A PharoThings Tutorial

Allex Oliveira

November 9, 2018

Copyright 2017 by Allex Oliveira.

The contents of this book are protected under the Creative Commons Attribution-ShareAlike 3.0 Unported license.

You are **free**:

- to **Share**: to copy, distribute and transmit the work,
- to **Remix**: to adapt the work,

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

For any reuse or distribution, you must make clear to others the license terms of this work. The best way to do this is with a link to this web page:
http://creativecommons.org/licenses/by-sa/3.0/

Any of the above conditions can be waived if you get permission from the copyright holder. Nothing in this license impairs or restricts the author's moral rights.

Your fair dealing and other rights are in no way affected by the above. This is a human-readable summary of the Legal Code (the full license):
http://creativecommons.org/licenses/by-sa/3.0/legalcode

Layout and typography based on the `sbabook` LaTeX class by Damien Pollet.

Contents

Illustrations iii

1 Installations 3
1.1 Installating OS on RASPBERRY (RASPBIAN) . . . 3
1.2 Copying Raspbian files on MAC OSX . . . 4
1.3 Copying Raspbian files on Linux . . . 4
1.4 Copying Raspbian files on Windows . . . 4
1.5 Installing the Raspbian in Raspberry Pi . . . 4
1.6 Installing PharoThings on Raspberry Pi . . . 5
1.7 Execute PharoThings on Raspberry . . . 8
1.8 Connecting Pharo client on remote Pharo . . . 9
1.9 In the next chapter . . . 10

2 Lesson 1 – Turning LED on/off 11
2.1 What we need? . . . 11
2.2 Experimental theory . . . 12
2.3 Experimental procedure . . . 13
2.4 Experimental code . . . 14
2.5 What did we learn? . . . 16
2.6 In the next lesson . . . 16

3 Lesson 2 – Blinking LED 17
3.1 What do we need? . . . 17
3.2 Experimental code . . . 18
3.3 In the next lesson . . . 18

4 A brief introduction to Pharo object-oriented language 19
4.1 Developing a simple LED blinker . . . 19
4.2 Our use case . . . 19
4.3 Create your own class remotely . . . 20
4.4 Create a package . . . 20
4.5 Create a class . . . 21
4.6 Create a protocol . . . 21
4.7 Creating an initialize method . . . 21
4.8 Using your new class . . . 24
4.9 Save your work . . . 25
4.10 Conclusion . . . 25

5 Lesson 3 - LED Flowing Lights **27**
5.1 What we need? 27
5.2 Experimental procedure 27
5.3 Experimental code 29

Illustrations

1-1 Preparing SD Card. 4
1-2 Copying NOOBS. 4
1-3 Installing Raspbian. 5
1-4 Installing PharoLauncher. 5
1-5 Download Pharo 61. 6
1-6 Open your Pharo image. 6
1-7 Loading PharoThings. 7
1-8 Copying PharoARM. 7
1-9 Copying the folder on your Raspberry. 8
1-10 Server up and running. 9
1-11 Remote GPIO inspector 10

2-1 Led polarity and resistors. 13
2-2 Breadboard scheme. 13
2-3 Physical connection LED. 14
2-4 Remote Board Inspector. 15

3-1 Remote playground. 18

4-1 Remote System Browser. 20
4-2 Creating a package remotely. 21
4-3 Creating a class remotely. 22
4-4 Creating a package remotely. 22
4-5 Looking for ID and GPIO number on Remote Inspector. 23
4-6 Creating the initialize method. 24
4-7 Creating an operation method. 24
4-8 Remote playground. 25

5-1 Physical connection 8 LEDs. 28

In this booklet we will show you how to develop a little application that collect weather information. We will start to show how we can play with leds and others.

CHAPTER 1

Installations

The first step you need to do to get started with PharoThings is to install an Operating System in your Raspberry Pi. When you buy a Raspberry Pi, the OS is not factory installed.

1.1 Installating OS on RASPBERRY (RASPBIAN)

In this chapter, we will download and install NOOBS (New Out Of the Box Software). NOOBS is an easy operating system installer which contains Raspbian (https://www.raspberrypi.org/downloads/raspbian/) and LibreELEC (https://libreelec.tv).

Raspbian is the Foundation's official supported operating system, a Linux OS based on Debian Stretch to run in ARM processors.

Download

You can download an official image from the Raspberry Pi website Noobs downloads page (https://www.raspberrypi.org/downloads/noobs/). You will download a zip file and extract the files to your SD card.

Copying

You will need a computer with an SD card reader to install the image. This process basically extracts the files from the zip file downloaded into an SD card formatted and start the Raspberry Pi with this SD card.

You can go directly to your operating system by clicking on the links below:

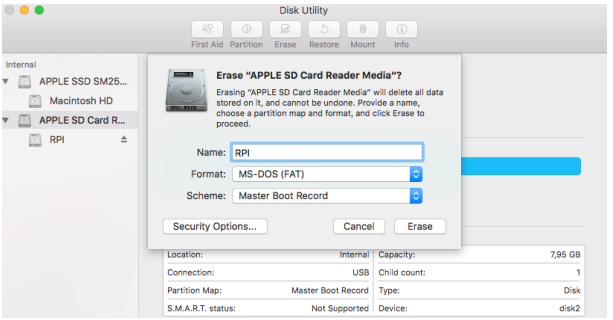

Figure 1-1 Preparing SD Card.

