

# Live Programming IoT devices

## Pharo Things

Do Hoang

Vietnam France University

June 13, 2019

- 1 Objective
- 2 Installation
- 3 Get Started
- 4 Day 1
- 5 Lesson 2 - Blinking LED
- 6 Lesson 3 - Introduce to Pharo object-oriented
- 7 Lesson 4 - LED Flowing Lights

## Section 1

## Objective

# Objective

- Run Pharo IoT in a Raspberry Pi that has Raspbian already installed.
- Install Pharo IoT and Raspbian from scratch in headless mode (without keyboard/mouse/screen);
- Run and use Pharo IoT IDE on your Linux, Windows or Mac OSX computer.

## Section 2

### Installation

# Install in a Raspberry that has Raspbian

- If you already have Raspbian running on your Raspberry Pi, you can simply use the Pharo IoT zero-conf.
- Open a terminal window in your Rpi ( local or remote SSH )

## Install in a Raspberry that has Raspbian

- Enter:  
  

```
wget -O - get.pharo/server | bash
```
- => You will download the server side and extract the files to the pharoiot-server folder

# Install in a Raspberry that has Raspbian

- Goto the folder and run pharo server

```
cd pharoiot-server
./pharo-server
```

- If everything is alright, you will see this message: ‘a TlpRemoteUIManager is registered on port 40423’.
  - This means that you have TelePharo running on your Raspberry on TCP port 40423.
- Now you can use Pharo IoT on your computer to connect to your Raspberry and create IoT applications remotely.



# Install Raspbian and Pharo IoT from scratch (Raspberry Pi Headless Setup)

- There are many options to install Raspbian on your Raspberry. The most common way is to download the ISO image from the official Raspberry website and follow the steps to install it.
  - Basically, they are: copy an ISO image to an SD card, insert it in Raspberry Pi, turn On the Rasp and use a keyboard/mouse/screen to use it as a normal computer.
- But we will not use keyboard/mouse/screen to install Raspbian and run Pharo IoT! We do not need them.

# Install Raspbian and Pharo IoT from scratch (Raspberry Pi Headless Setup)

- We will use a third-party program to perform these tasks automatically:
  - Install Raspbian Full OS
  - Setting the Raspberry Pi hostname
  - Set boot to console
  - Enable the I2C and SPI modules
  - Connect it in your WiFi network
  - Download Pharo IoT (requires Raspberry Pi connected on the internet)
  - and start the Pharo IoT server at every boot

# Pibakery

- Download the Pibakery.
- Download the configuration file.

`http://get.pharoiot.org/pibakeryPharoIoT.xml`

- Write to your Rpi SD card.

# Pibakery

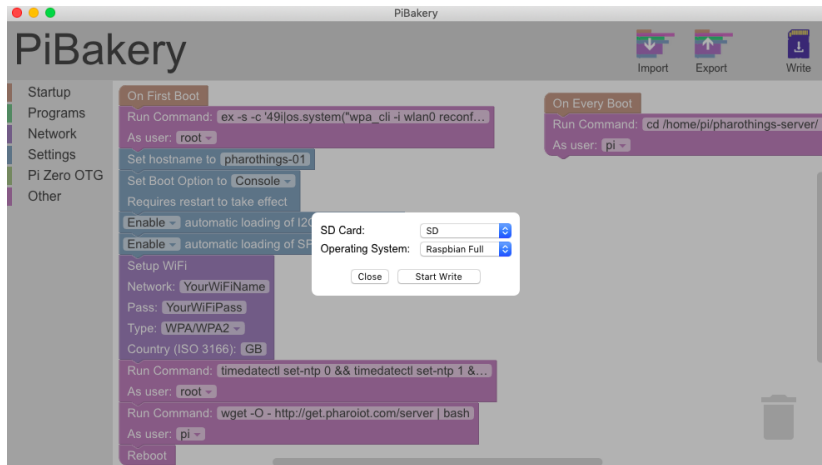


Figure 1: Configuration

# Pibakery

- Change your hostname and WiFi configuration in PiBakery;
- Insert the SD card into your machine, click Write and select the Operation System Raspbian Full;
- After the process, insert the SD in the Raspberry and wait about 3 minutes to complete the automatic configuration. Time depends on the speed of your internet;
- You can now find your Raspberry by the Hostname you defined above.
- You do not need to do anything else on your Raspberry Pi. It is already loaded with Pharo IoT starting at every boot on the TCP port 40423

