Programming the Internet of Things Using Node.js* and HTML5

Michael McCool
Ramesh Peri
Rex St John

O’Reilly Solid Conference, June 23, 2015
Agenda

- Internet of Things (10m)
- Set up the Thing (45m)
- Build the Thing (10m)
- Program the Thing (30+10m)
- Network the Thing (50+25m)
- Connect the Thing (10m)
- Conclusion (5m)
Agenda

• Internet of Things (10m)
• Set up the Thing
• Build the Thing
• Program the Thing
• Network the Thing
• Connect the Thing
• Conclusion
The Internet of Things is...
Basically...

Things...

Devices with embedded computers

- Often without displays
- Reading physical sensors
- Controlling physical actuators
- Limited power, memory, compute

...on the Internet.

Communicating globally

- Variety of networking protocols
- Interface to existing infrastructure
- Autonomous operation modes
- Coordinated operation modes
- Part of a system

Think “system”, not “device”
IoT: Examples and Use Cases

**Industrial and Municipal**
- Factory equipment monitoring
- HVAC system monitoring/control
- Lighting systems
- Security
- Maintenance (wearable)
- Point of Sale
- Inventory tracking
- Precision agriculture
- Energy consumption management

**Consumer and Mobile**
- Health monitoring (wearable)
- Information access (wearable)
- Home security
- Delivery management
- Automotive monitoring
- Family status monitoring (wearable)
- Home maintenance
- Personal communication (wearable)
- Energy consumption management
Internet of Things (IoT) Systems

Endpoint

Gateway

Wi-Fi

Gateway

LTE

Human Interface

Analysis (Server)

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Prototypical IoT System: Smart Light

A “smart light” is a prototypical IoT system: a simple device, but adding intelligence to it enables major efficiencies and opportunities.

Light Detail
- Light ID: LP340-0106
- Location: (50.759 N, 122.081 W)
- Control System: Moon Island PLC
- Lamp Type: 1 x Dali Cosmo White
- Power (W): 140
- Installation Date: 8/1/2011
- Model: GE Evolve LED
- Serial Number: KVM320-001
- Hardware ID: 05:01: AD:3E:44:00
- Status: Light Expiration Imminent

Schedule Service
A Smart Light Prototype

Using JavaScript* (in the form of both Node.js* and HTML5) and an Intel® Edison compute module we’ll be building a “smart light” prototype.
Why JavaScript* for IoT?

JavaScript* is in wide use

- Large numbers of web developers are familiar with it
- Well-documented with a strong ecosystem
- Already standardized and with multiple implementations

JavaScript is consistent with web programming and portable mobile apps

- It is used by HTML5, and HTML5 is useful for developing UI “companion apps” for IoT devices

JavaScript is well-suited to embedded device programming

- Supports asynchronous function calls and I/O
- Asynchronous calls are useful for event-driven hardware programming
- The Node.js* engine in particular has many useful features for both web services and embedded devices

JavaScript engines are high-performance

- Chrome* V8 is nearly as fast as C/C++! Fastest “high-productivity” language available.
- Recent developments, such as SIMD.js, even expose high-performance computation features
Why Node.js* for IoT?

Strong ecosystem and package management system
• Over a hundred thousand packages available via the npm package manager (www.npmjs.org)

Programming model well-suited to embedded devices as well as servers
• Event-driven asynchronous programming model, support for asynchronous functions
• Lack of explicit event loop means transparent power state management can be implemented
• Good support for interfacing to native C/C++ libraries

Community is already using Node.js* for embedded devices and robotics
• For example: http://nodebots.io/, Firmata, Cylon, JohnnyFive, ...

Web services can (of course!) also be built with Node.js
• A complete end-to-end endpoint/gateway/mobile/browser/server IoT solution is possible

Good documentation
• Both online and in book form, although to date mostly focused on web services
**Node.js** vs. **HTML5 JavaScript**

**Node.js**
JavaScript with bindings to C/C++ libraries and OS services

- Designed for “Server side”
  - Headless (no UI by default) services
- Asynchronous and event-driven
  - Nonblocking asynchronous I/O
- C++ Bindings (can call C++ libs)
- Communicate over variety of protocols
  - Including but not limited to http
- No built-in sandbox; **focus on functionality**
- Community driven module repository
  - Wide variety of modules available, including support for parallelism, computer vision, etc.
  - Automated download and install
  - Modules can include native components