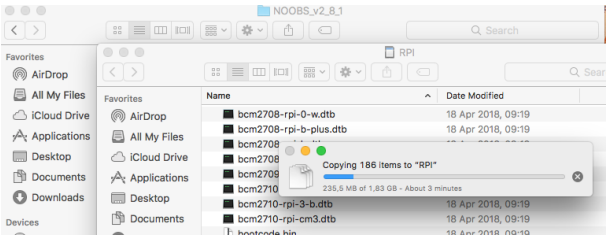

Figure 1-2 Copying NOOBS.

1.2 Copying Raspbian files on MAC OSX

- Open “disk utility”, select the SD Card and Erase (Format MS-DOS FAT) as shown in Figure 1-1.
- Copy the files from folder NOOBS_xxx to SD Card as shown in Figure 1-2.

1.3 Copying Raspbian files on Linux

1.4 Copying Raspbian files on Windows

1.5 Installing the Raspbian in Raspberry Pi

Insert the SD Card on Raspberry and turn it on. Select “Raspbian” and “Yes” as shown by Figure 1-3.

In a few minutes, you will have your Raspberry Pi running Raspbian OS. Now you can install PharoThings and control devices remotely.

Figure 1-3 Installing Raspbian.

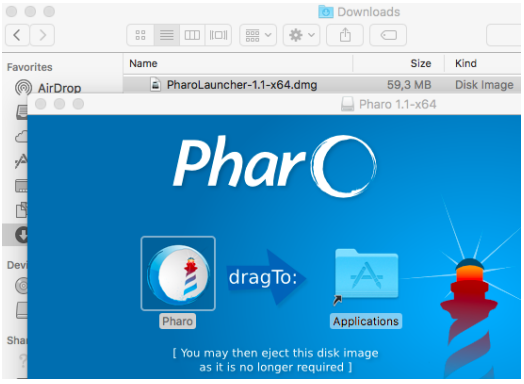

Figure 1-4 Installing PharoLauncher.

1.6 Installing PharoThings on Raspberry Pi

Install PharoThings requires to get Pharo, PharoThings and an ARM virtual machine. We will build the image on our local computer with the correct files and after we will copy to a Raspberry Pi and execute it there. In the third step, we will connect the Pharo from the local computer into Pharo of Raspberry Pi.

Download PharoLauncher

Use the PharoLauncher (an application to help you running multiple versions and images of Pharo) and install Pharo 6.1. You can get the launcher from http://pharo.org/download. You can also directly install a version of Pharo from the same place.

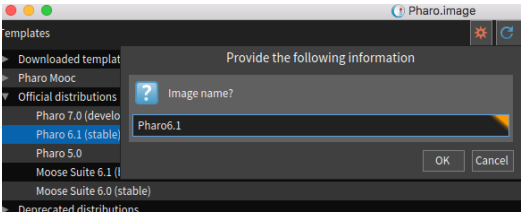

Figure 1-5 Download Pharo 61.

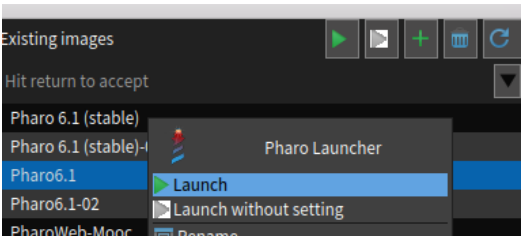

Figure 1-6 Open your Pharo image.

Download Pharo 61

Run the Pharo Launcher. Double click the distribution you want to create a image and give a name to image (see Figure 1-5). A short name and without spaces is recommended, because we will type this name and path in command line on Linux.

Execute your Pharo image

Launch the image as shown in Figure 1-6. A folder with the image name will be created inside the folder Pharo: `/Users/your_user_name/Documents/Pharo/`

In this example the folder is `/Users/my_user_name/Documents/Pharo/Pharo6.1`

Load PharoThings

Open Playground and execute this command to install the server part of PharoThings (as shown in Figure 1-7):

```
Metacello new
  baseline: 'PharoThings';
  repository: 'github://pharo-iot/PharoThings/src';
  load: #(RemoteDevServer Raspberry).
```

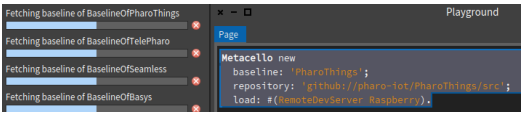

Figure 1-7 Loading PharoThings.

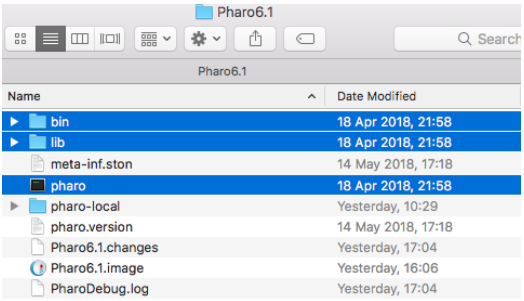

Figure 1-8 Copying PharoARM.

Then configure image to disable slow browser plugins (instead remote browser will be much slower):

```
ClySystemEnvironmentPlugin disableSlowPlugins.
```

Snapshot your Image

In Pharo, click and "Save and Quit". This way all your code and configurations are saved and ready to be reused.

