CODING IN CONSTRUCT

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

This section covers some details about how JavaScript and TypeScript code is run in Construct.

Expressions aren't exposed to script

Construct's event sheets fundamentally work differently to code. Consequently object expressions that you may enter in to conditions and actions are not directly accessible from script. Refer to the scripting reference for the methods and properties that are directly accessible from scripts. You can also still indirectly access things that are not directly exposed to scripts by passing values between scripts and event sheets - for an example of that see integrating events with script.

Case sensitivity

Many features in Construct are case insensitive, such as "Player" being considered equivalent to "player". However like most programming languages, JavaScript and TypeScript are case sensitive. Therefore when referring to objects, variables and other items in your project from code, you must use the same case as it was written with in Construct.

Modules

All scripts in Construct are Modules (also known as ES Modules, or ESM). This allows use of the `import` and `export` syntax. It also has a few differences to legacy "classic" mode scripts, notably:

1. Each module (script file) has its own top-level scope. This means top-level declarations are not globally accessible. Instead prefer to use imports and exports, or if you have to, explicitly use properties of `globalThis` to make them global.
2. Modules run code in strict mode. This eliminates common problems such as silent errors, typos accidentally creating variables, and fixes some aspects of the language considered mistakes. In particular it helps avoid confusing problems that beginners often run in to. Since all code is already run in strict mode, there is no need to add the `"use strict"` directive to any of your code.

See the Imports & exports example for a demonstration of using modules in Construct.

Worker mode

When the *Use worker* project property is *Yes*, the Construct runtime is hosted in a Web Worker instead of in the DOM, and renders using an OffscreenCanvas. This is generally good for

performance since the runtime can run independently of the browser main thread, which can be blocked by browser tasks like layout. However Web Workers have a reduced set of APIs available. Notably there are no `window` or `document` objects available, and so the DOM cannot be directly accessed. However there are many other APIs still available, such as fetch (and the older XMLHttpRequest), IndexedDB, WebSocket and others - see Functions and classes available to workers for more details.

The default mode for *Use worker* is *Auto*. This means Construct automatically decides whether to run your project in a Web Worker or the DOM. Currently this means it will run in a worker unless your project uses any JavaScript code, in which case it will run on the DOM on the assumption your code will need to make use of DOM APIs. Note however that this loses the performance benefits of worker mode. If your JavaScript code can run in a Web Worker, you can change the setting back to *Yes* and gain the performance benefits.

Worker mode with TypeScript

When using TypeScript, Construct will update type definitions to match the environment. In other words if your project runs in a worker, it will use type definitions for a worker, so accessing things like `document` will be an error. Otherwise if the project runs in the DOM then it uses type definitions for the DOM, permitting the use of such DOM-specific APIs.

Accessing global scope

Sometimes the `window` object is used to refer to global scope, such as with `window.myGlobal = 1;`. Note however that the `window` object is not available in worker mode, so this won't work there.

The standard way of accessing the global scope is with `globalThis`, such as with `globalThis.myGlobal = 1;`. Since this is available in both the DOM and Web Workers, this code will work everywhere.

Browser/platform support

When using the very latest JavaScript features, you may need to check which browsers support it. For example if you use a JavaScript feature that not all browsers or platforms support, you may get an error trying to run your game on that platform. We recommend the MDN web docs as a good place to check compatibility.

This also applies when using TypeScript. Since TypeScript essentially just deletes type annotations and then runs the code as JavaScript, you still need to be aware of browser support for any newer JavaScript features you use.

You can also rely on modern support for JavaScript being available, because Construct only supports relatively modern browsers. For example the C3 runtime does not support Internet Explorer so you do not need to worry about supporting it at all. Note many older code examples across the web use an older style designed to support defunct browsers. For example many old code examples use `var` to declare variables, whereas in modern JavaScript `let` and `const`

are preferred. Bear this in mind when looking at other code examples, and note there may be modern features that can considerably simplify the code. It can be a good idea to update any code snippets you use in your projects to a modern style.

Undocumented features

> *Do not use undocumented features in your JavaScript code.*

If you explore the functions and variables available in a debugger, you may find undocumented APIs specific to the Construct engine. The only reason these can be found is because the way JavaScript works makes it difficult to hide them. **Do not use any such undocumented features in your JavaScript code.** These are internal details of the Construct engine and can change at any time, and such changes can easily break your code. **No support will be provided for undocumented APIs**, even if engine changes break your code. Responsible developers know to only use documented and officially supported APIs. These can be found in the scripting reference. If new engine functionality is essential to you, please file a feature request.

Note this only applies to the Construct engine - all other browser APIs are of course available for use.

When using TypeScript, type definitions are only provided for documented features, so it will prevent you using undocumented features. This can be circumvented with the `any` type, but naturally we would strongly advise against that.