Software

DISCUSSION: JERRYSCRIPT PACKAGE MANAGER

Martijn Thé

martijn.the@intel.com

JerryScript Package Manager

- Sorry, the title is only to draw your attention ☺
- No, but really, we need to figure out how to make it easier to share and reuse code.
- Examples of things worth sharing: well-known and new interfaces, like
  - `console.log/warn/error`
  - `setTimeout/setInterval/etc…`
  - `fetch()`
  - Sensor (W3C draft)
  - Bluetooth (W3C draft)
  - USB (W3C draft)
  - utils (C as well as JS)
    - `bool js_check_and_log_uncaught_error(jerry_value_t val);`
    - `jerry_value_t js_get_property(const jerry_value_t obj_val, const char *name);`
    - etc.

Open sourcing JerryScript module code today

- Put it up on Github
- Hope that someone finds it
- Hope that they will know how to add it into their project correctly, do any magic init dances etc.
  - jerryx_module feature removes a lot of friction here already

Consuming someone's JerryScript module code today

- Try to find something with Google/Baidu/…
- You're lucky: you found something (probably from someone in this room ;)
- Add their code as a submodule to your project
- Hrmm.. They use a different build system than I do. Don't want to get into that, so let's add the sources to my own build, figure out the compiler/linker flags, etc.
- Find out that it drags in a lot of helper functions that are more or less already present in your project, bloating the code size.
- Implement any platform interfaces that the module needs.

The Dream: JerryScript Package Manager

- npm like experience for publishing and consuming (native) modules
- **`npm add @jrs/sensor`**
  - Finds “sensor” package in the repo
  - Adds the sources to your project’s build system
  - Optionally creates a xyz-module-platform.c file with stub functions that the module depends on. You’ll need to write glue code to provide the desired (i.e. to connect to your project’s sensor backend, timer backend, etc.)

Challenges

- Module author will have to design the platform interfaces such that they can be implemented easily on different platforms. Not easy to do, probably means that a module's platform interface won't be very stable in the beginning.
- Build system: probably need the equivalent of node-gyp (or just use node-gyp?). Hooking into each and every project's build system is probably a nightmare.
  - Instead: "standardize" the build set up (but let the project be able to tell what compiler to use, additional project flags, etc.) and produce a static/dynamic library + headers that can be used by and linked into the final binary of the project.
- Modules might bring along .js source as part of the implementation
  - How these are stored and how they are made accessible at runtime is project-specific.
    - `require() / import` would solve that
    - jpm would need to be able to generate list of .JS files that the project needs to include and make available through `require() / import`
- ...

Who wants to take the lead?

- Happy to help but not sure if we are in the right position to pull this off.
- Who volunteers? ;-)

DISCUSSION