# Run Pharo IoT IDE on your Linux, Windows or Mac OSX computer

**Download** Pharo from

`get.pharoiot.com/multi.zip`

# Run Pharo IoT IDE on your Linux, Windows or Mac OSX computer

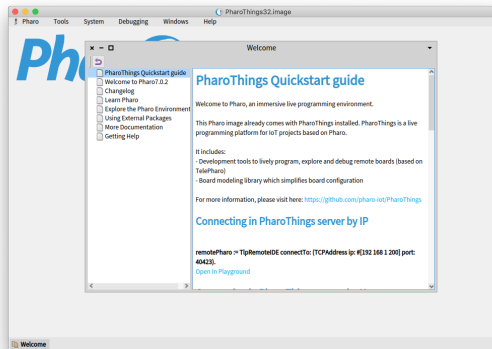


Figure 2: Pharo IDE

## Section 3

### Get Started



# Connecting from your computer to Raspberry Pi

## Connecting in Pharo IoT server by Hostname

```
ip := NetNameResolver addressForName: 'pharoiot-01'.
```

```
remotePharo := TlpRemoteIDE connectTo:  
    (TCPAddress ip: ip port:40423).
```

## Connecting in Pharo IoT server by IP

```
remotePharo := TlpRemoteIDE connectTo:  
    (TCPAddress ip: #[192 168 1 200] port: 40423).
```

## Working remotely

- If you don't receive any error, this means that you are connected.
- Now you can call the Remote Playground, Remote System Browser, and Remote Process Browser.

### Remote control

```
remotePharo openPlayground.  
remotePharo openBrowser.  
remotePharo openProcessBrowser.
```

## Inspect the remote Raspberry Pi GPIO board

- You can inspect the physical board of your Raspberry Pi.
- For Raspberry, it will be one of the RpiBoard subclasses. Currently, you can use the following classes according to the models:
  - RpiBoardBRev1: Raspberry Pi Model B Revision 1
  - RpiBoardBRev2: Raspberry Pi Model B Revision 2
  - RpiBoard3B: Raspberry Pi Model B+, Pi2 Model B, Pi3 Model B, Pi3 Model B+

With the chosen class evaluate the following code to open an inspector:

```
remoteBoard := remotePharo evaluate: [
    RpiBoard3B current].
remoteBoard inspect.
```

# Remote GPIO inspector

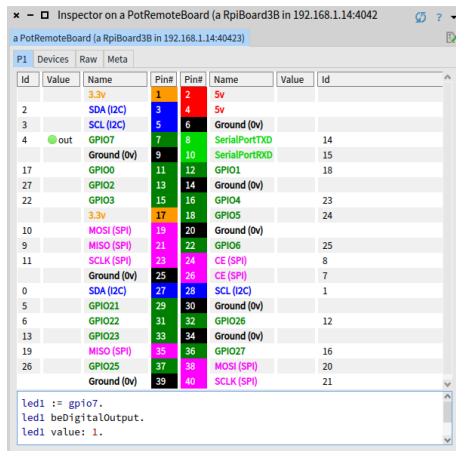


Figure 3: Remote GPIO inspector

# GPIOs

- A live tool which represents the current pins state.
- The evaluation pane in the bottom of the inspector provides bindings to gpio pins which you can script by `#dolt/printIt` commands
- Digital pins are shown with green/red icons which represent high/low (1/0) values.
- Able to toggle the value.

