IoT Prototyping with Arduino, Particle Photon and IFTTT

thomas.amberg@yaler.net

CC BY-SA tamberg.org

September 2015
Internet of Things (IoT)

Computers with sensors and actuators connected through Internet protocols

Instead of just reading and editing virtual resources, we can now measure and manipulate physical properties

The Internet starts pervading the real world
Topics of this workshop

Getting started
(Setup and programming of IoT hardware)

Measuring and manipulating
(Physical computing: sensors and actuators)

Connecting your device to the Internet
(IoT: monitoring sensors, controlling actuators)

Mash-ups with 3rd party services and devices
(Connecting Web-enabled devices to each other)

How the Internet works under the hood
(Some definitions and details, in case you wonder)

Questions? Just ask / Use Google / Help each other
Choosing your hardware

We use Arduino and Particle Photon hardware

Both speak the same programming language

Arduino is a classic and easier to set up

Know Arduino? Try the Photon!

Note: Check arduino.cc and particle.io to learn more
Getting started

The IDE (Integraded Development Environment) allows you to program your board, i.e. “make it do something new”

You edit a program on your computer, then upload it to your board where it’s stored in the program memory (flash) and executed in RAM

Note: Once it has been programmed, your board can run on its own, without another computer
Getting started with Arduino

To install the Arduino IDE and connect your Arduino board to your computer via USB, see

http://arduino.cc/en/Guide/MacOSX or
http://arduino.cc/en/Guide/Windows or
http://arduino.cc/playground/Learning/Linux

Or install https://codebender.cc/static/plugin and use the https://codebender.cc/ online IDE

**Note:** Codebender is great, but has some limitations
Getting started with Photon

To install the **Particle CLI** and connect your Photon board to your computer via USB, see [https://docs.particle.io/guide/getting-started/connect/photon/](https://docs.particle.io/guide/getting-started/connect/photon/)

Then access the **Particle IDE** online at [https://build.particle.io/](https://build.particle.io/)

Or use the Atom IDE [https://www.particle.io/dev](https://www.particle.io/dev)

**Note:** There is an app for Photon setup, but the command line interface (CLI) is more robust
Hello (serial output)

```cpp
void setup () { // runs once
    Serial.begin(9600); // set baud rate
}

void loop () { // runs again and again
    Serial.println("Hello"); // print Hello
}
```

**Note:** type this source code into your IDE and upload it to the device, then check the next slide.
Serial output with Arduino

Click the *Serial Monitor* icon to see serial output, and make sure the baud rate (e.g. 9600) matches your code.

*Note:* Serial output is great to debug your program.
Serial output with Photon on Mac

Open a **terminal**, connect the Photon to **USB**, and type

$ screen /dev/tty.u

Then hit **TAB** to find the USB device name

$ screen /dev/tty.usbmodem1431

Add the baud rate matching your source code

$ screen /dev/tty.usbmodem1431 **9600**

And hit RETURN to see the output

**Note:** Serial output is great to debug your program
Serial output with Photon on PC


Note: Serial output is great to debug your program.
Examples on Bitbucket

The source code of the following examples is available on Bitbucket, a source code repository.

Download the ZIP from https://bitbucket.org/tamberg/iotworkshop/get/tip.zip

Or browse code online at https://bitbucket.org/tamberg/iotworkshop/src/tip

Note: use the Raw button to see files as plain text.
Measuring and manipulating
Measuring and manipulating

IoT hardware has an interface to the real world

GPIO (General Purpose Input/Output) pins

Measure: read sensor value from input pin
Manipulate: write actuator value to output pin

Inputs and outputs can be digital or analog
The LED

The LED (Light Emitting Diode) is a simple digital actuator.

LEDs have a short leg (-) and a long leg (+) and it matters how they are oriented in a circuit.

To prevent damage, LEDs are used together with a 1KΩ resistor (or anything from 300Ω to 2KΩ).
The resistor

Resistors are the **workhorse of electronics**

Resistance is *measured in Ω (Ohm)*

A resistor’s orientation does not matter

A resistor’s Ω value is **color-coded** right on it

**Note:** color codes are great, but it’s easier to use a multi-meter if you’ve got one, and just measure Ω
The breadboard

A breadboard lets you wire electronic components without any soldering

Its holes are connected “under the hood” as shown here
Wiring a LED with Arduino

The long leg of the LED is connected to pin D7, the short leg to ground (GND).