**Browser (HTML5) JavaScript**
JavaScript with bindings to an HTML5 document/view

- Designed for “Client side”
  - Generate and control a *user interface* (UI)
  - Alter HTML5 document content
  - Control a browser (open panes, etc.)
- Communicates asynchronously using http
- Limited/semi-standardized hardware support
  - Depends on platform/browser
  - Ex: webcam, accelerometer, GPS
- Sandboxed; **focus on security**
  - Specific set of standardized built-in libraries
  - Other JavaScript-only libraries can be downloaded via http dynamically

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.*
JavaScript* for IoT

Developer View

Intel® XDK IoT Edition
Development System

Optimization Notice

Copyright © 2015, Intel Corporation. All rights reserved. *Other names and brands may be claimed as the property of others.
Intel® XDK IoT Edition – Node.js* Application Development

- Edit Node.js* app
- Send app to device
- Run app remotely
- Remote debug
- Also provides convenient access to command-line interface on device via ssh or serial port
Agenda

• Internet of Things
• Set up the Thing (45m)
• Build the Thing
• Program the Thing
• Network the Thing
• Connect the Thing
• Conclusion
Setting Up (45 min)

Setup Instructions specific to this lab: http://bit.ly/xdk-setup

- These instructions focus on only what you need right now
- Please group yourselves by operating system
- Start by copying folder from USB sticks being passed around to your desktop
- Then look in “Course/Install” directory specific to your OS

Summary: install drivers for the Edison on your computer, update its operating system and software, set up a development tool (the Intel® XDK IoT Edition) specific to Node.js on the Edison.

Network: SSID is “Solid”; no password
Agenda

- Internet of Things
- Set up the Thing
- Build the Thing (10m)
- Program the Thing
- Network the Thing
- Connect the Thing
- Conclusion
Hardware Kit

Intel® Edison Development Kit
• Intel® Atom™ dual-core processor
• Arduino* compatible hardware interface adapter
• Yocto* Linux* based Intel® IoT Development Kit software

Seeed* Studio Grove Starter Kit Plus (Intel® IoT Edition)
• Base shield and collection of useful modules and cables

PLUS
• Extra micro-USB cable
• 12V power adapter
Seeed* Grove Starter Kit Plus Contents

- Set of useful Seeed* Grove modules with base shield, cables
- Micro USB-B to USB-A cable
- Not used for Edison:
  - RJ45 network cable
  - FTDI USB-to-serial cable
Hardware Assembly

- Intel® Edison module and Seeed* Grove shield attached to Arduino* adapter
- Push button on D2
- LED on D3
- Light sensor on A0
- Rotation sensor on A1
- 7-15V power connected
- TWO micro USB cables to computer
- USB selection switch toward micro USB host port
Agenda

- Internet of Things
- Set up the Thing
- Build the Thing
- Program the Thing (30+10m)
- Network the Thing
- Connect the Thing
- Conclusion
Introduction to JavaScript

C-like syntax, but ";" are optional.

Strings:

'this is a string'
"so is this"

Declaring a variable:

var my_variable;

Points to note:

- Types are dynamic
- All numbers are floats
- **Scope is function**, not block
- Scope is generally static, except for 'this'

Objects and associative arrays are the same:

```javascript
dog['ear'] = 'up';
dog.ear = 'up';
```

Object prototypes cloned with “new” operator:

```javascript
dog = new Dog();
```

Functions are values. The 'this' variable can be used to refer to the current object:

```javascript
dog.raise_ear = function() {
    this.ear = 'up';
};
```

JSON-like syntax for object properties:

```javascript
dog = {
    ear: 'up';
    raise_ear: function (...) {...};
}
```
**Embedded Hardware Interfaces**

**GPIO:** General-purpose digital I/O. Can read or write 0's or 1's. Need to set direction (input or output).

**Analog:** Input. Senses a voltage between 0 and reference value (usually the power supply value).

**PWM:** Pulse-width modulation output. Rapidly switches between 0 and the power supply value. If averaged, approximates an analog output. Can also be used (if set to the right frequency) to control RC servos.

**I2C:** Two-wire (clock and data) half-duplex serial interface using a shared bus. Devices on the bus have fixed address and a set of registers that can be read or written. Each device has its own registers and interface specification. Speed: 100KHz and 400KHz.

**SPI:** 3 or 4-wire (clock, Tx and Rx data, chip select) full-duplex serial interface. Higher performance than I2C (up to 25MHz) but more complex wiring.
Embedded Hardware Interfaces

**GPIO:** General-purpose digital I/O. Can read or write 0’s or 1’s. Need to set direction (input or output).

**Analog:** Input. Senses a voltage between 0 and reference value (usually the power supply value).

**PWM:** Pulse-width modulation output. Rapidly switches between 0 and the power supply value. If averaged, approximates an analog output. Can also be used (if set to the right frequency) to control RC servos.