Download the VM

- Download ArmVM from http://files.pharo.org/vm/pharo-spur32/linux/armv6/latest.zip.
- Unzip it
- Copy the files shown in FIgure 1-8 to Pharo folder `/Users/your_user_name/Documents/Pharo/pharo_image_folder`

Copying Sources

Copy the file PharoV60.sources from the folder `/Applications/Pharo.app/Contents/MacOS` to folder `/Users/your_user_name/Documents/Pharo/images/pharo_image_folder/lib/pharo/5.0-201804182009/`

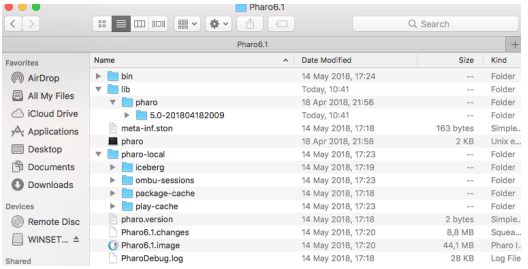

Figure 1-9 Copying the folder on your Raspberry.

Copy to the Raspberry

Copy this folder to your Raspberry Pi (via flashdrive, network etc). The folder must have the structure shown in Figure 1-9.

1.7 Execute PharoThings on Raspberry

Turn on your Raspberry and connect it to the network.

In this example, the folder Pharo6.1 was copied to folder `/home/pi/`.

Is necessary apply execute permissions on the Pharo files, using the command chmod +x

```
chmod +x /home/pi/Pharo6.1/pharo

chmod +x /home/pi/Pharo6.1/lib/pharo/5.0-201804182009/pharo
```

Start Server

Start the Pharo typing the following command in the Terminal on your Raspberry :

```
pharo --headless Server.image remotePharo --startServerOnPort=40423
```

If all is right, you will see the answer:

```
'a TlpRemoteUIManager is registered on port 40423'
```

So now we have the Raspberry running the TelePharo on TCP port 40423 (as shown in Figure 1-10) and we can connect into it from another computer.

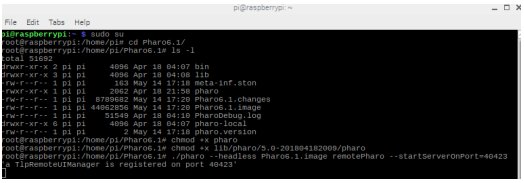

Figure 1-10 Server up and running.

1.8 Connecting Pharo client on remote Pharo

Open again the Pharo on your local computer and execute this command to install the PharoThings client:

```
Metacello new
baseline: 'PharoThings';
repository: 'github://pharo-iot/PharoThings/src';
load: 'RemoteDev.
```

Type this command to connect to the remote TelePharo on Raspberry (change the IP to your Raspberry IP):

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

Here we are using specialized Raspberry tools. They require the auto-refresh feature of inspector which is not enabled by default in Pharo 6. To activate it evaluate:

```
GTInspector enableStepRefresh
```