Note: the additional 1K Ω resistor should be used to prevent damage to the pins / LED if it’s reversed.
Wiring a LED with Photon

The long leg of the LED is connected to **pin D7**, the short leg to ground (**GND**).

**Note**: the additional **1K Ω** resistor should be used to prevent damage to the pins / LED if it’s reversed.
Controlling a LED (digital output)

```cpp
int ledPin = 7;

void setup () {
    pinMode(ledPin, OUTPUT);
}

void loop () {
    digitalWrite(ledPin, HIGH);
    delay(500); // wait 500ms
    digitalWrite(ledPin, LOW);
    delay(500);
}
```

**Note:** blinking a LED is the *Hello World* of embedded software

Set `ledPin` as wired in your LED circuit

**HIGH** = digital 1 (5V) means LED is on,
**LOW** = digital 0 (0V) means LED is off
Controlling a relay (digital output)

Note: the relay shield uses pins D0-D3 for the relays
The switch

A switch is a simple, digital sensor

Switches come in different forms, but all of them in some way open or close a gap in a wire

The pushbutton switch has four legs for easier mounting, but only two of them are needed

Note: you can also easily build your own switches, for inspiration see e.g. http://vimeo.com/2286673
Wiring a switch with Arduino

Note: the resistor in this setup is called pull-down ‘cause it pulls the pin voltage down to GND (0V) if the switch is open.

Pushbutton switch
10K Ω resistor
5V
GND
D2 (max input 5V!)
Wiring a switch with Photon

**Note:** the resistor in this setup is called *pull-down* ‘cause it pulls the pin voltage down to GND (0V) if the switch is open.

Pushbutton switch
10K Ω resistor
VIN = 4.8V out
GND
D2 (max input 5V!)
Reading a switch (digital input)

```cpp
int sensorPin = 2; // e.g. button switch

void setup () {
    Serial.begin(9600); // set baud rate
    pinMode(sensorPin, INPUT);
}

void loop () {
    int sensorValue = digitalRead(sensorPin);
    Serial.println(sensorValue); // print 0 or 1
}
```

Open the IDE serial monitor or terminal to see log output
Switching a LED

```
int switchPin = 2;
int ledPin = 7; // or 13
void setup () {
    pinMode(switchPin, INPUT);
    pinMode(ledPin, OUTPUT);
}
void loop () {
    int switchValue = digitalRead(switchPin);
    if (switchValue == 0) {
        digitalWrite(ledPin, LOW);
    } else { // switchValue == 1
        digitalWrite(ledPin, HIGH);
    }
}
```

Note: figure out the wiring or just use the built-in LED, i.e. pin 13 on Arduino and D7 on Photon

The code inside an if statement is only executed if the condition is true, else is executed otherwise.
The LDR

A photoresistor or LDR (light dependent resistor) is a resistor whose resistance depends on light intensity.

An LDR can be used as a simple, analog sensor.

The orientation of an LDR does not matter.
Wiring an LDR with Arduino

**Photoresistor (LDR)**
- 10K Ω resistor
- 5V
- GND
- A0

**Note:** this setup is a voltage-divider, as the 5V total voltage is divided between LDR and resistor to keep 0V < A0 < 2.5V
Wiring an LDR with Photon

Note: this setup is a voltage-divider, as the total voltage is divided between LDR and resistor to keep $0V < A0 < 2.5V$

Photoresistor (LDR)
10K Ω resistor

$\text{VIN} = 4.8V$ out

GND
A0
Reading an LDR (analog input)

int sensorPin = A0; // LDR or other analog sensor

void setup () {
  Serial.begin(9600); // set baud rate
}

void loop () {
  int sensorValue = analogRead(sensorPin);
  Serial.println(sensorValue); // print value
}

Note: use e.g. Excel to visualize values over time

Open the IDE serial monitor or terminal to see log output
The Servo

A **servo** motor takes an input between 0 and 180 which is translated into a motor position in degrees.