**I2C:** Two-wire (clock and data) half-duplex serial interface using a shared bus. Devices on the bus have fixed address and a set of registers that can be read or written. Each device has its own registers and interface specification. Speed: 100KHz and 400KHz.

**SPI:** 3 or 4-wire (clock, Tx and Rx data, chip select) full-duplex serial interface. Higher performance than I2C (up to 25MHz) but more complex wiring.
Hardware Interface Libraries in Node.js*

**Intel-Developed**

**mraa:** Open source library for Intel® Galileo and Intel® Edison boards. Low-level access to GPIOs, PWM, and Analog IO ports. Developed by Intel Developer Relations.

**upm:** Open source, high level library for specific sensors and actuators that use mraa. Developed by Intel Developer Relations.

**iot-io:** Open source library for Intel Galileo/Edison boards. Uses an API similar to the Arduino* API. Also provides remote access via RPC. Developed by Intel Labs China.

**Third Party**

**johnny-five:** Open source I/O library with plugins for various boards including Intel Galileo and Edison boards. The Intel board support uses direct interfacing rather than going through Firmata when using the galileo-io adapter module.

**cylon:** Object-oriented, asynchronous open-source portable robotics framework.

**onoff:** Open source I/O library that can be used to manipulate ‘raw’ GPIO/PWM via sysfs interface.

**i2c:** Node.js* I2C library using sysfs.

**NOTE:** Using sysfs requires understanding low-level details of the board.
Hardware Interface Libraries in Node.js*

**Intel-Developed**

**mraa:** Open source library for Intel® Galileo and Intel® Edison boards. Low-level access to GPIOs, PWM, and Analog IO ports. Developed by Intel Developer Relations.

**upm:** Open source, high level library for specific sensors and actuators that use mraa. Developed by Intel Developer Relations.

**iot-io:** Open source library for Intel Galileo/Edison boards. Uses an API similar to the Arduino* API. Also provides remote access via RPC. Developed by Intel Labs China.

**Third Party**

**johnny-five:** Open source I/O library with plugins for various boards including Intel Galileo and Edison boards. The Intel board support uses direct interfacing rather than going through Firmata when using the galileo-io adapter module.

**cylon:** Object-oriented, asynchronous open-source portable robotics framework.

**onoff:** Open source I/O library that can be used to manipulate 'raw' GPIO/PWM via sysfs interface.

**i2c:** Node.js* I2C library using sysfs.

**NOTE:** Using sysfs requires understanding low-level details of the board.
MRAA Setup

```javascript
var mraa = require("mraa");

var button = new mraa.Gpio(2);
button.dir(mraa.DIR_IN);

var led = new mraa.Pwm(3);
led.period(1);
led.enable(true);

var knob = new mraa.Aio(1);
```

Load the module

Enable GPIO 2 (D2)

Put it into input mode

Configure PWM output on D3

Set period to 1ms

Turn on PWM output

Enable Analog input on A1
**MRAA Hello World**

```javascript
setInterval(function() {
    var push = button.read();
    if (push) {
        led.write(
            knob.readFloat()
        );
    }
}, 100); // every 0.1s (100ms)
```

**Do something periodically**

- Read state of button
- If the button is pressed
- Read the knob state and write it to the LED

**Repeat every 0.1s (100ms)**
MRAA Hello World

```javascript
setInterval(function() {
    var push = button.read();
    if (push) {
        led.write(
            knob.readFloat()
        );
    }
}, 100);  // every 0.1s (100ms)
```

Do something periodically
Read state of button
If the button is pressed
Read the knob state and
write it to the LED
Repeat every 0.1s (100ms)
Assignment 1 (5 min; SKIP)

```javascript
setInterval(function() {
  var push = button.read();
  if (push) {
    led.write(
      knob.readFloat()
    );
  }
}, 100); // every 0.1s (100ms)
```

This code really just updates the state of the LED based on the knob when the button is pushed, and ignores it otherwise.

The button does not actually turn the light on and off.

