Future of Emscripten Output Modularization

Sam Clegg

Where we are today

Default

Emscripten by default outputs plain JS code that can be used in a variety of ways. You can include it directly within another JS file or on a web page or you can wrap it manually to turn it into some kind of JS module. When used in this fashion native and JS symbols from the emscripten compiled code are available directly as top level symbols. For Wasm symbols we generate wrappers that are replaced/modified once the module is loaded. Attempting to call any of these functions before initialization is complete will fail/assert. Emscripten internals are all directly exposed in the global scope, and because of this only one module can exist in that scope at once.

`-sMODULARIZE`

This option wraps the generated into a promise returning function and hides all the emscripten internals from the user. The resulting code is compatible with the CommonJS module system.

`-sEXPORT_ES6`

This option builds on `-sMODULARIZE` and produces an ES6 module that exports a single promise-returning function. The promise resolves with the module object.

`-sMODULARIZE_INSTANCE`

This option no longer exists. When it did, it used to export all module members individually, rather than exporting a promise. As in the default case these would assert/fail if called too early.

Where we would like to be

Goals:

1. Emscripten internals should be hidden/inaccessible in all cases.
2. Emscripten modules should not interfere with each other on the same page.
3. It should be easy for users to generate valid ES6 modules.
4. Output should be consumable by bundlers

The JS community has been transitioning to ES6 modules for many years.   Even though they are still not common in web deployments, many projects use bundlers to produce flat JavaScript from a set of ES6 modules. In other words, many teams use them during development and expect all their inputs to be in the form of ES6 modules. This includes a lot of teams at Google, for example anyone using MSS (go/mss).

In an ideal world we would have just a single output format, and it probably makes sense that that would be ES6 modules.   But I fear we don't live in that world, at least not yet.

Plan of action

In order to get to where we want to be, the following steps are proposed:

Step 0: Design new configuration space

Choose a new configuration name that will enable all the options we want to be on by default eventually.

Step 1: Bring back `-sMODULARIZE_INSTANCE`.

We might want to give it a better name.   This mode is likely to be what most users actually want. Most users don't need to instantiate a module more than once so the default should really be a single instance. In this mode only a single variable is set in the global scope which is \`Module\` (or `-sEXPORT_NAME` if set).

This new behavior can be enabled in `-sSTRICT` mode since this hiding of internals is something that we would really like to do by default one day.

Can we think of a better name for this:
- -sMODULARIZE=instance (avoids a new setting)
- -sTYPE=module (a la node)
- -sESM
- -sMODULE
- -sEXPERIMENTAL (will enable this and other features here)

Whichever one we choose will trigger an experimental warning.

How does this interact with embind?
Embind + ES Modules + Bundlers

Step 2: Add a new symbol export method: `-sEXPORT=foo`

We already have a plan this for a new way to export stuff: #8380. In this new scheme native wasm symbols are no longer prefixed with underscore, meaning the exported name matches the Wasm export name. Having the exported names match the Wasm names should make it more likely that we can expose wasm exports directly, and make the job of the bundler simpler.

Since using this new export method will almost certainly be breaking change we can use it as a signal that the user is opting into more modern behaviors, including `-sMODULARIZE_INSTANCE.` Should we also default to `-sEXPORT_ES6` in this mode?

Open questions

Should we expose module exports directly (à la `-sMODULARIZE_INSTANCE`)? - YES

This can make the dependency graph easier to analyze for bundlers. It's also simple to read as it makes it very easy to see exactly which symbols are being imported from the emscripten-generated module.   We would need to separately export some kind of `ready()` promise so that the user would know when it was safe to call the other exports.

I believe this is how ES6 Wasm module integration works. If so, it would be good to follow that pattern.

Should we make -`sEXPORT_ES6` the default when using c?

I hope we can do this. Do we need to continue to support the older/other module formats?

Should we make -`sMODULARIZE` the default?

This depends on how common the non-module use case is going to be going forward.

Can we completely remove the non-`MODULARIZE` path one day?

It sounds like CommonJS output, or vanilla JS output will likely have users for some time.   It would be good to somehow survey our users to find out if this is true and for how long it might be needed.

Should `EXPORT_NAME` default to the output file name?

e.g if I run \`emcc -o foo.js\` should that generate a global \`foo\` object rather than a global \`Module\` object. Note that \`Module\` is a bad name because it starts with an uppercase letter which implies that its a type in JS, and not an instance.

What API should the ES module output provide?

Potential Options