A servo is a **analog actuator**

To create an analog output for the servo, the device uses pulse width modulation (PWM).
Wiring a Servo with Arduino

Note: PWM pins on Arduino are those with a ~ symbol

5V
GND
D3 (PWM)
Wiring a Servo with Photon

**Note:** PWM pins on Photon are D0 - D3, A4 and A5

Vin = 4.8V out
GND
D3 (PWM)
Controlling a Servo (PWM output)

#include <Servo.h>  // remove this line on the Photon
Servo servo;  // create a new Servo object
int servoPin = 3;  // a PWM pin

void setup () {
    servo.attach(servoPin);
}

void loop () {
    for (int pos = 0; pos <= 180; pos += 10) {
        servo.write(pos);
        delay(100);
    }
}

Note: Servo objects let you use Servos without PWM skills
The for loop repeats from pos 0 until pos is 180, in steps of 10
#include <Servo.h>  // remove this line on the Photon
Servo servo;  // create a new Servo
int servoPin = 3;  // a PWM pin
int sensorPin = A0;  // LDR

void setup () {
    servo.attach(servoPin);
}

void loop () {
    int val = analogRead(sensorPin);
    int pos = map(val, 0, 255, 0, 180);
    servo.write(pos);
}
Connecting to the Internet
Web client with Curl

Install Curl from http://curl.haxx.se/ then open a terminal and type, e.g.

```
$ curl -vX GET http://www.oh-a-show.net/
```

The result is the same as opening the page http://www.oh-a-show.net/ in your browser, right-clicking it and selecting View Page Source

Note: browsers, curl or a device can be Web clients
Adding Ethernet to Arduino

**Note:** the Ethernet shield stacks onto the Arduino - just make sure the pins line up properly.

Pins 10, 11, 12 and 13 are used by the shield according to http://playground.arduino.cc/Main/ShieldPinUsage
Adding CC3000 Wi-Fi to Arduino

**Note:** make sure to use a reliable power source, e.g. USB, as Wi-Fi consumes lots of power.

- CC3000 VIN to 5V
- GND to GND
- CLK to D13, MISO to D12, MOSI to D11, CS to D10, VBEN to D5, IRQ to D3
Web client with Arduino (Ethernet)

After adding an Ethernet shield to the Arduino, connect it with the Ethernet cable, then open File > Examples > Ethernet > WebClient

```c
byte mac[] = { ... }; // MAC from sticker on shield
IPAddress ip(...); // set a unique IP or just ignore
```

If it works, change the HTTP request path and host

Note: open the serial monitor window to see output
Web client with Arduino (CC3000)

Install the library [http://learn.adafruit.com/adafruit-cc3000-wifi/cc3000-library-software](http://learn.adafruit.com/adafruit-cc3000-wifi/cc3000-library-software) then open File > Examples > Adafruit_CC3000 > WebClient

```c
#define WLAN_SSID "..." // set local Wi-Fi name
#define WLAN_PASS "..." // set Wi-Fi password
```

If it works, change the WEBSITE and WEBPAGE

**Note:** open the serial monitor window to see output
Web client with Photon

The Particle Photon has **built-in** Wi-Fi. See *Getting Started with Photon*, or press SETUP for 3s and set up **SSID** and **password** of a new local network with

$ particle$ setup wifi

In the Particle IDE, go to *Libraries > Community Libraries > HttpClient* and click *Use This Example*

If it works, change request hostname and path

**Note:** open the serial monitor window to see output
Monitoring connected sensors
ThingSpeak with Curl

The ThingSpeak service lets you store, monitor and share sensor data in open formats. Sign up at https://thingspeak.com/ to create a channel and get API keys, then try the following:

$ curl -vX POST http://api.thingspeak.com/update?key=WRITE_API_KEY&field1=42

$ curl -v http://api.thingspeak.com/channels/CHANNEL_ID/feed.json?key=READ_API_KEY
ThingSpeak with Arduino (Ethernet)


```java
byte mac[] = { ... }; // MAC from sticker on shield
String writeApiKey = "..." // from channel API keys
```

Analog input expected on pin A0, e.g. from an LDR

See [https://thingspeak.com/channels/CHANNEL_ID](https://thingspeak.com/channels/CHANNEL_ID)
Open *File > Examples > Adafruit_CC3000 > WebClient* and set *WLAN_SSID* and *WLAN_PASS* as before, then change web site and page to

```c
#define WEBSITE "api.thingspeak.com"
#define WEBPAGE "/update?
  key=WRITE_API_KEY&field1=42"
```

If it works, replace 42 with analog input, e.g. from an *LDR* using something like `+ String(analogRead(A0))`

See https://thingspeak.com/channels/CHANNEL_ID

**Note:** this example requires a bit of programming
ThingSpeak with Photon

In the Particle IDE, go to Libraries > Community Libraries > ThingSpeak and click Use This Example