So for your board model you need to choose an appropriate board class. 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: [ RpiBoardBRev1 current].
remoteBoard inspect.
```

And will be open the inspector showing the PIN scheme (as shown in Figure 1-11)

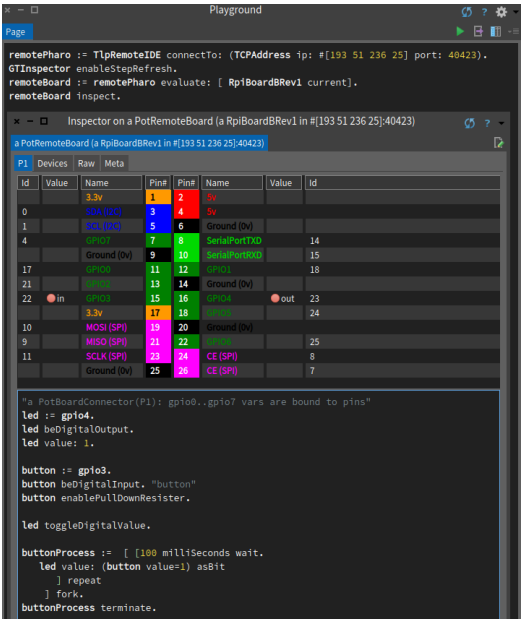

Figure 1-11 Remote GPIO inspector

GPIOs

The board inspector shows a layout of pins similar to Raspberry Pi docs. But here it is a live tool which represents the current pins state.

In the picture the board is shown with two configured pins: gpio3 and gpio4, which are connected to physical button and led accordingly.

Digital pins are shown with green/red icons which represent high/low (1/0) values. In case of output pins you are able to click on the icon to toggle the value. Icons are updated according to pin value changes. If you click on physical button on your board the inspector will show the updated pin state by changing its icon color.

The evaluation pane in the bottom of the inspector provides bindings to gpio pins which you can script by #doIt/printIt commands. The example shows expressions which were used to configure a button and led.

1.9 In the next chapter

Now that we have installed the Operation System and PharoThings in the Raspberry Pi, we can play with LEDs, sensors, LCD displays and more.

In the next chapter we will see how turing on/off a LED using PharoThings.

CHAPTER 2

Lesson 1 – Turning LED on/off

One of the classic analogies in electronics to "Hello World" is turn on a led or lamp. In this first lesson, we will learn how to connect correctly an LED to your Raspberry Pi and how to use PharoThings to control this led by turning it on and off.

Coding in Pharo is very simple, but it is very powerful and you can control all the GPIOs of your Raspberry Pi remotely.

If you did not yet see how to install the PharoThings on your Raspberry Pi and how to control it remotely, you can find the instructions in Chapter 1.

2.1 What we need?

In this lesson we will use a very simple setup.

Components

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

2.2 Experimental theory

Before constructing any circuit, you must know the parameters of the components in the circuit, such as their operating voltage, operating circuit, etc.

The LED

To turn on the LED, we need to send the correct voltage and current to it. The voltage and current can't be too high, otherwise, the led will burn, or in some cases, damage the Raspberry.

Typically, the forward voltage of an LED is between 1.8 and 3.3 volts. It varies by the color of the LED. A red LED typically drops 1.8 volts, but voltage drop normally rises as the light frequency increases, so a blue LED may drop from 3 to 3.3 volts[1].

Most 3mm and 5mm LEDs will operate close to their peak brightness at a drive current of 20 mA. This is a conservative current: it doesn't exceed most ratings (your specs may vary, or you may not have any specs–in this case, 20 mA is a good default guess [2])

In this experiment, the operating voltage of the LED is between 1.5V and 2.0V and the operating current is between 10mA and 20mA.

The Resistor

We must always use resistors to connect LEDs up to the GPIO pins of the Raspberry Pi to limit the voltage and current between the LED and the Raspberry to a safe value.

A small change in voltage can produce a huge change in current (see more: LED Current vs. Voltage [2])

In this experiment, we will use a 330ohm resistor. To identify the correct resistor, follow one of the following color sequences, depending on the number of bands [3]:

- If there are four colour bands, they will be Orange, Orange, Brown, and then Gold;
- If there are five bands, then the colours will be Orange, Orange, Black, Black, Brown.

It does not matter which way round you connect the resistors (in this experiment). Current flows in both ways through them. This means that you can connect the resistor at the positive pole or the negative pole of the LED, as well as starting with the first or last color, as shown in the Picture 2-1.

But the LEDs will only work if the power is supplied correctly (if the polarity is correct). You will not burn the LEDs if they connect the wrong way – they just will not turn on.

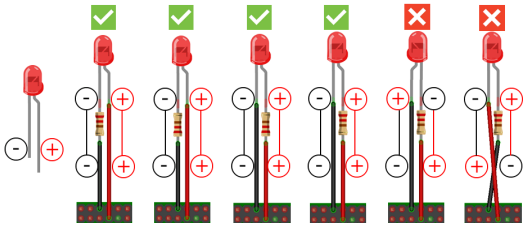

Figure 2-1 Led polarity and resistors.

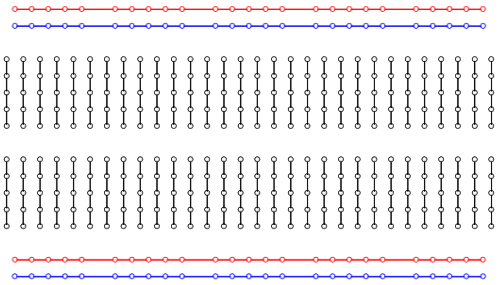

Figure 2-2 Breadboard scheme.

The Breadboard

A breadboard is used to build prototyping of electronics. With a breadboard, it is not necessary to use solder, this way you can reuse the board. This makes it easy to use to create temporary prototypes and experiment with circuit design.

The holes in the breadboard are connected following a pattern, as shown in the Picture 2-2.

- The red (+) and blue (-) rail are connected horizontally;
- The holes in the middle are connected vertically.

2.3 Experimental procedure

Now we will build the circuit. This circuit consists of an LED that lights up when power is applied, a resistor to limit current and a power supply (the Rasp).

- 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);

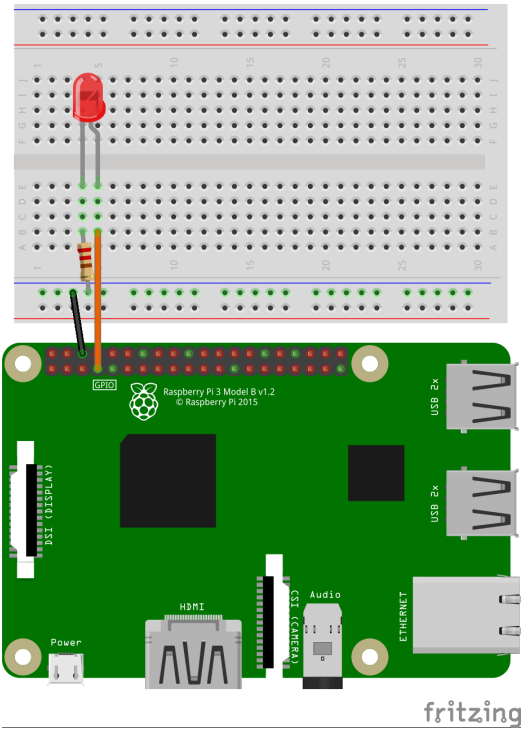

Figure 2-3 Physical connection LED.

- Then connect the resistor from the blue rail on the breadboard (-) to a column on the breadboard, as shown in the Picture 2-3;
- 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).

The Figure 2-3 shows how the electric connection is made:

2.4 Experimental code

Now, we can write some code in Pharo to control the GPIOs using PharoThings and turn the LED on. We have 2 options to do this:

- Write the code locally on the Raspberry;
- Use TelePharo to connect from your computer into the Raspberry and do all the work remotely.

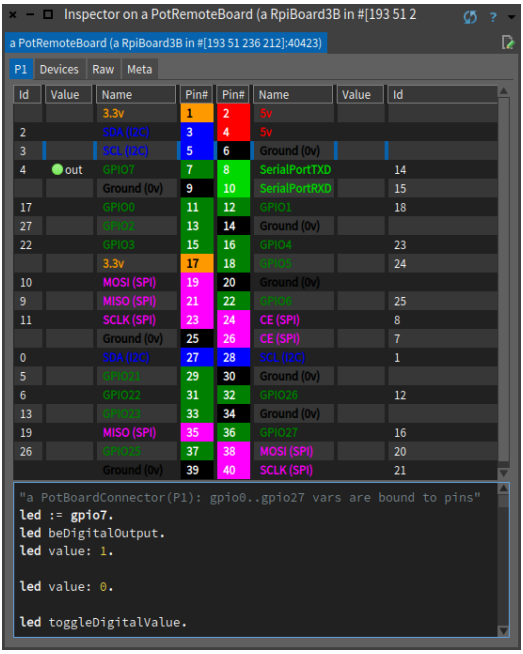

Figure 2-4 Remote Board Inspector.

In this experiment, we will use the second option: connect and do all the work remotely.

If you didn't see how to install the PharoThings on your Raspberry Pi and how to control it remotely, take a look in the Chapter 1: Installations.

Connecting remotely

Through your local Pharo image, let's connect in the Pharo image running on Raspberry, enable the auto-refresh feature of the inspector and open the inspector. Run this code in your local playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip: #[193 51 236
    167] port: 40423)
GTInspector enableStepRefresh
remoteBoard := remotePharo evaluate: [ RpiBoardBRev1 current].
remoteBoard inspect.
```

