DEBUGGING SCRIPT

View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/debugging-script

Browsers provide comprehensive developer tools ("dev tools" for short) to help debug and profile JavaScript code. Construct is designed to allow you to use these industry-standard tools to also debug code used in your project.

Developer tools are complex and sophisticated tools used by professional developers. For full documentation you should refer to each browser's own dev tools documentation, such as Google's Chrome DevTools documentation. However an overview is provided here, focusing on how to use dev tools with Construct specifically. Screenshots of dev tools are taken from Chrome's DevTools, as it is the most widely-used browser. Other browser's dev tools will look different, but generally they work in similar ways.

Debugging TypeScript

Browsers only directly support running JavaScript code. The way TypeScript works is it compiles the code in to JavaScript before running it, which generally amounts to deleting the type annotations. When debugging code written in TypeScript you'll actually find the compiled JavaScript output of your TypeScript code in the browser developer tools, all using the *.js* file extension rather than *.ts*. However this should not be a significant hindrance - it should look pretty much the same, including with all your comments, and only be missing the type annotations that TypeScript uses.

As TypeScript compiles to JavaScript for running in the browser, the rest of this section refers to JavaScript code.

Construct's debugger

It should be noted that Construct's debugger cannot debug JavaScript - only the browser dev tools can. In an event sheet Construct's debugger can only stop just before running a script block or script action, and continuing will run all the JavaScript in that block to completion and then advance to the next block. Construct's debugger has no way to stop on code in script files at all. Therefore it is not generally useful for debugging JavaScript code.

Using both the Construct debugger and browser dev tools simultaneously is possible, but is likely to be very confusing. Therefore it is recommended to only use one or the other at a time.

Opening dev tools

First make sure you have a preview window running and focused before opening developer tools. Otherwise you will open developer tools for the Construct editor, which is not useful, and due to security issues will show a warning in the console.

Once a preview is running, in most browsers you can press `F12` to open developer tools. In Safari, use `⌥⌘I`.

Normally the dev tools window can be undocked, so it appears in its own window. This is the easiest way to see the largest possible dev tools window, which is important for more advanced tasks like debugging JavaScript code.

For more information, including using remote debugging to open developer tools for a mobile device, see the tutorial Checking for errors in browsers.

The console

The *Console* tab provides a list of messages, used for development purposes. Calling `console.log("Hello!")` will add the message *Hello!* here. For more console features available in code, see the Console API. Adding console messages to indicate which parts of the code have been reached, and the contents of any important variables, can be a useful way to diagnose code.

You can also directly type in JavaScript code in the console, which runs when you press `Enter`. This is a useful way to try out snippets of code or quickly write and test a function to use in your project. If you are stopped on a breakpoint, code run in the console can also use any variables in the scope of the breakpoint. Often this is useful if you break in some code with the `runtime` variable available, allowing you to call runtime script interface functions from the console.

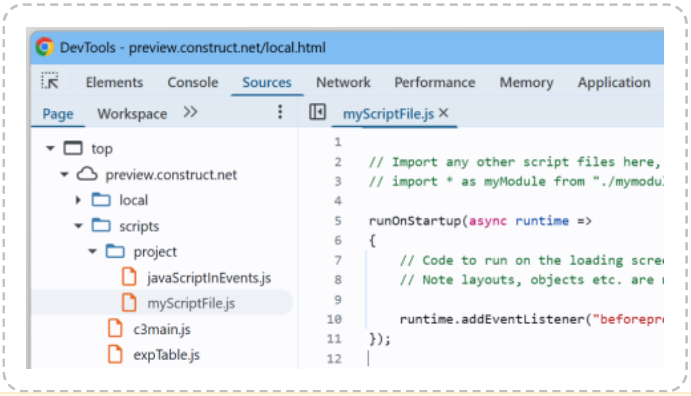
> *Remember that browsers can only directly run JavaScript code. Therefore you can only type JavaScript code in to the console - you can't enter TypeScript code there.*

Debugging JavaScript code

A more sophisticated tool is the JavaScript debugger. This is an incredibly powerful tool that allows you to pause JavaScript execution at any point, and inspect and alter the full state of your program. Normally this can be found in the *Sources* tab of dev tools.