Replace YOUR-CHANNEL-KEY with a write API key

If it works, replace int rand = ... with analog input, e.g. from an LDR on pin A0

See https://thingspeak.com/channels/CHANNEL_ID
Controlling connected actuators
The NeoPixel

A multi-color LED with a chip in each pixel that can be controlled with a (PWM-based) library
Wiring a NeoPixel with Arduino

Note: PWM pins on Arduino are those with a ~ symbol

Flat side of the LED is left on this picture

5V
GND
D6 (PWM)
Wiring a NeoPixel with Photon

Note: PWM pins on Photon are D0 - D3, A4 and A5

Flat side of the LED is left on this picture

VIN = 4.8V out
GND
D2 (PWM)
Testing a NeoPixel with Arduino

Install the library [https://github.com/adafruit/Adafruit_NeoPixel](https://github.com/adafruit/Adafruit_NeoPixel) then open File > Examples > Adafruit_Neopixel > strandtest

```
Adafruit_NeoPixel strip = Adafruit_NeoPixel(1, PIN, NEO_GRB + NEO_KHZ400);

If it works, replace loop content with strip.setPixelColor(0, strip.Color(0, 255, 0)); strip.show();
```

**Note:** the rapid blinking is intended for LED strands
Testing a NeoPixel with Photon

In the Particle IDE, go to Libraries > Community Libraries > NeoPixel and click Use This Example

#define PIXEL_COUNT 1 // 10
#define PIXEL_TYPE WS2811 // WS2812B

If it works, remove the for loops and try
strip.setPixelColor(0, strip.Color(0, 255, 0));

Note: the pixel is set green, red, blue (GRB), not RGB
Connected LED with Arduino (Ethernet)

Sign up at [https://yaler.net/](https://yaler.net/) to get a relay domain


$ curl -vX PUT http://RELAY_DOMAIN.try.yaler.net/led/color/ee6600

**Note:** replace RELAY_DOMAIN with your relay domain
Connected LED with Arduino (CC3000)

Sign up at [https://yaler.net/](https://yaler.net/) to get a relay domain

[https://bitbucket.org/tamberg/iotworkshop/src/tip/Arduino/NeoPixelWebServiceCc3k/NeoPixelWebServiceCc3k.ino](https://bitbucket.org/tamberg/iotworkshop/src/tip/Arduino/NeoPixelWebServiceCc3k/NeoPixelWebServiceCc3k.ino)

$ curl -vX PUT http://RELAY_DOMAIN.try.yaler.net/led/color/ee6600

**Note**: replace RELAY_DOMAIN with your relay domain
Connected LED with Photon

https://bitbucket.org/tamberg/iotworkshop/src/tip/ParticlePhoton/NeoPixelWebService/NeoPixelWebService.ino (include NeoPixel library)

$ curl -vX POST https://api.particle.io/v1/devices/DEVICE_ID/led -d access_token=ACCESS_TOKEN -d args=330033

Note: the Particle Cloud API simplifies Web services
Mash-ups
IFTTT

If This Then That (IFTTT) is a mash-up platform

An IFTTT Recipe connects two Web services (or a service and a device) using their Web APIs

The IFTTT Maker Channel uses Webhooks (outgoing HTTP requests) to call your device, and you can use Web requests to trigger IFTTT, the Particle Channel (for Photon) explains itself
IFTTT Do Button with Arduino

Connect the Maker Channel at https://ifttt.com/maker

Get the Do Button App, tap '+' > Channels > Maker > Create a new recipe > Make a Web request > ... then go to https://ifttt.com/myrecipes/do for convenience

URL: http://RELAY_DOMAIN.try.yaler.net/led?color=330033
Method: POST
Content Type: application/x-www-form-urlencoded
IFTTT Do Button with Photon

Connect the Particle channel at https://ifttt.com/particle

Get the Do Button App, tap '+' > Channels > Particle > Create a New Recipe > Call a function and select, e.g. led on DEVICE_NAME

Set the with input field to a color value, e.g. 330033
IFTTT Recipes

Once a recipe works, you can publish it (hiding unneeded fields) for everybody to clone, e.g.

https://ifttt.com/recipes/320868-light-up-arduino-led-at-sunset

https://ifttt.com/recipes/320870-light-up-photon-led-at-sunset

Note: "Do" recipes cannot be published for now
Light up LED at sunset
How the Internet works in detail

If you wonder what TCP/IP, HTTP or DNS means - or care about the difference between protocol, data format and API, read on...
Protocols

Parties need to agree on **how to exchange** data (communicating = exchanging data according to a protocol)

- e.g. **Ethernet** links local computers physically,
- **TCP/IP** is the foundation of the **Internet**, and
- **HTTP** is the protocol that enables the **Web**

**Note**: protocols are layered, e.g. HTTP messages transported in TCP/IP packets sent over Ethernet
TCP/IP

IP (Internet Protocol) deals with host addressing (each host has an IP address) and packet routing.

