, outlined in blue

- Click **Add Expression**  to create a new watch expression.
- Click **Refresh**  to refresh the values of all existing expressions. Values automatically refresh while stepping through code.
- Hover over an expression and click **Delete Expression**  to delete it.

Make a minified file readable

Click **Format**  to make a minified file human-readable.

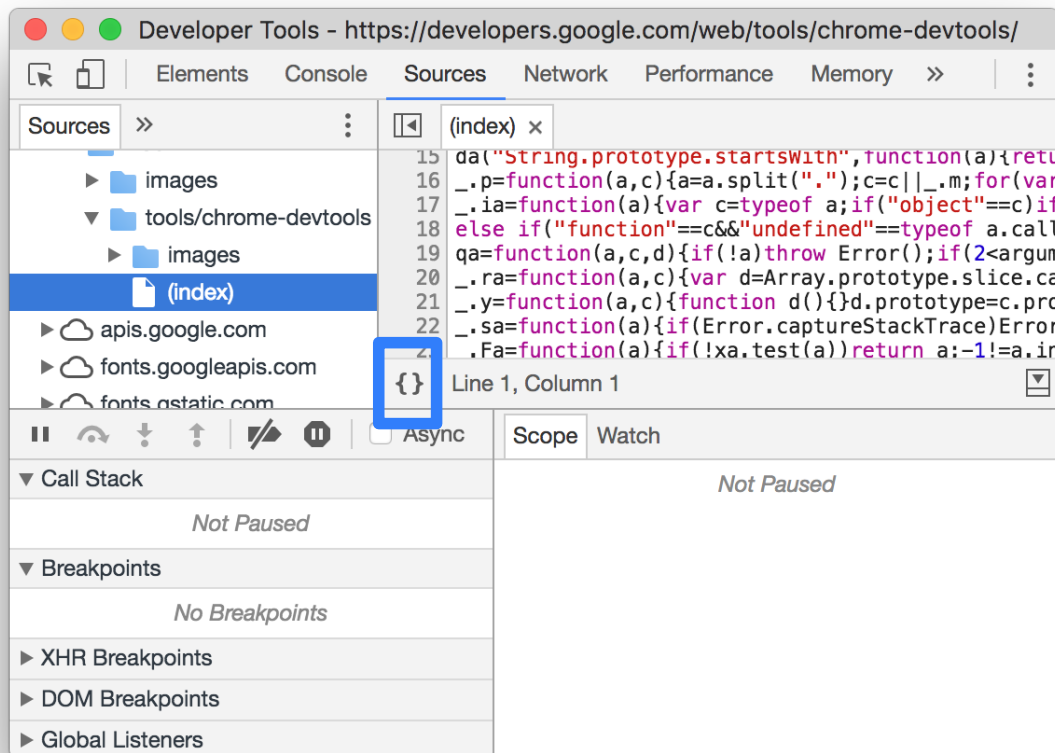


Figure 16. Format, outlined in blue

Edit a script

When fixing a bug, you often want to test out some changes to your JavaScript code. You don't need to make the changes in an external browser and then reload the page. You can edit your script in DevTools.

To edit a script:

1. Open the file in the Editor pane of the Sources panel.
2. Make your changes in the Editor pane.
3. Press Command+S (Mac) or Ctrl+S (Windows, Linux) to save. DevTools patches the entire JS file into Chrome's JavaScript engine.

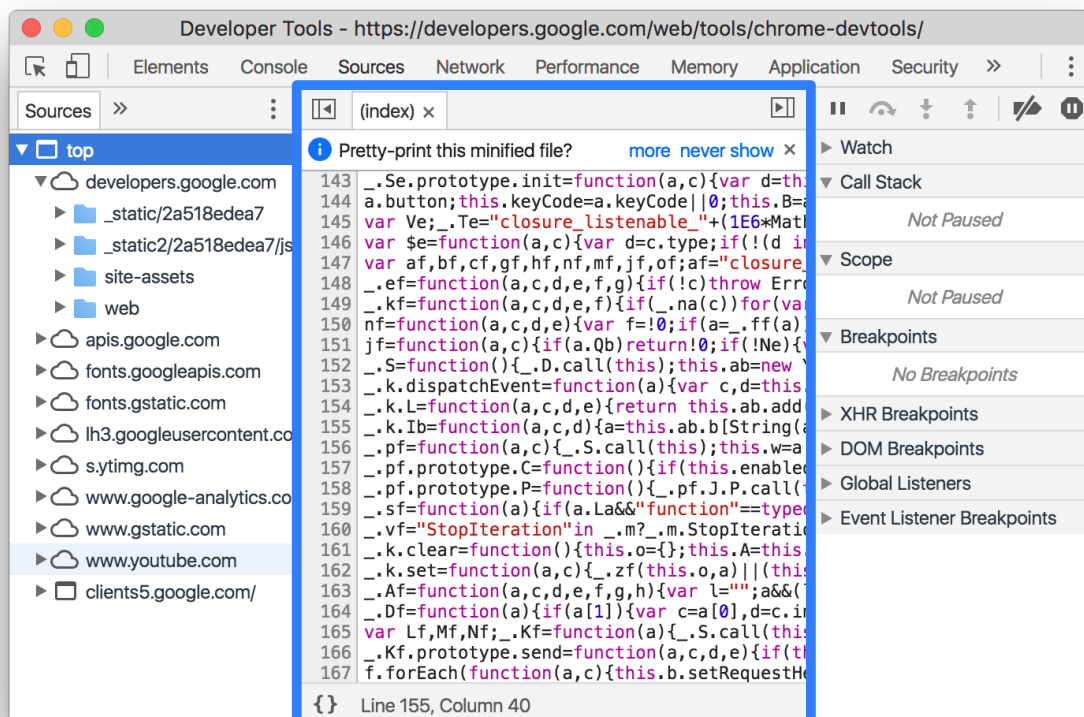


Figure 17. The Editor pane, outlined in blue

Except as otherwise noted, the content of this page is licensed under the [Creative Commons Attribution 3.0 License](#), and code samples are licensed under the [Apache 2.0 License](#). For details, see our [Site Policies](#). Java is a registered trademark of Oracle and/or its affiliates.

Last updated July 2, 2018.