

EXPORTING WITH ADVANCED MINIFICATION

View online: <https://www.construct.net/en/make-games/manuals/construct-3/scripting/guides/advanced-minification>

When **exporting your project**, there is an option to *Minify script*. This compresses all the JavaScript code used both in the Construct engine and in your own scripts, and can help obfuscate the project to make it more difficult to reverse-engineer. However if you choose the *Advanced* mode, you may need to adjust how you write JavaScript code in Construct.

When using TypeScript the same adjustments also need to be made. The TypeScript compiler essentially just removes type annotations and outputs regular JavaScript code which is then what gets minified, so the rest of this guide focuses on JavaScript.

The minify mode *None* skips minifying scripts so they are not altered. *Bundle only* combines multiple scripts into a single file without minifying. *Simple* mode eliminates whitespace and does simple adjustments like renaming local variables to shorter names. This does not affect how any of the code is run so is always safe to use. *Advanced* mode renames everything else, including object properties, class method names, and so on. This process is referred to as *property mangling*. It provides the best compression and obfuscation, but in some cases it can affect how the code is run. Therefore you have to be aware of how it works and write your code accordingly to safely use it.

Everything in the Construct engine supports Advanced mode. You only need to be careful about using Advanced mode if you use the scripting feature to write JavaScript code in Construct.

How property mangling works

Property mangling renames object properties to shorter names. For example consider the following code:

```
const obj = {  
    apples: 1,  
    oranges: 2  
};  
console.log(obj.apples, obj.oranges);
```

This will log the numbers 1 and 2 as they correspond to the object properties *apples* and *oranges*. After advanced minification the properties are renamed to shorter names, e.g.:

```
const obj = {
  a: 1,
  b: 2
};
console.log(obj.a, obj.b);
```

This is shorter code (which is faster to load) and harder to understand (which is harder to reverse-engineer). It also works identically to how it did previously.

Avoiding renaming

In some cases there are specific names you *don't* want renamed. For example if you load an external library at runtime and call a method like this:

```
externalLibrary.doSomethingUseful();
```

Property renaming will then change the name of the function, e.g.:

```
externalLibrary.a();
```

This then breaks the code, since it switched to calling a function that doesn't exist. The problem is that the minification process doesn't know about code that comes from outside your own scripts, and ends up renaming things it shouldn't.

To avoid this, you can use the string property syntax instead, as shown below.

```
externalLibrary["doSomethingUseful"]();
```

Advanced minification never renames string properties, so the name *doSomethingUseful* is preserved even after minification, and the code continues to work.

To write a global name as a string property, access it as a property of `globalThis`, e.g.:

```
globalThis["myGlobalFunction"]();
```

Built-in names

Construct uses an internal list of built-in browser API names which the minifier uses to avoid renaming built-in functions and properties. For example the names *console* and *log* are on the list, so it never renames the properties in built-in calls like `console.log("Hello")`.

In some cases a browser API may not be on the internal list, for example if it was only just added in the latest browser release. In this case you can access it with string properties to ensure it's not renamed.

Object and variable names from Construct

Some runtime APIs use the names of things from the Construct editor, such as `runtime.objects.Sprite` (which takes the name *Sprite* from the name of an object). These properties are all consistently renamed with everything else in your code, so you should access them normally as dot properties, to make sure your code still works after minifying.

Only object names that are valid JavaScript identifiers will be renamed. For example `Sprite` will be renamed, but `0Sprite` will not. The reason for that is because `0Sprite` is not a valid JavaScript identifier, it cannot be used as a dot property (e.g. `runtime.objects.0Sprite` is invalid). It must always be referred to as a string property (e.g. `runtime.objects["0Sprite"]`), and string properties are not changed by the minifier. Consider always using valid JavaScript property names for object names to avoid any confusion.

Don't mix property styles

The main mistake that breaks code with advanced minification is mixing normal properties (which are renamed) with string properties (which are not). For example consider the following code which will be broken:

```
const obj = {
  "apples": 1
};
console.log(obj.apples);
```

This uses both string syntax and normal syntax for the property name *apples*. Normally this code works (logging the number 1), but after advanced minification only one of the properties is renamed, resulting in code like this:

```
const obj = {
  "apples": 1
};
console.log(obj.a);
```

Notice how `"apples"` had its name preserved (because it uses string syntax), but `obj.apples` was renamed to `obj.a`. That property does not exist on the object, so now the code logs `undefined` instead of 1.

The solution is to always use the same syntax consistently. So long as a property name consistently never uses string syntax, or consistently always does use string syntax, it will work correctly. Code is only broken if the two types are mixed.

There are several less-obvious ways string properties can end up mixed up. For example `Object.defineProperty()` only takes a string of a property name. Since strings are never renamed, if you use this method, that property name must always be referred to with string syntax.

Global variables

One extra requirement of *Advanced* minifying is that global variables must always be referred to as a property of the global object. For example the code below shows a problem due to a global variable not being referred to as a property.

```
globalThis.myGlobal = 1;
console.log(myGlobal);
```

The use of `console.log(myGlobal)` normally does refer to the global variable. However this will fail to minify in advanced mode. (This is due to the difficulty of automatically renaming global variables that are written the same way as local variables.) The solution is to always refer to global variables as a property of the global object, i.e. with a dot, as shown below:

```
globalThis.myGlobal = 1;
console.log(globalThis.myGlobal);
```

Debugging problems

If *Advanced* minification appears to be breaking your code and you're finding it difficult to identify the problem, one option that can help is to choose the *Debug advanced* minify option. In this mode all properties that get renamed are changed to something that preserves the original name - for example `obj.apples` may be renamed to `obj._$apples$_`. This means anything that is broken by properties being renamed will still be broken, but you can see in the source code which property it is attempting to read. For example if an error occurs because `obj._$apples$_` is undefined, and you can see in the debugger that `obj.apples` exists, it would indicate that the property `apples` is inconsistently using both dot and string syntax. This mode also "beautifies" the output to help make it more readable.

Conclusion

Script minification helps compress the size of the code and makes reverse engineering more difficult. Advanced minification provides the best compression and protection, but requires some care in writing code to ensure it won't be broken by property mangling. Often code will work just fine without any specific changes in advanced mode. The only cases to be aware of are:

- ❶ Calling external library functions (or new browser APIs that aren't on Construct's built-in names list)
- ❷ Mixing string property syntax with normal property syntax
- ❸ Consistently referring to global variables as global properties

If you make sure your code handles the above correctly, then you should be able to safely use advanced minification mode with your own code in Construct.