*Modify this part of the code so that the light is only on when the button is pressed.*
Assignment 1: Solution

```
setInterval(function() {
  var push = button.read();
  if (push) {
    led.write(
      knob.readFloat()
    );
  } else {
    led.write( 0 );
  }
}, 100); // every 0.1s (100ms)
```

Do something periodically
Read state of button
If the button is pressed
  Read the knob state and write it to the LED
Otherwise
  Turn off the light
Repeat every 0.1s (100ms)
Assignment 2 (5 min; SKIP)

```javascript
setInterval(function() {
    var push = button.read();
    if (push) {
        led.write(
            knob.readFloat()
        );
    } else {
        led.write( 0 );
    }
}, 100); // every 0.1s (100ms)
```

We don't actually want to hold the button down all the time.

*Modify this code further so the button toggles the light state on and off.*
Assignment 2: Solution

```javascript
var light_state = 0;
var last_push = button.read();
setInterval(function() {
  var push = button.read()
  if (push && !last_push) {
    light_state = !light_state;
  }
  last_push = push;
  if (light_state) {
    led.write( knob.readFloat() );
  } else {
    led.write( 0 );
  }
}, 100); // every 0.1s (100ms)
```

Keep track of light power-on state

Do something periodically

Read state of button

If the button is pressed

  Toggle the light power state

If the light power state is true

  Read the knob state and write it to the LED

Otherwise

  Turn off the light

Repeat every 0.1s (100ms)
Assignment 3 (10 min)

```javascript
var light_state = 0;
var last_push = button.read();
setInterval(function() {
    var push = button.read();
    if (push && !last_push) {
        light_state = !light_state;
    }
    last_push = push;
    if (light_state) {
        led.write( knob.readFloat() );
    } else {
        led.write( 0 );
    }
}, 100); // every 0.1s (100ms)
```

*Modulate actual light intensity by ambient light so the light is dimmer in dimmer light to save power.*
Assignment 3: Solution

```javascript
var ambient = new mraa.Aio(0);
var light_state = 0;
var last_push = button.read();
setInterval(function() {
    var push = button.read();
    if (push && !last_push) {
        light_state = !light_state;
    }
    last_push = push;
    if (light_state) {
        led.write( knob.readFloat() * ambient.readFloat() );
    } else {
        led.write( 0 );
    }
}, 100); // every 0.1s (100ms)
```

Need to declare “ambient” analog input on analog pin 0.

Then multiply light power by ambient intensity.
Summary: Programming the Thing

• Learned hardware interfaces for GPIOs, PWMs, and Analog Inputs
• Learned how to control these types of interfaces from MRAA
• Implemented a program that gives a thing a useful default behavior:
  - Toggle a light on and off when a button is pressed
  - Modulate intensity with a knob
  - Modulate intensity based on ambient light
• Have edited and loaded code with the Intel® XDK
  - Lets us debug the code
  - Makes code persistent (when board reboots it will be run automatically)
Agenda

• Internet of Things
• Set up the Thing
• Build the Thing
• Program the Thing
• Network the Thing (50+25m)
• Connect the Thing
• Conclusion
Networking and Communication

• Physical:
  UART: Serial communication
  Wireless: Bluetooth®, Wi-Fi*, ZigBee*
  Protocols: TCP, UDP

• Modules:
  net: TCP sockets
  http: web protocols
  socket.io/shoe: websockets
  dnode: remote async function invocation
  mqtt: status and configuration
User Interfaces for Things

- Things don’t often have displays
- But IoT devices do have network connections (by definition)
- Users can be assumed to have web browsers and smartphones

**HTML5 can be used to build portable graphical UIs:**

- Can be installable “hybrid” apps
- Can be served directly from a thing
- Can be served by a cloud service
- Can run on smartphones or tablets
- Can run in browsers on larger computers
Letting a Thing Serve its Own User Interface

• In some cases, *if only a few users will ever connect to a smart thing*, it can make sense for that thing to serve its own HTML interface.

• *We will also show how to use dnode and websockets to export a object-oriented, asynchronous API to an HTML user interface.*

• A dnode API can even have callbacks, allowing very general communication between code in the HTML and on the thing.
Assignment 4 (3h; SKIP): Add a Web Interface

• Device should serve its own web UI
• Web UI should allow turning the light on and off
• Web UI should display the current state of the light (it should be updated if the physical button is pressed)
• Use websockets to communicate between the Web UI and the device
System Architecture

Interface Client (e.g., Browser) -> Smart Thing
- http request
- http response (JS + HTML)
- web socket/dnode request
- dnode API connection
Basic HTML5 Template for Smart Light UI

```html
<!doctype html>
<html>
  <head>
    <title>Light Control</title>
    <script src="js/ext/jquery.js"></script>
  </head>
  <body>
    <img src="images/off.jpg" id="light_button">
    <script src="js/gen/bundle.js"></script>
  </body>
</html>
```

- Simplified doctype for HTML5
- We'll use jQuery to get handles on objects
- This image is set by default to a light bulb in the “off” state, but we give it an id so we can manipulate it later
- The “bundle.js” script will be created with “browserify” from Node.js code and will contain our application logic
User Interface Client Script (client.js)

```javascript
var dnode = require("dnode");
var shoe = require("shoe");

var sock = shoe("/api");
```

Get some modules. We will use the “browserify” Node.js app to package these for the browser.