## Saving the remote image

```
remotePharo saveImage.
```

## Disconnect all remote sessions

```
TlpRemoteIDE disconnectAll.
```

## Section 4

### Day 1

# Lesson 1 – Turning LED on/off

## Components

- 1 Raspberry Pi connected to your network (wired or wireless)
- 1 Breadboard
- 1 LED
- 1 Resistor (330 ohms)
- Jumper wires

# Experimental procedure

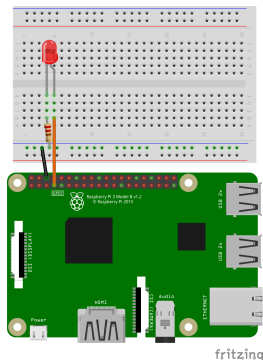


Figure 4: Physical connection LED

\* The circuit consists of an LED that lights up when power is applied, a resistor to limit current and a power supply (the Rasp).



## Physical connection LED detail

- Connect the Ground PIN from Raspberry in the breadboard blue rail (-). Raspeberry Pi models with 40 pins has 8 GPIO ground pins. You can connect with anyone. In this experiment we will use the PIN6 (Ground);
- Then connect the resistor from the blue rail on the breadboard (-) to a column on the breadboard
- Now push the LED legs into the breadboard, with the long leg (with the kink) on the right;
- And insert a jumper wire connecting the rigth column and the PIN7 (GPIO7).

# Experimental code

## Connecting remotely

- Run this code in Playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 167] port: 40423)
```

```
GTInspector enableStepRefresh
```

```
remoteBoard := remotePharo evaluate: [ RpiBoard3B
    current].
```

```
remoteBoard inspect.
```

⇒ Make a new connection to your Rpi and Open the *Remote Playground*

# Experimental code

- To control the LED we first introduce the named variable `#led` which we assigned to GPIO7 pin instance:

```
led := gpio7.
```

- Then we configure the pin to be in digital output mode and set the value:

```
led beDigitalOutput.
```

```
led value: 1.
```

⇒ It turns the LED on.

# Experimental code

- You can **notice** that gpio variables are not just numbers/ids.
- They are real **objects** with behaviour.
- For example you can ask pin to toggle a value:

```
led toggleDigitalValue.
```

- Or ask a pin for current value if you want to check it:

```
led value.
```

# Result

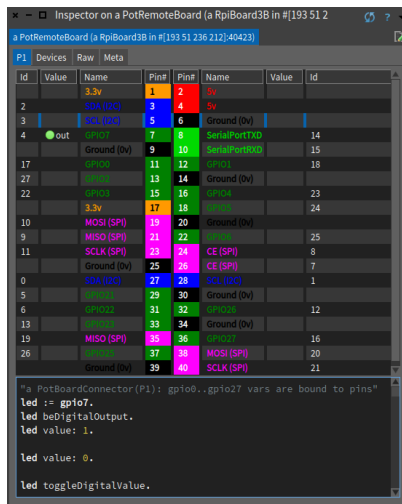


Figure 5: Remote Board Inspector

## Section 5

### Lesson 2 - Blinking LED

# Experimental code

## Connecting remotely

- Run this code in Playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 212] port: 40423)
```

```
GTInspector enableStepRefresh.
```

```
remoteBoard := remotePharo evaluate:
    [ RpiBoard3B current].
```

```
remoteBoard inspect.
```

⇒ Make a new connection to your Rpi and Open the *Remote Playground*

## Experimental code

- We still assigned the LED pin to GPIO7 pin:

```
led := gpio7.  
led beDigitalOutput.
```

- To blink the LED, we create a loop to change the value of LED by time.
- We use the method *toggleDigitalValue* as previously.
- For example we blink the LED every 1 second by 10 times.

```
[ 10 timesRepeat: [  
  led toggleDigitalValue.  
  (Delay forSeconds: 1) wait  
] ] forkNamed: 'BlinkerProcess'.
```

⇒ Your LED is blinking now!



## Section 6

### Lesson 3 - Introduce to Pharo object-oriented

## Blinking LED using OOP

### Experimental code

```
|blinker|  
blinker := Blinker new.  
blinker timesRepeat: 10 waitForSeconds: 1.
```

# Experimental code

- We declare the variable *blinker* in the first line.
- We will use this variable to create an object using the Blinker class.
- In the second line, we instantiate the Blinker class in the *blinker* variable
- In the third line, we send some messages to the blinker object to controll the times.

