Agenda for the:	**special meeting of Ecma TC53 on I²C Topics**
in:	Teleconference
on:	June 28, 2022

Attendance:

Peter Hoddie, Moddable (PH) - Chair
Andy Carle, Moddable (AC)
Donovan Buck (DB)
Josh Siegel, Michigan State University (JS)
Nick Hehr, Betterment (formerly Tessel), working on xs-dev (NH)
Bryan Hughes (BH), Staff frontend engineer at Patreon, creator of Raspi IO
Rick Waldron (RW), creator of Johnny-Five

1 Welcome, opening and meeting logistics

PH: Not a regularly scheduled meeting but still a TC-53 meeting so all the usual practices apply. Expect things to be recorded in notes, and you will have an opportunity to edit those notes.

<Introductions>

2 Review of the Agenda

3 I²C Topics

I2C Backpack for Johnny-Five

PH:

BH: The challenge that we have is that there are two separate use cases that we're targeting Ecma-419 at. One is bare metal/RTOS. The other is non-realtime OSes (RPi, Windows, Linux).

BH: I think of these as very different use cases because of what we want to do with them. In theory, we could very quickly use existing I/O libraries to build an Ecma-419 port on top of.

BH: Being on top of RPi opens up the full JS ecosystem. Everything with Node, web servers, etc. Need to not break the Node.js event loop.

BH: On bare metal you want low-latency. On RPi you really want to be event-driven.

RW: The stack you described is exactly what makes NodeBots so special. We can have a web server that is serving up its own UI for controls or streaming video. And then that webserver can talk over SocketIO to control the hardware. I see a future where Moddable becomes a general platform that can also do that sort of thing. They're not that far off from each other. But, you're right about the mechanisms that allow these things to function as good citizens on each platform. The conflict that you described is really a conflict.

RW: With everything you just said in mind, the solution that Donovan and I discussed seems like a great fit for how the problem has been framed here.

DB: The client-facing API should be consistent for the underlying solution.

BH: And to be fair, I started my discussion from a technical perspective too based on a gut feeling that later turned out to be my philosophical view

RW: Reads on Moddable work so fast, they can be synchronous without issue; as opposed to Node-land, where this would need to rely on a read / readAsync pairing (or the opposite of read / readSync) to be efficient.

RW: Compare the usage of "bare-metal" RTOS vs Linux user space, the latter must adhere to the async model of that runtime. Embedded i2c is so fast that it will not hold up the general execution of the program. In Node, the i2c read _could_ take some time and end up blocking the execution of the program and being inefficient to the expectations of the environment.

PH: I'm not sure I understand the "slowness" described by the concept of blocking I/O for Node on fairly quick devices like the Raspberry Pi. How do we consider what is significant blocking on devices?

BH: While Raspberry Pi devices are seemingly fast, there is more overhead to perform the same operations on a traditional embedded device using an RTOS.

PH: The starting point around the Raspberry Pi conversation was as a "stepping stone" for running ecma 419 on microcontrollers, as the expected target for that specification, versus creating an implementation for Node as an expected target.

BH: Windows IoT is an example target OS as a solution for industrial applications, especially for use on the Raspberry Pi.

AC: There seems to be an underlying assumption that the JS engine should be able to keep up with the hardware interactions of the device, where that might not be true for Node.

BH: If someone chooses a Raspberry Pi, then they are usually opting into the wider Node ecosystem instead of choosing a microcontroller like ESP32, and not using an alternative engine.

RW: Idea will not change the existing ECMA 419 spec. Focused on portability across runtimes, like Node and Moddable XS. Use the real worker agent and shared array buffer API to implement the i2c API for ECMA 419. The userland code will be called the “main process” and send instructions to the “background process” as the worker to perform the heavy lifting of reads and writes of the communication with a sensor. The worker will place data from the sensor into the shared array buffer memory for the main process to read from in a synchronous manner.

PH: This is interesting and theory makes sense for how Atomics are expected to work. This reminds me of W3CSensors and the way that there the reads would always work ahead of the synchronous reads?

RW: This is correct and similar to how the Tessel worked in its architecture for sending instructions from the embedded Linux process to a companion microcontroller to perform the actual hardware communication.

DB: Worried a little bit about determining the size of the shared array buffer.

RW: Make it big

DB: Also considering the freshness of data to make sure the code can guarantee accuracy of the data being read by the main process.

RW: I assumed the user code would consider the frequency of reads and how fresh the data should be.

BH: All reads on the Raspberry Pi (no matter the protocol) are already async, so the background worker would not be needed. If not data if available, could we return undefined?

PH: The IO class pattern specifies reads as being able to return undefined when no data is available. There is difference between i2C and serial communication.

Andy: Reads are usually a transactional process.

BH: There are oddities of the system that could be solved by providing code at the driver level.

PH: we’ve heard some potential solutions. What are some next steps?

DB: I would like to see some real-world usage patterns for i2c from the spec to help provide insight for implementations. If we had a readAsync pattern defined, the implementor could choose to provide it or not. This could be added today to the Rasp-io implementation, but we want to avoid the “wild west” direction of breaking away from the ECMA 419 spec.

BH: Why don’t we only do async? I understand we want more timing control provided by sync calls.

AC: There are more considerations like the size of the engine and complexity of the execution of the program.

PH: I like DB's discussion of taking a look at the API from the callers perspective.

NH: We should clarify the scope of the spec, especially around the broad term of "single board computer"

DB: I don't know how to solve the issue of having a good user experience with the API.

AC: Someone needs to send out this proposal to the mailing list to gather feedback on how that could be used.

PH: We have a baseline to move from and we can open an issue on GitHub to continue this discussion.

DB: I'll open the issue, just won't offer a solution