Connect to server websocket.

On device:

```bash
npm install -g browserify  # first time only

cd /node_app_slot/web/js

browserify client.js -o gen/bundle.js
```

On host (in web/js/gen directory of XDK project):

```bash
scp root@edison.local:/node_app_slot/web/js/gen/bundle.js bundle.js
```
var light_button = $('#light_button');

function set_light_state(state) {
  if (state) {
    console.log("turning light on");
    light_button.attr("src","images/on.jpg");
  } else {
    console.log("turning light off");
    light_button.attr("src","images/off.jpg");
  }
}
User Interface Client Script (client.js)

```javascript
var dn = dnode();

dn.on("remote", function (remote) {
    remote.state(set_light_state);
    remote.set_notify(set_light_state);

    light_button.click(function() {
        remote.toggle(set_light_state);
    });
});

sock.pipe(dn).pipe(sock);
```

Get dnode API and when page loads...

Get current light state

Set up notifier callback

update light state.

Indicate what to do when light button clicked.

Connect dnode server to web socket.
User Interface Client Script (client.js)

```javascript
var dn = dnode();
dn.on("remote", function (remote) {
    remote.state(set_light_state);
    remote.set_notify(set_light_state);
    light_button.click(function() {
        remote.toggle(set_light_state);
    });
});
sock.pipe(dn).pipe(sock);
```

- Get dnode API and when page loads...
- Get current light state
- Set up notifier callback update light state.
- Indicate what to do when light button clicked.
- Connect dnode server to web socket.
User Interface Client Script (client.js)

```javascript
var dn = dnode();

dn.on("remote", function (remote) {
    remote.state(set_light_state);

    remote.set_notify(set_light_state);

    light_button.click(function() {
        remote.toggle(set_light_state);
    });
});

sock.pipe(dn).pipe(sock);
```

Get dnode API and when page loads...
Get current light state

Set up notifier callback update light state.

Indicate what to do when light button clicked.

Connect dnode server to web socket.
User Interface Client Script (client.js)

```javascript
var dn = dnode();

dn.on("remote", function (remote) {
    remote.state(set_light_state);

    remote.set_notify(set_light_state);

    light_button.click(function() {
        remote.toggle(set_light_state);
    });

});

sock.pipe(dn).pipe(sock);
```

Get dnode API and when page loads...

Get current light state

Set up notifier callback update light state.

Indicate what to do when light button clicked.

Connect dnode server to web socket.
var http = require("http");
var ecstatic = require("ecstatic")
  (__dirname + "/web");

var dnode = require("dnode");
var shoe = require("shoe");

var notify_light = null;

Basic web protocol support
Basic web server that can serve static web resources to client.

Remote function call service
Web socket wrapper/emulator.

Keyword: Callback to notify client of changes in state; “null” value means it has not been initialized by client yet.
var api = {
    toggle: function(callback) {
        light_state = !light_state;
        callback(light_state);
    },
    state: function(callback) {
        callback(light_state);
    },
    set_notify: function(callback) {
        notify_light = callback;
    }
};

Specify the API interfaces.
Toggle light state, then return current state in a callback.
Read the current value of the light without changing it.
Register a callback to call if state changes.
Thing Behavior and Server Script (main.js)

```javascript
var serv =
  http.createServer(ecstatic);
serv.listen(8080);

var sock = shoe(
  function (stream) {
    var d = dnode(api);
    stream.pipe(d).pipe(stream);
  }
);
sock.install(serv,"/api");
```

Start up a simple static web server.

Listen to a test port (note we use port 8080 rather than the standard 80).