⇒ This will make the GPIO behave according to the parameters

# Create your own class remotely

- To create a class, we need first to create a package.
- In your local playground, call the Remote System Browser of your Raspberry Pi

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 212] port: 40423).
remotePharo openBrowser.
```

## Create a package

- Using the Remote Browser to create.
- *Right-click* the package area and enter the package name.
- For example, we create a package named *PharoThings-Lessons*

## Create a class

- Edit the default class template by changing the *#NameOfSublass* to the name of new class
- For example, let's create the class *#Blinker*
- The class name begin with hash symbol (#) and a calpital leter.

```
Object subclass: #Blinker
instanceVariableNames: 'led'
classVariableNames: ''
package: 'PharoThings-Lessons'
```

- Right click on the code and select *Accept* option.  
 ⇒ The class is compiled and added to the system.

# Create a protocol

- Create a new protocol to organize the methods.
- The first protocol: `initialization`

# Create a protocol

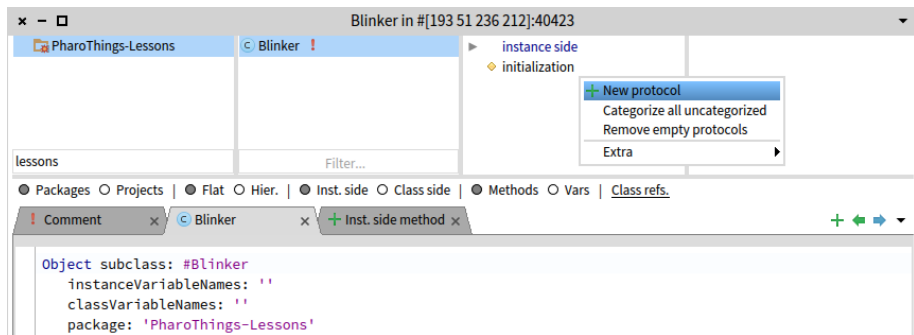


Figure 6: Creating a package remotely.



# Create a method

- Inside this protocol, create an `initialize` method.  
⇒ Everytime a new object was created using `Blinker` class, this method will be executed to define some variable in the new object.
- Let's use the instance variable `led`.
- The instance variable is private to the object and accessible by any methods inside this class.
- These methods can access this variable to get or set any value to it.

## initialize

```
initialize
```

```
led := PotClockGPIOPin id: 4 number: 7.
led board: RpiBoard3B current; beDigitalOutput
```

## Creating the initialize method

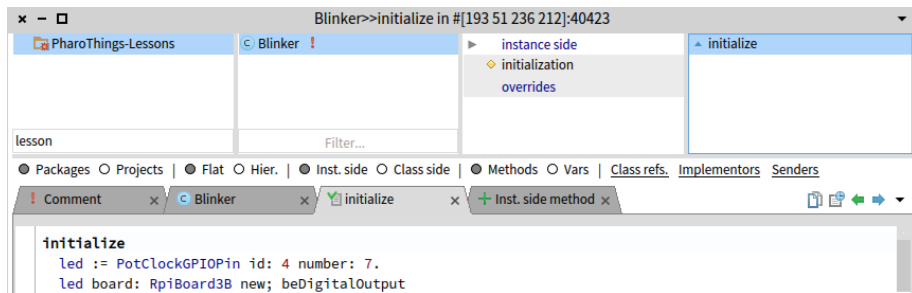


Figure 7: Creating the initialize method

# Explanation

## Explanation

- The first line defines the name of the method;
- In the second line, we configure the GPIO that we wanna use. Note that we need the GPIO number and ID. The ID is required to communicate with WiringPi Library. You can see the ID and GPIO number in PotRemoteBoard inspector.
- In the third line, we define the model of the Raspberry board and configure this GPIO as beDigitalOutput. This means that when the GPIO change to value:1, the power will go out of the GPIO to power the LED.
- Compile your code (cmd + S) and the method will be shown in the remote browser.

## Create a method to do actions

- Let's create a method to control the object led inside the class Blinker.
- Create a protocol operations and inside this protocol, create the following method

### Method

