SCRIPTS IN EVENT SHEETS

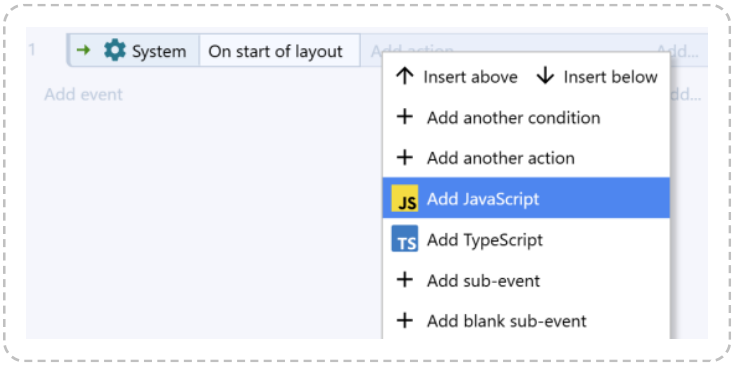
View online: https://www.construct.net/en/make-games/manuals/construct-3/scripting/using-scripting/scripts-in-event-sheets

A great way to get started with JavaScript or TypeScript is writing snippets of code in the event sheet. You can write some code to run in the place of an action or an event block.

> *In the context of JavaScript/TypeScript, an event commonly refers to a callback function that runs when something happens, similar to a trigger in Construct's event sheets. However the term event is also used in Construct to refer to a block in an event sheet. To clarify the difference the scripting section of the manual refers to event blocks or event sheets to refer to Construct's event system.*

Scripts in actions

JavaScript or TypeScript code can be added as an action. The code runs whenever the action is run, i.e. after all the event's conditions are met, and all previous actions have run. To add some code as an action, use the *Add...* menu to the right of the *Add action* link, and select *Add JavaScript* or *Add TypeScript*. Alternatively you can use the `J` or `T` keyboard shortcuts respectively when an action is selected.



A code editor will appear for you to enter the code to run.

Scripts in blocks

Alternatively JavaScript or TypeScript code can be added as a block, in the same place as other event blocks appear. The code runs whenever an event block in that place would be run. You can add a script block using any of the following methods:

1. Right-click in the margin of an existing event block and select Add▶Add JavaScript or Add▶Add TypeScript
2. Click the *Add...* menu at the end of a group, or the end of the event sheet, and select *Add JavaScript* or *Add TypeScript*

3. Use the `J` or `T` keyboard shortcuts when an event block is selected (note if an action is selected, a script action will be inserted instead)

Remember that the event sheet runs all events in top-to-bottom order every tick, so a script in a block at the top level of an event sheet is run every tick (as if it was an action in an *Every tick* event). Often it is more useful to use script blocks as sub-events, which only run if their parent event block was true.

Using 'await'

Code written in either a script action or script block is actually run inside an async function. This means your code can use the `await` keyword. For example you can await the result of a fetch over the network.

Note that the rest of the event sheet will continue to run while `await` waits for its result. In other words, `await` pauses the script execution and any actions after the code will immediately run. Then once the awaited promise resolves, any code after the `await` will then run - which will be *after* the following actions have run. You can change this using the System action *Wait for previous actions to complete*: this will wait for any code blocks before it to finish before running the following actions.

Using imports

Since code in a script action or script block is run inside a function, you cannot use `import` or `export` statements inside scripts in event sheets, since the JavaScript/TypeScript programming languages do not permit these inside functions.

Instead you can import things in to a script file with the purpose *Imports for events*, normally named *importsForEvents.js*. For example if you add the following JavaScript line in the *Imports for events* script:

```
// importsForEvents.js
import * as Utils from "./utilities.js";
```

...then all your JavaScript code in script actions and script blocks can use the exports in *Utils*, e.g. `Utils.myExport()`. This provides a useful way to write a library of functions that can be used by code anywhere else in the project, and is another good way to closely integrate event sheets with code.

Do not `export` anything in your Imports for events script. This does not work because Construct combines this script with other content and then renames it. Therefore nothing else can import the Imports for events script file as it does not really exist, and so there's no point exporting anything from it either. If you need to share content between scripts, put it in a separate script and import it to the Imports for events script.

Note that ***Imports for events* scripts are language-specific.** If you add a JavaScript *Imports for events* file, then it only applies to JavaScript code in event sheets - it will not apply to TypeScript code in event sheets - and vice versa. You can have both a JavaScript and TypeScript *Imports for events* file in the same project, but it is recommended to use either all JavaScript or all TypeScript to avoid creating confusing situations where the two *Imports for events* files have different contents.

Using the runtime interface

All scripts in the event sheet can access a special `runtime` variable which refers to the runtime script interface. This provides functions and properties that lets your code access and control Construct's runtime. This also includes ways to closely integrate code and events, such as iterating the instances picked by the current event.

A very simple example is shown below, which can be used to show a dialog box with the project name in it.

```
alert(runtime.projectName);
```

Accessing local variables

A useful way to pass values between scripts and the event sheet is to use local variables in the event sheet. These can be accessed by both script actions and script blocks using the variable name `localVars`. This is set to an object with a property for each local variable in scope. The available local variables are the same as are available to a *Set value* action in the same place. This includes any parameters for event sheet functions.

For example a script in an event group with a local variable named *temp* can access the local variable using `localVars.temp`. A useful pattern is to use an action to set a local variable to an expression, and then read from it in a following script action. Alternatively a script could calculate a value and assign it to a local variable, to subsequently be used in the event sheet. It could also be used both ways at once, both reading the variable and then assigning it. See also the Integrating events with script example which shows one of the ways this can be used.

Note that `localVars` excludes global variables, which are available via runtime.globalVars instead. `localVars` is also unique to scripts in the event sheet - script files cannot access it, because they do not have a scope in the event sheet.

> *In some cases, event variables may have names that aren't valid JavaScript/TypeScript identifiers. In this case you can use the string property syntax, e.g.* `localVars["temp"]`.

Errors

Any exceptions, or rejected promises, arising from a script in an event sheet will be caught by the Construct engine and logged to the console with information about where the error came from. This means unhandled exceptions or rejections will not crash the game (since browsers stop

running scripts if they encounter an unhandled error). However you should keep an eye on the browser console to check for any unexpected errors. For more information see the section on debugging script.