Cheat Sheets

emcc Flags

General flags	

`-s INVOKE_RUN`	`// Don't auto-run main() (also see Module object)`
	`// Default = 1`
	`-s INVOKE_RUN=0`

`-s FORCE_FILESYSTEM`	`// Include support for a virtual file system`
	`// Default = will detect if using files in C code`
	`-s FORCE_FILESYSTEM=1`

`-s EXPORTED_FUNCTIONS`	`// Export C functions (LLVM prepends underscore)`
	`// Default = ['_main', '_malloc']`
	`-s EXPORTED_FUNCTIONS="['_increment']"`

`-s EXTRA_EXPORTED_RUNTIME_METHODS`	`// Export additional Module.fn() runtime functions`
	`// e.g. cwrap() calls custom C functions from JS`
	`// Default = []`
	`-s EXTRA_EXPORTED_RUNTIME_METHODS="['cwrap']"`

`-s EXIT_RUNTIME`	`// Allow code to exit the runtime`
	`// Otherwise, after main() is called, stdio`
	`// streams are not flushed (see this FAQ entry)`
	`// Default = 0`
	`-s EXIT_RUNTIME=1`

Include libraries	

`-s USE_ZLIB`	`// Include zlib library for .gz support`
	`// Default = 0`
	`-s USE_ZLIB=1`

`-s USE_PTHREADS`	`// Add support for pthreads`
	`// Default = 0`
	`-s USE_PTHREADS=1`

Include SDL libraries (type `em++ --show-ports` to see all available ports)	
`USE_SDL`	`// Define which version of SDL to use (e.g. v2)` `-s USE_SDL=2`

USE_SDL_IMAGE	// Define SDL Image library version to use
	// It's used to load images such as BMP/PNG/etc
	-s USE_SDL_IMAGE=2

USE_SDL_TTF	// Define SDL TTF library version to use
	// It's used to enable TTF fonts
	-s USE_SDL_TTF=2

USE_SDL_MIXER	// Define SDL Mixer library version to use
	// It's used for managing audio
	-s USE_SDL_MIXER=2

gcc Options	
-o	// Compile to .wasm and generate .js glue code -o myscript.js
-O	// See the standard gcc optimization flags -O2
-I	// Define location of header files -I ./Common/
Preloading files	

--preload-file	// Preload a local file and mount to /
	--preload-file localfile.txt
	
	// Preload a local file and mount to /tmp
	--preload-file localfile.txt@/tmp/myfile.txt
	
	// ^^ both these approaches will output a .data
	// file, in addition to the .js/.wasm files
	// already output

--use-preload-plugins	// Will automagically decode files based on their
	// extensions. See Chapter 8 for an example of
	// how we used this flag alongside --preload-file
	--preload-file /data \
	--use-preload-plugins

Module Object

For more info, see Chapter 5.

Module parameters

Customize initialization		
noInitialRun	Parameter	Set to true if you don't want `main()` to be called at page load
arguments	Parameter	Array of arguments sent to `main()` function once the module is initialized (only valid if `noInitialRun = false`).
onRuntimeInitialized	Callback	Called once WebAssembly module is ready to be used. e.g. `() => console.log("Initialized");`
locateFile	Callback	By default, the `.wasm` file will be loaded from the same folder as the `.js` file. If that is not correct, use this function to define a different path, e.g. `path => `wasm/${path}``, or even a URL to download the file from, e.g. `path => `https://domain.com/wasm/${path}``
preInit	Function(s)	This function is called before the WebAssembly module is downloaded and initialized. `preInit` accepts an array of functions; note that the last function in the array will be executed first. This is a useful place to mount other file systems as we cover in Chapter 6.
preRun	Function(s)	This function is called after preInit, after the WebAssembly module is loaded, but before `callMain()` is called. As with `preInit`, `preRun` either accepts a function or an array of functions; last function in the array is executed first.
onAbort	Callback	This function is called if the `.wasm` file you're trying to load is not found; useful for detecting such errors and notifying the user and/or logging an error in your systems. This function is also called if you make a call to `abort()` within your C code.
Customize stdout/stderr behavior		
print	Callback	Custom function to capture stdout, e.g. `out => alert(out)`
printErr	Callback	Custom function to capture stderr, e.g. `err => alert(err)`
logReadFiles	Parameter	If set to `true`, will output "`read file: <path>`" to `stderr` the first time you read from a file (see Chapter 6 for details about file management with WebAssembly).

Sample Usage

```
<script type="text/javascript">
var output = [];
var Module = {
  // Don't run main on page load
  noInitialRun: true,
  // Run custom function on page load
  onRuntimeInitialized: () => {
    console.log("Launching main...");
    Module.callMain();
    console.log("Done");
  },
  // Custom function to process stdout: keep in memory
  print: stdout => output.push(stdout),
  // Custom function to process stderr: show in console as errors (in red)
  printErr: stderr => console.error(stderr)
};
</script>
<script src="hello.js"></script>
```

Note

Remember to declare the `Module` variable before you import the `.js` file output by Emscripten.

File System

For more info, see Chapter 6.

Files	
Create file	`FS.writeFile("file.txt", "some contents");`

Create file, with options	`FS.createDataFile(`
	`  "/data",  // folder where file will be saved`
	`  "file.txt",  // file name`
	`  "abcdef",  // file contents (a string)`
	`  true,  // is this file readable?`
	`  true  // is this file writable?`
	`);`

Rename file	`FS.rename("/data/file.txt", "/data/renamed.txt");`
Read file contents	`FS.readFile("/data/file.txt", { encoding: "utf8" });`
Delete the file	`FS.unlink("/data/file.txt");`
Folders	
Create folder	`FS.mkdir("/data");`
List folder contents	`FS.readdir("/data");`
Delete empty folder	`FS.rmdir("/data");`
Get working directory	`FS.cwd();`

FileReader API

For more info, see the WebWorkers + FileReader APIs Guide.

File Object

- name: file name
- type: MIME type
- size: size in bytes
- lastModified: UNIX timestamp representing last modified date

Sample Usage

```
<input type="file" id="upload">

<script type="text/javascript">
window.onload = function()
{
  // Create a FileReader object
  var reader = new FileReader();

  // Watch for files being selected
  document
    .getElementById('upload')
    .addEventListener('change', function(e) {
      // Retrieve File object from selected file
      var file = this.files[0];
      // Define what should happen once the file is loaded
      reader.onload = event => console.log(event.target.result);
      // Read the File object as text
      // Use .readAsBinaryString() to read as a binary file
      reader.readAsText(file);
    });
}
</script>
```

WebWorkers

For more info, see the WebWorkers + FileReader APIs Guide.

Sample Usage

```
// Main Thread

// Launch worker
var worker = new Worker("worker.js");

// Catch messages from worker
worker.onmessage = msg => {
  var data = msg.data;
  console.log("worker says:", data);
};

// Sample data
var size = 100 * 1024 * 1024;
var data = new Int8Array(size);

// Send message to Worker
worker.postMessage(data);

// Transfer message to Worker
worker.postMessage(data);
</script>
```

```
// WebWorker code (worker.js)

// Catch messages from main thread
self.onmessage = msg => {
  var data = msg.data;
  console.log("main thread says",data);

  // Send a message to main thread
  self.postMessage("message");
};
```

Note

Inside a worker, use `self` (instead of `window`) to represent the current scope.