```
timesRepeat: anInteger waitSeconds: aNumber
    [ anInteger timesRepeat: [
        led toggleDigitalValue.
        (Delay forSeconds: aNumber) wait
    ] ] forkNamed: 'BlinkerProcess'.
```

# Explanation

## Explanation

- In the first line, we define the message with `timesRepeat:` and `waitForSeconds:`. We inform the kind of value will be received, creating 2 variables: `aNumber` and `anInteger`;
- We replace these variables in the code and now we have the control to say how many times repeat and for how many seconds;
- We finished the code by putting everything inside a fork to create a process in Pharo. While the process is running, you can open the Remote Process Browser (`remotePharo openProcessBrowser`) and see the process. This is useful when you wanna kill the remote process.

## Using the new class

- Now we can use the class that we created, the Blinker class. To do this, let's open the Remote Playground:

```
remotePharo openPlayground
```

### Call blinker class

```
|blinker|
  blinker := Blinker new.
  blinker timesRepeat: 10 waitForSeconds: 1.
```

## Save your work

Don't forget to save your work remotely. To do this, run this command on your local playground:

```
remotePharo saveImage.
```

## Section 7

### Lesson 4 - LED Flowing Lights



# What do we need

## Components

- 1 Raspberry Pi connected to your network (wired or wireless)
- 1 Breadboard
- 8 LEDs
- 8 Resistors 330ohms
- Jumper wires

## Schema connection 8 LEDs

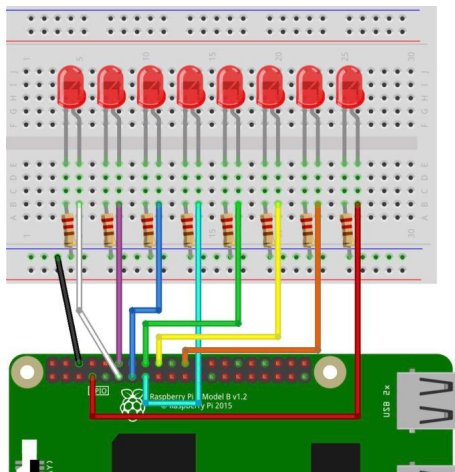


Figure 8: Schema connection 8 LEDs

## Physical connection 8 LEDs

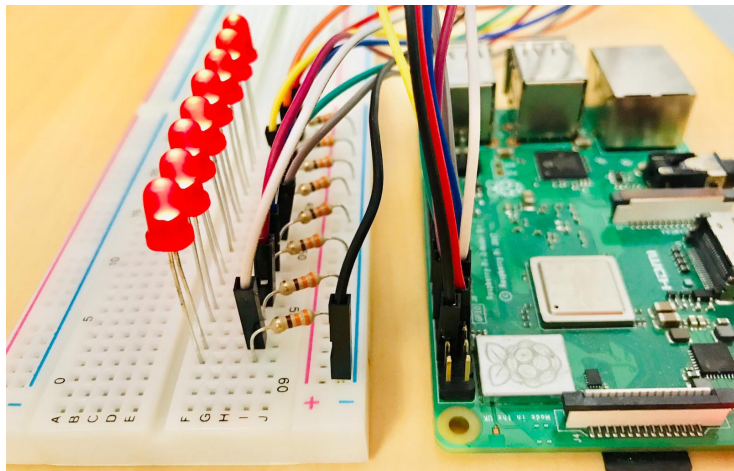


Figure 9: Physical connection 8 LEDs

## Experimental process

- Connect the Ground PIN from Raspberry in the breadboard blue rail (-).
- Then connect the 8 resistors from the blue rail (-) to a column on the breadboard.
- Now push the LED legs into the breadboard, with the long leg (with the kink) on the right.
- And insert the jumper wires connecting the right column of each LED to GPIO from 0 to 7.

## Connecting remotely

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip:
    #[193 51 236 212] port: 40423)
GTInspector enableStepRefresh.
remoteBoard := remotePharo evaluate: [ RpiBoard3B current].
remoteBoard inspect.
```

# Experimental code

- Let's create an array and initialize the 8 LEDs, putting each one in a position of the array.

```
gpioArray := { gpio0. gpio1. gpio2. gpio3.
               gpio4. gpio5. gpio6. gpio7 }.
gpioArray do: [ :item | item beDigitalOutput ].
```