In your inspect window (Inspector on a PotRemoteBoard) you can see a scheme of pins similar to the Raspberry Pi docs. But here it is a live tool which represents the current pins state.

Digital pins are shown with green/red icons which represent high/low (1/0) values. In case of output pins you are able to click on the icon to toggle the

value.

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 turnes the led on.

You can notice that gpio variables are not just numbers/ids. PharoThings models boards with first class pins. 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.
```

2.5 What did we learn?

With PharoThings you can remotely control all the GPIOs in your running board!

You can:

- Interact remotely with pins and boards;
- See the current pins state in real time;
- Run the code dynamically.
- Easy, powerful.

2.6 In the next lesson

Let's use what we learned in this lesson and write a simple code to blink the LED.

CHAPTER 3

Lesson 2 – Blinking LED

Now we can play with the LEDs, turn them on and off. Let's use this basic setup to write some code on the inspector playground to blink the LED. Next, we will learn how to remotely create a very simple application using classes, methods, and instances to control the LED.

3.1 What do we need?

We are using the same setup as the last section: 1 Raspberry Pi, 1 Breadboard, 1 LED, 1 Resistor 330ohms. If you didn't do the last lesson to understand how to do the connections, go back to Chapter 2 and do it.

Connecting remotely

Through your local Pharo image, let's connect to the Pharo image by running on Raspberry, enable the auto-refresh feature of the inspector, and open the inspector.

Run this code in your local playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip: #[193 51 236
    212] port: 40423)
GTInspector enableStepRefresh.
remoteBoard := remotePharo evaluate: [ RpiBoardBRev1 current].
remoteBoard inspect.
```

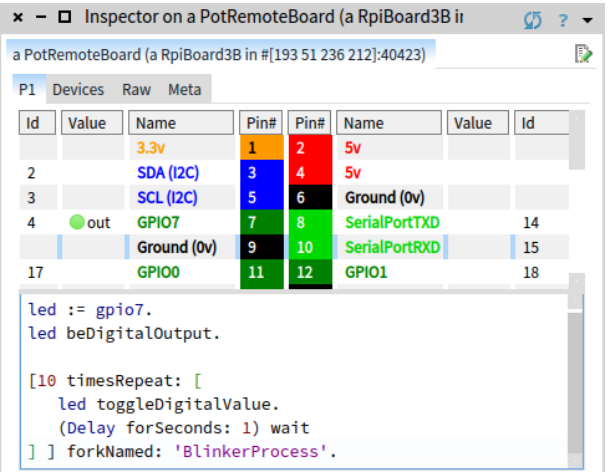

Figure 3-1 Remote playground.

3.2 Experimental code

In your inspect window (Inspector on a PotRemoteBoard), let's initialize the led and set the pin 7 to be in digital output mode as we did in the last lesson:

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

To blink the LED let's create a simple loop to change the value of the LED every 1 second by 10 times. To change the value of the object (led value), let's call the method `toggleDigitalValue`, as we saw previously:

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

Run this code, as shown in Figure 4-5 and... `cool`! Now your LED is blinking!