Start up a web socket server and install the RPC API on it using dnode.
var mraa = require("mraa");

var button = new mraa.Gpio(2);
button.dir(mraa.DIR_IN);

var led = new mraa.Pwm(3);
led.period(1);
led.enable(true);

var ambient = new mraa.Aio(0);
var knob = new mraa.Aio(1);

var light_state = false;
var last_push = button.read();
A slight change here.
Can you spot it?

```
setInterval(function() {
    var push = button.read();
    if (push && !last_push) {
        light_state = !light_state;
        if (notify_light) {
            notify_light(light_state);
        }
    }
    last_push = push;
    if (light_state) {
        led.write( knob.readFloat() * ambient.readFloat() );
    } else {
        led.write( 0 );
    }
}, 100);
```
Thing Behavior and Server Script (main.js)

```
setInterval(function() {
  var push = button.read();
  if (push && !last_push) {
    light_state = !light_state;
    if (notify_light) {
      notify_light(light_state);
    }
  }
  last_push = push;
  if (light_state) {
    led.write( knob.readFloat() * ambient.readFloat() );
  } else {
    led.write( 0 );
  }
}, 100);
```

Call any non-null notifier callback when light state changes.
Assignment 5 (25 min): Add a Blink Control

• Modify the provided example code (A4) to add a new web control to blink the light automatically.

This might be used for emergency signaling, for example.

• Extend the dnode API with a new state variable, blink_state (mimic the light_state state variable setup)

• Add a new button on the web page using the provided “blink_on.png” and “blink_off.png” images (mimic the code for the light image...)

• Modify the hardware control so that when the blink_state state variable is 1, the light should automatically cycle between on and off with a period of 2s (1s on, 1s off), but only if light_state is also 1

• The image of the light on the web page should also blink along with the physical light
Assignment 5: Solution

• For the solution please see the files in the A5 directory
However... There is a Problem

• Unfortunately, a single notify callback is only sufficient for a single UI client
A Single Registered Callback is Insufficient

• If multiple clients are used, the second callback registration overwrites the first one
A Single Registered Callback is Insufficient

- If multiple clients are used, the second callback registration overwrites the first one.
Solution: Multiple Registered Callbacks

• This leads to the “publish-subscribe” pattern
• A single “publish” event triggers multiple calls to “subscribers”
Publish-Subscribe

var subscribers = {};

function subscribe(cb, id) {
    subscribers[id] = cb;
}

function unsubscribe(id) {
    delete subscribers[id];
}
function publish(data) {
    for (id in subscribers) {
        subscribers[id](data);
    }
}
module.exports = function() {

    this.subs = {};

    this.subscribe = function(cb, id) {
        this.subs[id] = cb;
    }

    this.unsubscribe = function(id) {
        delete this.subs[id];
    }

    this.publish = function() {
        for (id in this.subs) {
            this.subs[id].apply(arguments);
        }
    }

};
Automatic Unsubscribe on Disconnect

```javascript
pubsub = require("./pubsub.js");
hub = new pubsub();

var d = dnode(function(client,conn) {
    ...
    this.subscribe = function(cb,id) {
        hub.subscribe(cb,id);
        conn.on('end', function() {
            hub.unsubscribe(id);
        });
    }
});
stream.pipe(d).pipe(stream);
```

Create a pub-sub hub

Create a dnode interface

Subscribe function automatically unsubscribes when connection ends

Connect to the socket
Automatic Unsubscribe on Disconnect

```javascript
pubsub = require("./pubsub.js");
hub = new pubsub();

var d = dnode(function(client, conn) {
    this.subscribe = function(cb, id) {
        hub.subscribe(cb, id);
        conn.on('end', function() {
            hub.unsubscribe(id);
        });
    }
});

stream.pipe(d).pipe(stream);
```

Create a pub-sub hub

Create a dnode interface

Subscribe function automatically unsubscribes when connection ends

Connect to the socket
Assignment 6 (15 min; SKIP): Use Pub/Sub

- Replace notify callbacks with pub/sub subscriptions for both light and blink update events
- Result should look the same but will properly support multiple clients
- A pubsub.js library has been provided in A5...

Solution:
- Look in A6
Periodic Publishing → Data Streams

- Generalized publish/subscribe model can also be used to stream data
- Here we have augmented the web interface with some graphs and dials that are updated regularly with a “status update” stream
- Still has a “notify” subscription for button push events
- All graphs and dials are implemented using HTML5 “canvas” functionality
Assignment 7 (3h; SKIP): Data Streaming and Viz

- Add status subscription hub and API
- Average down the knob, light, and led state data and publish status data every second
- Use meters to display the status data
- Add a strip chart to display the status over time
- Update the light display to show the apparent brightness before ambient adjustment

Solution is in A7.
Assignment 8 (10 min; SKIP): Add Ambient Sensor

• Actually, the previous picture is not really what you see when you run the A7 code

• The “ambient” meter is missing, and it's not shown on the chart

• Modify A7 to support the ambient sensor, following the code used for the knob.