## Experimental code

- In the previous lesson, we use `toggleDigitalValue` to change the value of the object (Led value)

```
gpioArray do: [ :item | item toggleDigitalValue ].
```

## Experimental code

- Let's put a Delay after changing the led value, to wait a bit time before to change the next LED value. Let's also put this inside a process using the method forkNamed:

```
[
  gpioArray do: [ :item | item toggleDigitalValue.
    (Delay forSeconds: 0.3) wait ].
] forkNamed: 'FlowingProcess'.
```



# Result

- Execute this code and... cool! Now your LEDs are on by flowing an ordering!

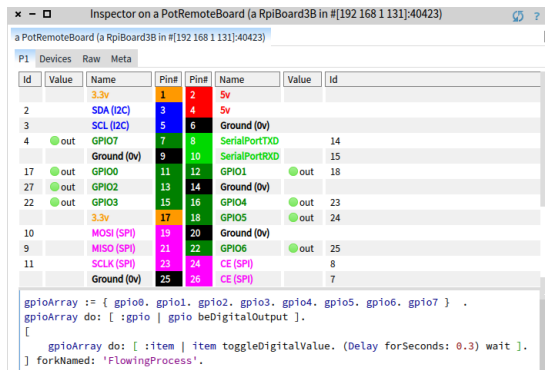


Figure 10: Code on Inspector

## Result

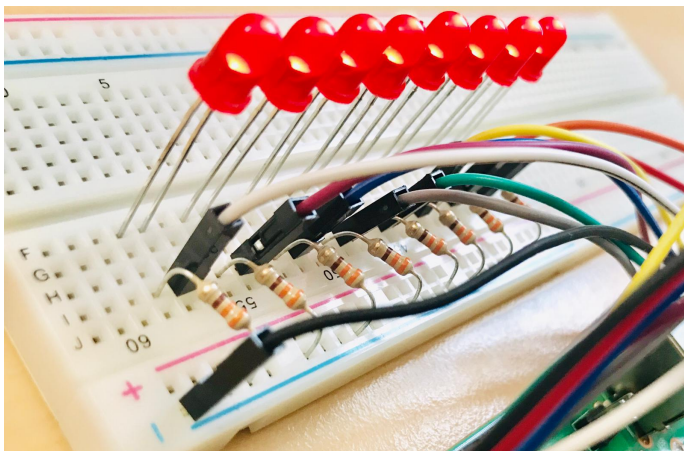


Figure 11: LEDs turn On.

## Adding features

### Control loop time?

```
[ 2 timesRepeat: [  
    gpioArray do: [ :item | item toggleDigitalValue.  
    (Delay forSeconds: 0.1) wait ].  
] ] forkNamed: 'FlowingProcess'.
```

# Adding features

## Reversing the flow

```
[ 2 timesRepeat: [  
  gpioArray reverseDo: [ :item | item toggleDigitalValue.  
    (Delay forSeconds: 0.1) wait ].  
] ] forkNamed: 'FlowingProcess'.
```

# Terminating the process

- Call the **Remote Process Browser**

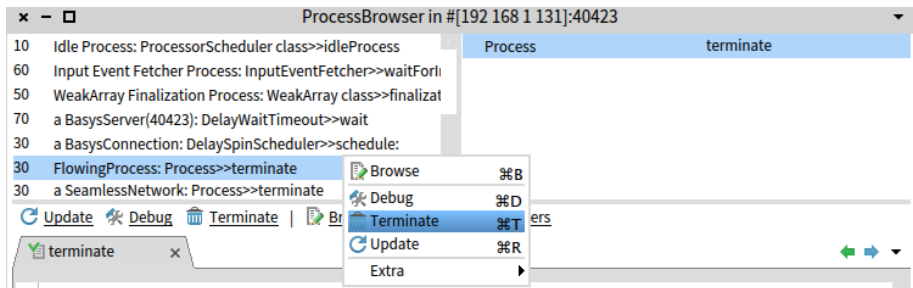


Figure 12: Remote Process Browser