Change the values to repeat more times and to wait less time between toggling. This will cause the LED to blink faster.

3.3 In the next lesson

In this tutorial, you learned how to blink a LED by typing some code in the remote inspector. But with Pharo we can do more! And in the next lesson, let's start to use OOP (object-oriented programming). Let's create a simple application, write classes and methods, all remotely.

CHAPTER 4

A brief introduction to Pharo object-oriented language

Pharo is a new generation reflective language and programming environment. The last code was executed inside the remote inspector. To get started using OOP (Object-Oriented Programming) with classes, methods, and instances, I invite you to implement a simple application to blink the LEDs.

4.1 Developing a simple LED blinker

The following part of this chapter and application was based on the exercise `Developing a Simple Counter`, of Week 1 of Pharo MOOC (https://-mooc.pharo.org/). I strongly recommend that you read and do the "counter exercise" to better understand the concepts explained here. And, of course, do the MOOC to learn how to develop using Pharo and the OOP concept.

4.2 Our use case

We want to create a blinker LED using a few parameters such as time to repeat the blinking LED and how many seconds to wait between blinks. The following code should run in the playground when we finish this lesson:

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

Here is a short explanation of this code:

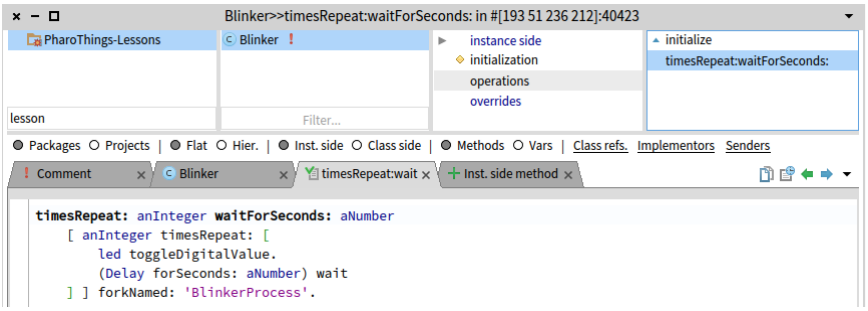

Figure 4-1 Remote System Browser.

- In the first line, we declare the variable `blinker`. We can use any name. We will use this variable to create an object using the Blinker class;
- In the second line, we instantiate the Blinker class (with uppercase B) in the `blinker` variable, creating an object. In this lesson, we will create this class and methods to control the LED;
- In the third line, we send some messages to the `blinker` object, for how long and how many times per second. This will make the GPIO behave according to the parameters sent.

Now we will develop all the mandatory class and methods to support this scenario.

4.3 Create your own class remotely

Let's create our first class. To create a class in Pharo, we need first to create a package. Inside the package, you can create many classes and inside the classes, you can create many methods. The methods are organized in protocols, to become more easily navigate between them. Take a look in Figure 4-1 to better understanding. *edit image, put name in windows

In your local playground, call the Remote System Browser of your Raspberry Pi. If you are already connected to your Raspberry Pharo, you do not need to run the first line below again. This will open a window as shown in Figure 4-1

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

4.4 Create a package

Let's create a package using the Remote Browser. Right-click the package area and enter the package name, as shown in Figure 4-2. In this example, we

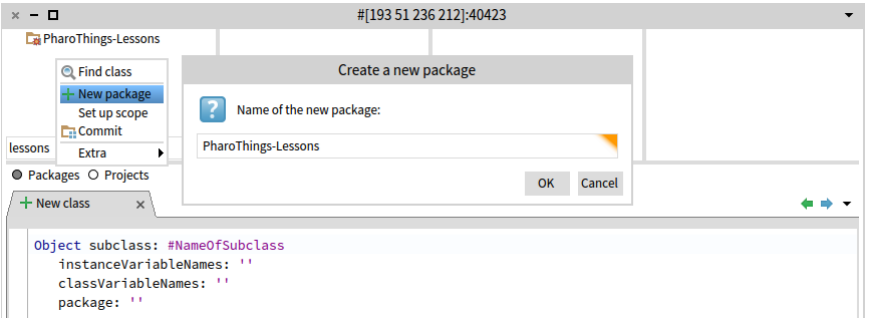

Figure 4-2 Creating a package remotely.

will create a package named `PharoThings-Lessons`.

4.5 Create a class

To create a new class, edit the default class template by changing the #NameOfSubclass to the name of the new class. In this example let's create the class `#Blinker`. Take care that the class name begins with a capital letter and that you do not remove the hash symbol (#) in front of NameOfSubClass.

You must then fill in the names of the instance variables for this class. We need an instance variable called `led`. Be careful to leave the string quotes!

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

Now we need to compile it. Right click on the code area and select Accept option. The `Blinker` class is now compiled and added to the system, as shown in Figure 4-3.

4.6 Create a protocol

Let's create a new protocol to organize the methods. The first protocol we are going to create is `initialization`, as shown in Figure 4-4.

4.7 Creating an initialize method

Inside this protocol, we will create a `initialize` method. This means that every time we create a new object using this class, in this case, the Blinker class, this method will be executed to define some variable in the new object.