Solution: Please see the files in the A8 directory
Agenda

- Internet of Things
- Set up the Thing
- Build the Thing
- Program the Thing
- Network the Thing
- Connect the Thing (10m)
- Conclusion
Web Services

Server (using Express)

```
// Create Express application object
var express = require('express');
var app = express();

// Return some dynamic content
var time = require('time');
app.get('/time.txt', function(req, res){
    var now = time.Date();
    res.send('The time is ' + now.getHours() + ':
             + now.getMinutes());
});

// Start listening for connections
var server = app.listen(8080, function()
    console.log('Listening on port %d',
                server.address().port);
});
```

Client (using Request)

```
// Get the “request” module
var request = require('request');

// Make a request to a web server
request('http://192.168.0.3/time.txt:8080',
        function (error, response, body) {
            if (!error && response.statusCode == 200) {
                console.log(body) // Print the time
            }
        });
```
**MQTT Server using Mosca and MongoDB**

```javascript
var mosca = require('mosca');

var ascoltatore = {
  // Using ascoltatore backend
  type: 'mongo',
  url: 'mongodb://localhost:27017/mqtt',
  pubsubCollection: 'ascoltatori',
  mongo: {};
};

var settings = {
  port: 1883,
  backend: ascoltatore
};

var server = new mosca.Server(settings);

server.on('ready', setup);
// Called when server is ready
function setup() {
  console.log('Mosca is up');
}

// Called when client connects
server.on('clientConnected',
  function(client) {
    console.log('client connected',
               client.id);
  });

// Called when message is received
server.on('published',
  function(packet, client) {
    console.log('Published',
               packet.payload);
  });
```
```javascript
var mqtt = require('mqtt');

// Create an MQTT client (here using static IP of server)
client = mqtt.createClient(1883, '192.168.0.2');

// Indicate what topics we care about
client.subscribe('presence');

// Respond to message on subscribed topic(s)
client.on('message', function(topic, message) {
    console.log(topic, ":", message);
});

// Exit after 10s whether or not we get a message
setTimeout(function() {
    client.end();
}, 10000)
```

```
var mqtt = require('mqtt');

// Create an MQTT client (here using static IP of server)
client = mqtt.createClient(1883, '192.168.0.2');

// Indicate what topics we care about
client.subscribe('presence');

// Publish a message
client.publish('presence', 'Hello mqtt');
```
Extend Light to Publish/Subscribe with MQTT

Publish

```javascript
var mqtt = require('mqtt')

// Create an MQTT client
client = mqtt.createClient(1883, '192.168.0.2');

// Get my hostname
var os = require("os");
var hn = os.hostname();

// Check knob periodically, publish state when it changes
var knob_state = knob.readFloat();
setInterval(function() {
    var new_knob_state = knob.readFloat();
    if (new_knob_state != knob_state) {
        knob_state = new_knob_state;
        client.publish('light/' + hn + '/knob', knob_state);
    }
},100);
```