TCP (Transmission Control Protocol): connection oriented, reliable data stream (packets in-order, errors corrected, duplicates removed, discarded or lost packets resent) from client to server.

Note: DHCP assigns an IP address to your device which is mapped to the device’s MAC address.
HTTP

HTTP (Hypertext Transfer Protocol) enables the distributed, collaborative system we call the Web.

The client sends an HTTP request, the server replies with a response.

HTTP Message = Request | Response

Request = (GET | POST | ...) Path CRLF *(Header CRLF) CRLF Body

Response = "HTTP/1.1" (200|404|...) CRLF *(Header CRLF) CRLF Body

CRLF = "\r\n"

(Read the spec: http://tools.ietf.org/html/rfc2616)

Note: HTTP is human readable, i.e. it’s easy to debug.
URIs

The **URI** (Uniform Resource Identifier) is a string of characters used to identify a resource

```
```


**QR codes, NFC tags** can contain a machine readable URI

**IoT**: URIs can refer to things or their physical properties

**Note**: good URIs can be hand-written on a napkin and re-typed elsewhere, without any ambiguity
DNS

DNS (Domain Name System) maps Internet domain names to one or more IP addresses.

Try it in your desktop computer terminal, e.g.

```
$ nslookup google.com
173.194.35.6 ...
```

**Note:** if your device doesn’t support DNS you can connect to the server’s IP, but beware of changes.
Data formats

Parties need to agree on **what is valid** content (parsing = reading individual content tokens)

CSV: easy to parse, suited for tables, old school

**JSON**: easy to parse, de facto standard

XML: used by many services, W3C standard

Semi-structured text, e.g. Twitter’s @user, #tag

**Binary** formats, e.g. PNG, MP3, ...
RSS

In addition to generic data formats like CSV, JSON, XML there are refinements that add semantics to the document.

RSS (or Atom) is a data format for lists of items.

Invented for blogs, RSS is great for data feeds.

Note: RSS documents are also XML documents, but not all XML documents contain valid RSS.
HTML

**HTML (Hypertext Markup Language)** is a data format describing how a Web page should be structured and displayed.

Look at the HTML (and Javascript) code of any Web page with "view source" in your browser.

**Note:** HTML documents are not always valid XML documents, but Web browsers are very forgiving.
APIs

An API (Application Programming Interface), is an agreement between clients and providers of a service on how to access a service, how to get data out of it or put data into it.

The UI (User Interface) of a service is made for humans, the API is made for other computers.

Note: good APIs are documented or self-explanatory.
REST

REST (Representational State Transfer) is a style of designing an API so that it is easy to use.

REST APIs use **HTTP methods** (GET, PUT, POST, DELETE) to let you perform actions on **resources**.

REST APIs can be explored by following links.

**Note:** good Web UIs are often built following the same principles, therefore REST APIs feel natural.
Learning more

Electronics: Ohm’s law, Kirchhoff’s current and voltage law (KCL & KVL), Make: Electronics by Charles Platt

Interaction Design: Smart Things by Mike Kuniavsky, Designing Connected Products by Claire Rowland et al.

Physical Computing: Making Things Talk by Tom Igoe

REST: RESTful Web Services by Leonard Richardson

Programming: read other people’s code, e.g. on GitHub

IoT: Designing the Internet of Things by Adrian McEwen and Hakim Cassimally, Postscapes.com, IoTList.co

Note: MechArtLab Zürich has an OpenLab on Tuesdays
Reducing E-waste

Tired of hacking?

Donate your hardware...

e.g. MechArtLab
Hohlstrasse 52
8004 Zürich
Thank you

thomas.amberg@yaler.net
twitter.com/tamberg	

tamberg.org

Slides online at http://goo.gl/n3hCbk