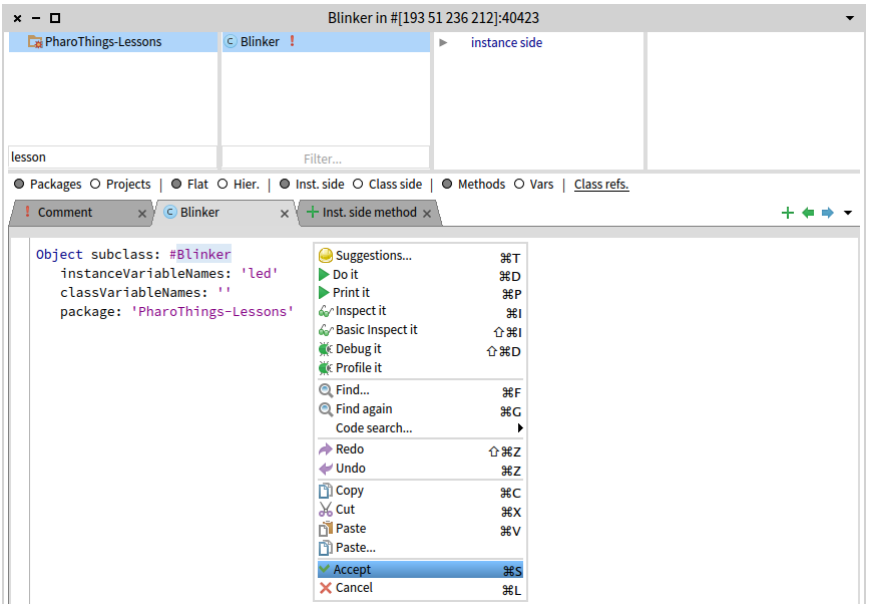

Figure 4-3 Creating a class remotely.

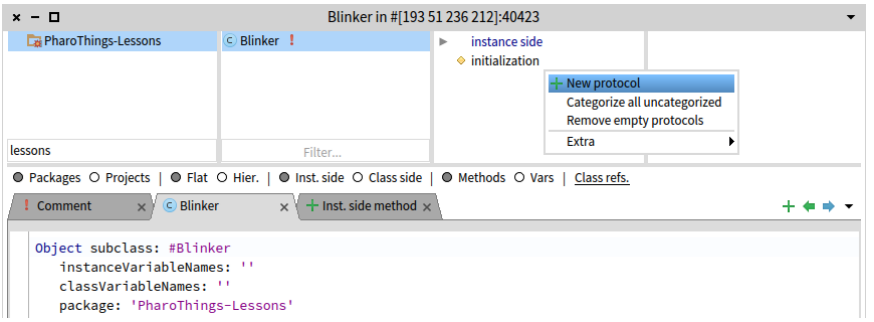

Figure 4-4 Creating a package remotely.

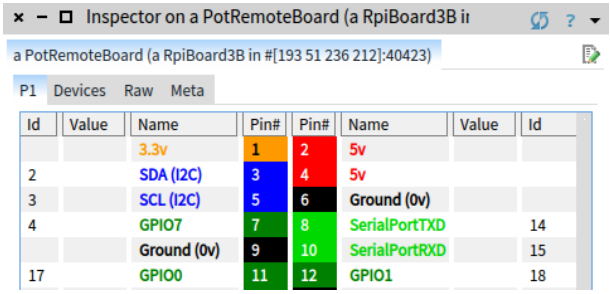

Figure 4-5 Looking for ID and GPIO number on Remote Inspector.

Let's use the instance variable `led`, which we defined when we created the class. 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
  led := PotClockGPIOPin id: 4 number: 7.
  led board: RpiBoard3B new; beDigitalOutput
```

Here is a short explanation of this code:

- 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 Wiring Pi Library. You can see the `ID` and `GPIO number` in PotRemoteBoard inspector, as shown in Figure 4-5;
- 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, as shown in Figure 4-6:

Creating a method to do actions

Now let's create a method to control the object led inside the class Blinker. Let's take the code that we used in PotRemoteBoard inspector to do the LED to blink and replace the numbers on code for two arguments. Create the protocol `operations` and inside this protocol, create the following method:

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

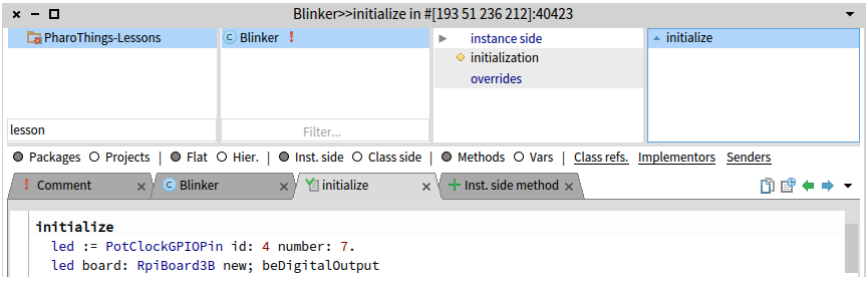

Figure 4-6 Creating the initialize method.

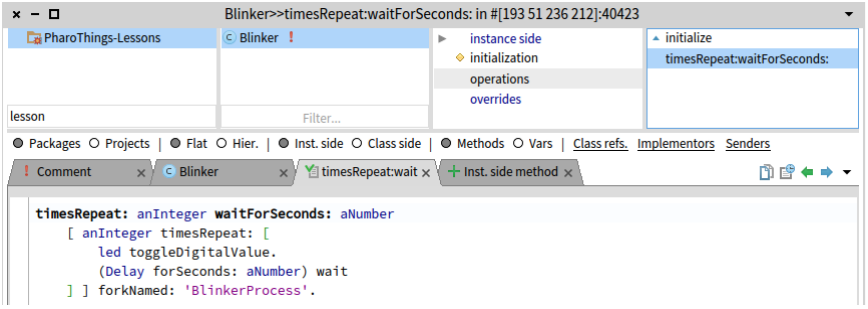

Figure 4-7 Creating an operation method.

Here is a short explanation of this code:

- 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.

Compile your code (cmd + S) and the method will be shown in the remote browser, as shown in Figure 4-7:

4.8 Using your new class

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

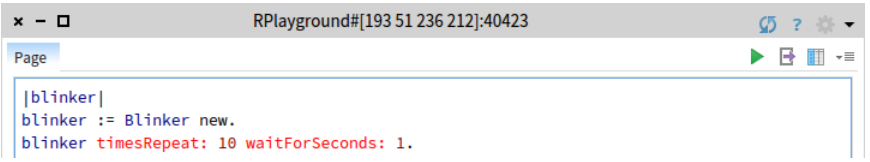

Figure 4-8 Remote playground.