Finding your scripts

You can find any script files in your project in this section of the debugger. For example if you add *myScriptFile.js* to your project, it will be listed here (normally on the left). The file should appear under the origin *preview.construct.net*, since that is the domain that project previews run on. Your project scripts will also appear in the subfolder **scripts/project**.



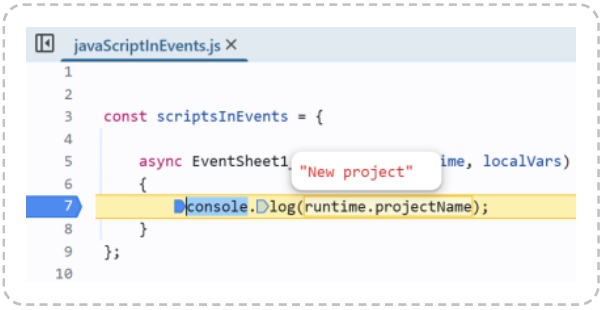
In worker mode, the scripts will instead appear under a Web Worker named Runtime.

There may also be a special script file named *javaScriptsInEvents.js*. As the name suggests, all JavaScript code used in event sheets appear in this file. Each code block appears as a function whose name is derived from its location in the event sheet, such as *EventSheet1_Event1_Act1*. There may also be *typeScriptInEvents.js* which is the same but for all TypeScript code used in event sheets, but note this is also the JavaScript output of compiling the original TypeScript file.

Messages logged to the browser console also identify where they came from. You can click the location that logged the message to open that script in the debugger, which is a useful way to jump to the relevant code.

Using breakpoints

Once you have opened a script file in the debugger, you can add a breakpoint by clicking in the margin of the file, where the line numbers appear. The breakpoint can be removed by clicking it again. When the breakpoint is present, just before that line of code runs all script execution will pause and the debugger will highlight that line, allowing you to inspect the state of your code. In this state you can hover the mouse cursor over variables to see their values, look at the call stack, and even run code in the console to alter the state of the program or call functions.



Execution will not continue until you resume the debugger. Normally there is a *Continue* button for this. There are also features to step one line at a time, jump in to or out of functions, and so on. With keyboard shortcuts experienced developers can comfortably control execution of their code through a debugger, watching how the program runs at every step.

You can also use the special *debugger* statement in code:

```
debugger;
```

Whenever this line of code runs, the debugger will automatically stop as if there was a breakpoint on that line. However it will always do this until you delete the statement from your code, whereas normal breakpoints can be removed at any time.

Breakpoints on startup

Breakpoints, including the `debugger` statement, only ever stop when dev tools are open. If you want to stop on a breakpoint run on startup, this means by the time you open dev tools, the code has already run without stopping. This can be solved by reloading the preview window once you have opened dev tools. Press `F5` with the preview window focused to refresh it, or right-click the caption and select *Reload*. (Some debuggers also provide a reload button.) Since the preview window reloads with dev tools open, all breakpoints in startup code will be able to stop.

Avoiding stepping in to runtime code

While debugging code that makes lots of calls to the runtime APIs, you may find yourself regularly stepping in to the internal runtime code. This is not normally useful as you only really want to debug your own code.

You can avoid the debugger ever stepping in runtime code by ignoring the runtime scripts. The debugger will avoid ever showing you an ignored script, even if you try to step inside it. In Chrome you can ignore a runtime script if you step in to it, right-click the code, and then select *Add script to ignore list*.

More tools and techniques

Browsers also provide advanced capabilities like profiling performance, tracking all network requests, and more. Many other debugging tools are also provided. This section has only scratched the surface; over time you'll find and learn useful debugging techniques that will help you learn to quickly diagnose problems in your code and fix them. These are invaluable tools that

experienced developers cannot live without! For beginners, learning to use a debugger and step your code one line at a time is also a very useful technique. It will show you, step-by-step, exactly what your code is doing. This helps you learn the flow of a program and often helps reveal the cause of a problem in an obvious way, where you'd be completely stuck without it.