NodeBots - The Rise of JS Robotics

FRIDAY, JULY 13, 2012

On September 7, 2010, I committed the very first implementation of node-serialport with a disclaimer in the README of "do not use". I had extracted the library from a larger project for communicating to various wireless spectrums (open and propertiary, think X10) for physical monitoring via sensors. Originally, I was using the very robust and established pyserial library to much success, but something about the way I was interacting with my sensors through pyserial felt awkward to me. I will openly admit that this feeling was more than anything due to me than any misgiving from the library as it is a great library.

It was actually during this time that the JSConf EU team was soliciting topics for presentation and one from Nikolai Onken and Jörn Zaefferer of Uxebu popped up about Robotic JavaScript. In the proposal, they pitched that the thesis of their talk would be creating a world in which controlling devices could be as simple as:

```
$("livingroom").bind("motion", function() {
  $(this).find("lights").brightness("75%").dimAfter("120s");
});
```

Now, that struck a (good) nerve for me.

By modeling real world objects and actions as chainable, evented processes felt almost natural. For my specific use case, I was using the serial port as a response handler, the perfect example of this is “When the front door opens, toggle all lights”. The close similarity to jQuery had the added benefit that as we increased in size, we wouldn’t need a hardcore programmer that understood serial ports AND Python, but just JavaScript and a rough comprehension of jQuery. I felt like there was something worthwhile here and so I sat down to code.

While programming the first revision of node-serialport I realized a couple amazing things that I wanted to share. I see the same questions every single time someone posts a new project using node-serialport to the world and I would like to provide my perspective of the answers. I prefer to use node.js for programming arduinos, wireless stacks, printers, toasters, etc. (anything through the serial port), for some very definitive reasons the most direct reason is simplicity. The node module system and the very thin native bindings allows me to develop new low level functionality very swiftly without having to involve too much node specific code segments. This also affords me, the library maintainer, the ability to leverage the larger world of knowledge about how to program, debug, and test serial ports directly and not have to spend days working on getting it to work with node.js. The layer to connect straight C++ code with node.js is so thin, it is nearly trivial to implement (and implement properly) which is why I fell in love with it. I know people have a similar love (or strong hate) for Python and Ruby’s native binding, but, personally, am of the opinion that the syntax similarities between C and JavaScript make it a lot smoother of a process for mentally visualize the binding despite spanning two languages. If you look at the low level C++ of node-serialport you can probably fully understand it even if you have only ever programmed JavaScript. That can both be powerful and wrought for confusion, admittedly, but for me it just clicked and made writing the low level serial port code an absolute joy. Your mileage may vary, but at least it is worth investigating.

Once the low level code was written, one of the main reasons I have continued to develop on top of node.js for my serial port needs is that desire to programmatically describe my problem domain in a manner that fits it. As mentioned, my system, and most systems using node-serialport (arduino, etc), are inherently reactionary in that they wait for some event or data and then do some thing. This is arguably JavaScript's bread and butter, it is how most people are trained already to think when they think in node and JavaScript. Can this be done in other languages, sure, but the simplicity of the implementation all the way up from the low-level C++ to the interfacing code to even the event handling makes for something very capable and very understandable with little effort. In the recent comments about the awesome node.js wifi-extending robot, an individual asked why not clojure or scala or python or ruby and my most direct answer is that the volume of code to create the reactionary system (including eventmachine/twisted/redis/pubsub/etc) quick grows beyond simple hobbyist experimentation. And lets be completely honest here, at least for now - arduino, x10, raspberry pi, etc are all at best in the hobbyist domain especially under the context of "software developers playing with hardware". I would actually apply the theory of increasing developer happiness from Ruby as the reason to use node.js for building robot PROTOTYPES. It is lightweight, simple, and easy to make complex reactionary systems with little effort. Best of all it is JavaScript, so you have no risk of falling madly in love with your initial prototype and will eventually rewrite in something 'more production-worthy'. Or not.

As an example of how lightweight node.js hardware development can be, I would offer up the amazing work done by Rick Waldron in Johnny-Five, take a look at the source code for processing events from an acceleromator it is easily readable, understandable and just over 10 lines of code. Controlling a servo, also just over 10 lines of code and just as readable and comprehensible. With node-serialport and firmata (which kicks butt, props to Julian Guatier) the physical world is an oyster for JS developers, if you don't believe, three of the best presentations at NodeConf 2012 were ones using node.js to control physical devices. Also review his slides.

When I look at all the people working with node-serialport, the one thing I see across the board is that the "ease" of programming JavaScript, the evented model, and how real world devices exist and operate – all of that seems to blend nicely. Another fine example is the tmpad, a DIY MIDI pad from Elijah Insua. The source code is make a midi pad is almost nothing and that is a good thing. Elijah was able to create somethign amazing with little effort, which is why I believe we are on the cusp of the broader adoption of JS as a leading language for building and controlling hobbyist hardware systems.

I say this because every day I hear more people making newer, brighter, more robotic things with node-serialport. I recently came across BreakoutJS which is nobly attempting to make it even easier to interoperate with sensors and devices. This makes me happy and it should make you happy as well, if just to witness the energy and excitement. For me, the hobbyist hardware domain is more real than programming code, it is crossing of the boundary from abstract to real-life and has a massive potential to do a lot of good for a lot of people. Should it eventually be coupled with a strong AI system like a Clojure? Absolutely, but for now, play and experimentation are the key. If you must demand starting with a lisp, then just use ClojureScript and viola, best of both worlds!

An Announcement

One of the things that has been lacking from node-serialport was support for Windows. I recently found that Joe Ferner was working on a fork that would (among other things) add Windows support. In the full beauty of social, open source development, we synced up and are happy to present for your cross-platform robot creation plans of world domination:

NODE-SERIALPORT 1.0.0

We have tried to maintain the same API as the original node-serialport system, but add a couple of other accessible items. We have vetted this version with people that we had email addresses of and knew were building on top of node-serialport, if you were building something and run into an issue with this new version, please register a ticket. We will try to resolve it as quickly as possible. We are quite excited about this release and happy to say that after almost 2 years of watching the JS robotics community grow, I can officially say: “Welcome your robotic javascript overlords. Better yet, program them (even on Windows)!”

If you want to talk about things you are doing with node-serialport, arduino, anything robotics and beyond, please come join us in IRC on the freenode channel #robotjs.

```
npm i serialport
```

Happy Hacking,

@voodootikigod

Oh, and get off the Internet and go change the world!

SHARE THIS POST

AUTHOR

Chris Williams