Subscribe

```javascript
var mqtt = require('mqtt')

// Create an MQTT client
client = mqtt.createClient(1883, '192.168.0.2');

// Indicate what topics we care about (anything about any light)
client.subscribe('light/#');

// Respond to any messages on subscribed topic(s)
client.on('message',
    function (topic, message) {
        console.log(topic,": ",message);
    }
);

// Hang around until killed
setInterval(function() {
    console.log("still listening");
},10000)
```
Streaming Data to the Cloud with AWS*Kinesis*

```javascript
var AWS = require('aws-sdk');
AWS.config.loadFromPath(
  "/home/root/aws_config.json");
var kinesis = new AWS.Kinesis();

hub.subscribe(function(err,status) {
  },"kinesis") ;  // publish ID

Set up AWS* SDK
Load AWS Credentials
Access Kinesis* sub-API

Create subscription for status stream

Send data to Kinesis*
Streaming Data to the Cloud with AWS* Kinesis*

var AWS = require('aws-sdk');
AWS.config.loadFromPath(
  "/home/root/aws_config.json");
var kinesis = new AWS.Kinesis();
hub.subscribe(function(err,status) {
},"kinesis"); // publish ID

Send data to
Kinesis*

var log_data = {
  'Data': JSON.stringify(status),
  'StreamName': 'Light',
  'PartitionKey': ...
};
kinesis.putRecord(log_data,
  function(err,data) {
    if (err) {
      ...
    }
  }
}
Agenda

- Internet of Things
- Set up the Thing
- Build the Thing
- Program the Thing
- Network the Thing
- Connect the Thing
- Conclusion (5m)
Summary

- The Internet of Things combines internet and embedded programming
- Embedded things also need to interface with mobile devices and services
- Mobile devices can act as user interfaces to things
- One language can be used to program all of these: JavaScript*
- JavaScript is used in both Node.js* (services, things) and HTML5 (devices)
- A variety of libraries are available for hardware interfacing
- A variety of mechanisms and libraries are available for communication
- Things can talk to web services and even provide their own services
- The Intel® XDK IoT Edition provides a convenient way to program and debug both mobile devices using HTML5 and things using Node.js
Call to Action

What have you done?

• You have seen how to build a complete IoT system
  - Not just a single thing, but a thing talking to both interface devices and services
  - You seen how all this can be done using a single programming language, JavaScript*

• Javascript is not a “toy” language

• ...It is a power tool that can scale to thousands of devices and servers

What can you do?

Pretty much anything....

• Home automation and security systems
• Smart devices to manage energy or monitor the environment
• Swarms of robot minions helpers to do your bidding...
Additional Sources of Information


Additional Seeed* Grove sensors:
http://www.seeedstudio.com/wiki/GROVE_System

Node.js* home page: http://nodejs.org/

Other Node.js packages (over a hundred thirty thousand to choose from...):
https://www.npmjs.org/


Other references used in the preparation of these notes: next page
References


Blogs and articles:

- [http://nodestreams.com/](http://nodestreams.com/)
- [http://substack.net/roll_your_own_pubsub_with_dnode](http://substack.net/roll_your_own_pubsub_with_dnode)
Legal Disclaimer & Optimization Notice

- INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

- Copyright © 2014, Intel Corporation. All rights reserved. Intel, Pentium, Xeon, Xeon Phi, Core, VTune, Cilk, and the Intel logo are trademarks of Intel Corporation in the U.S. and other countries.

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804
Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Cost reduction scenarios described are intended as examples of how a given Intel-based product, in the specified circumstances and configurations, may affect future costs and provide cost savings. Circumstances will vary. Intel does not guarantee any costs or cost reduction.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

Statements in this document that refer to Intel's plans and expectations for the quarter, the year, and the future, are forward-looking statements that involve a number of risks and uncertainties. A detailed discussion of the factors that could affect Intel's results and plans is included in Intel's SEC filings, including the annual report on Form 10-K.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel does not control or audit third-party benchmark data or the web sites referenced in this document. You should visit the referenced web site and confirm whether referenced data are accurate.

Intel, Atom, Core, and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. and any use of such marks by Intel is under license.

*Other names and brands may be claimed as the property of others.

© 2015 Intel Corporation.
Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as "anticipates," "expects," "intends," "plans," "believes," "seeks," "estimates," "may," "will," "should" and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel's actual results, and variances from Intel's current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be important factors that could cause actual results to differ materially from the company's expectations.

Demand for Intel's products is highly variable and could differ from expectations due to factors including changes in the business and economic conditions; consumer confidence or income levels; customer acceptance of Intel's and competitors' products; competitive and pricing pressures, including actions taken by competitors; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Intel's gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; and product manufacturing quality/yields. Variations in gross margin may also be caused by the timing of Intel product introductions and related expenses, including marketing expenses, and Intel's ability to respond quickly to technological developments and to introduce new features into existing products, which may result in restructuring and asset impairment charges. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Results may also be affected by the formal or informal imposition by countries of new or revised export and/or import and doing-business regulations, which could be changed without prior notice. Intel operates in highly competitive industries and its operations have high costs that are either fixed or difficult to reduce in the short term. The amount, timing and execution of Intel's stock repurchase program and dividend program could be affected by changes in Intel's priorities for the use of cash, such as operational spending, capital spending, acquisitions, and as a result of changes to Intel's cash flows and changes in tax laws. Product defects or errata (deviations from published specifications) may adversely impact our expenses, revenues and reputation. Intel's results could be affected by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel's ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. Intel's results may be affected by the timing of closing of acquisitions, divestitures and other significant transactions. A detailed discussion of these and other factors that could affect Intel's results is included in Intel's SEC filings, including the company's most recent reports on Form 10-Q, Form 10-K and earnings release.