```
remotePharo openPlayground.
```

and run the code that we saw in the begin of this lesson:

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

Run this code, as shown in Figure 4-8 and... `cool`! Now your LED is blinking! And the better, you did this using object-oriented programming!

You do not need to change your code every time you wanna change these parameters. Just change the messages you send to the object and it will behave as you want.

4.9 Save your work

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

```
remotePharo saveImage.
```

4.10 Conclusion

In this tutorial, you learned how to define packages, classes, and methods. The flow of programming that we chose for this first tutorial is similar to most of the programming languages.

With PharoThings you can remotely develop and manage your Raspberry GPIOs. Very easy and powerful.

In the next lesson, let's use what we learned in this lesson and write a simple code to flow lights using 8 LEDs.

CHAPTER 5

Lesson 3 - LED Flowing Lights

Now we can play with the LEDs, turn them on, off, and blink. Let's put 8 LEDs on the breadboard and create a code to turn on/off one at a time. As we did in the last lesson, let's write the first code in playground and then create a class with methods to better control the flow of LED lights.

5.1 What we need?

We are using the lesson 1 setup, but let's use 8 LEDs and 8 resistors.

Components

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

5.2 Experimental procedure

We saw in the lesson 1 how to connect the LED and resistors on the breadboard. Now let's do the same, but putting more 7 LEDs and resistors on the breadboard.

- Connect the Ground PIN from Raspberry in the breadboard blue rail (-).

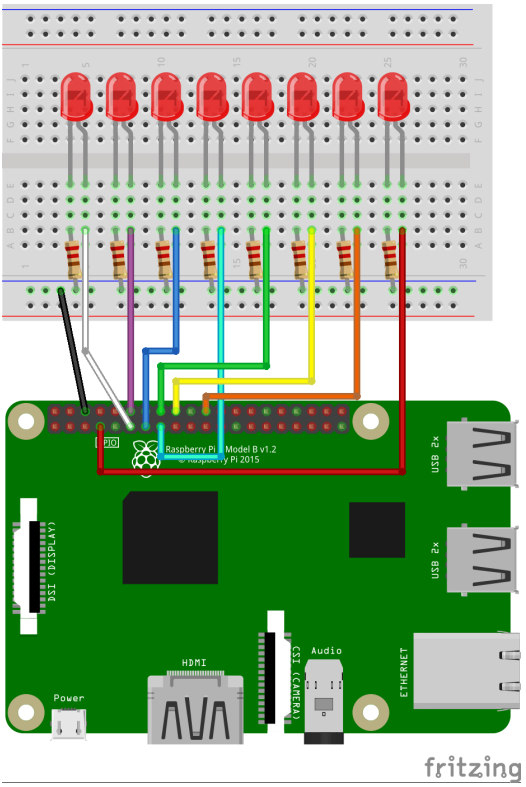

Figure 5-1 Physical connection 8 LEDs.

- Then connect the 8 resistors from the blue rail (-) to a column on the breadboard, as shown below;
- 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 rigth column of each LED to GPIO from 0 to 7, as shown in the Picture 5-1.

The Figure 5-1 shows how the electric connection is made:

Connecting remotely

Through your local Pharo image, let's connect in the Pharo image by running on Raspberry, enable the auto-refresh feature of the inspector, and open the inspector.

Run this code in your local playground:

```
remotePharo := TlpRemoteIDE connectTo: (TCPAddress ip: #[193 51 236
    212] port: 40423)
GTInspector enableStepRefresh.
remoteBoard := remotePharo evaluate: [ RpiBoardBRev1 current].
remoteBoard inspect.
```

5.3 Experimental code

In your inspect window (Inspector on a PotRemoteBoard), let's initialize the 8 LEDs and set the GPIOs to be in digital output mode:

```
code...
```

To blink the LED let's create a simple loop to change the value of the LED every 1 second by 10 times. To change the value of the object (led value), let's call the method `toggleDigitalValue`, as we saw previously:

```
code...
```