

Raspberry and Pharo

Pharo run on RaspberryPI

- ArmVM: <http://files.pharo.org/vm/pharo-spur32/linux/armv6/latest.zip>
- JIT
- FFI
- OSProcess/OSSubprocess
 - <https://github.com/marianopeck/OSSubprocess>

Low level GPIO libraries

- WiringPI bindings from Jean Baptiste
 - <http://smalltalkhub.com/#!/~Pharo/loT>
 - docs <http://wiringpi.com>
- Pigpio bindings from Tim Rowledge
 - <http://www.squeaksource.com/HardwarePeripherals.html>
 - docs <http://abyz.co.uk/rpi/pigpio/>
 - Required old compiler to install in Pharo

Remote development of Raspberry

1. Prepare Raspberry image

- Download Pharo 6 and install server part of PharmIDE

Metacello new

```
smalltalkhubUser: 'Pharo' project: 'PharmDIE';  
configuration: 'PharmIDE';  
version: #stable;  
load: 'Server'.
```

2. Save image with running server where IDE will connect

```
PrmRemoteUIManager registerOnPort: 40423
```

3. Or start image on Raspberry with command line option

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

Remote development of Raspberry

- Prepare development image:
 - Download Pharo 6 and install client part of PharmIDE

```
Metacello new
  smalltalkhubUser: 'Pharo' project: 'PharmDIE';
  configuration: 'PharmIDE';
  version: #stable;
  load: 'Client'.
```

- Connect to running Raspberry image from playground:

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

- Script Raspberry from remote playground:

```
remotePharo openPlayground
```

- Browse/edit Raspberry image from remote browser:

```
remotePharo openBrowser
```

Remote playground

```
RPlayground#[193 51 236 167]:40423
Page
OSSUnixSubprocess new
  command: 'uname';
  arguments: #('-a');
  redirectStdout;
  runAndWaitOnExitDo: [ :process :outString |
    ^outString
  ].
"'Linux raspberrypi 3.12.28+ #709 PREEMPT Mon Sep 8 15:28:00 BST 2014 armv6l GNU/Linux
'".

lib := WiringPiLibrary uniqueInstance.
lib wiringPiSetupGpio.
lib pin: 18 mode: 1. "output"
lib pin: 18 write: 1. "led on"
lib pin: 18 write: 0. 'led off'
```

Remote browser

The screenshot shows an IDE window titled "WiringPiLibrary>>pin:mode: in #[193 51 236 167]:40423". The left sidebar displays a project tree with folders like "WiringPi-Core", "WorldMenuHelp", "WriteBarrierTests", and various "Zinc-Character-Encoding" and "Zinc-Resource-Meta" folders. The main editor area shows the "WiringPiLibrary" class with a list of "inherited methods" including "FFI - Concurrent Processing", "FFI - Core", "FFI - Interrupts", "FFI - RaspberryPi", "FFI - Setup", "FFI - Thread Priority", "FFI - Timings", "accessing platform", "initialization", and "overrides". The "pin:mode:" method is highlighted in the list. Below the list, there are tabs for "Comment", "WiringPiLibrary", and "pin:mode:". The "pin:mode:" tab is active, showing the following code and documentation:

```
pin: pinNumber mode: mode
```

"This sets the mode of a pin to either INPUT=0, OUTPUT=1, PWM_OUTPUT=2 or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.

This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program."

```
^self ffiCall: #(long pinMode(long pinNumber, long mode))
```

At the bottom of the window, there is a status bar with "1/10 [1]" on the left and "Format as you read W +L" on the right.

Online docs on GPIO

projects.drogon.net

P1: The Main GPIO connector:

wiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	wiringPi Pin
—	—	3.3v	1 2	5v	—	—
8	R1:0/R2:2	SDA	3 4	5v	—	—
9	R1:1/R2:3	SCL	5 6	0v	—	—
7	4	GPIO7	7 8	TxD	14	15
—	—	0v	9 10	RxD	15	16
0	17	GPIO0	11 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 14	0v	—	—
3	22	GPIO3	15 16	GPIO4	23	4
—	—	3.3v	17 18	GPIO5	24	5
12	10	MOSI	19 20	0v	—	—
13	9	MISO	21 22	GPIO6	25	6
14	11	SCLK	23 24	CE0	8	10

High level tools

- Low level libraries are not object based
- Pharo IoT project
 - Load to Raspberry image with:

```
Metacello new
```

```
  smalltalkhubUser: 'Pharo' project: 'IoT';  
  configuration: 'IoT';  
  version: #stable;  
  load: 'RemoteToolsServer'.
```

- Load to client dev image with:

```
Metacello new
```

```
  smalltalkhubUser: 'Pharo' project: 'IoT';  
  configuration: 'IoT';  
  version: #stable;  
  load: 'RemoteToolsClient'.
```

Pharo IoT project

- Includes remote development tools
- Simple object model for boards
 - pins are objects
 - hierarchy of boards with specific configuration of pins
 - RpiModelBRev1 with single connector P1
 - RpiModelBRev2 with two connectors P1 and P2
 - more in future
 - BeagleBoard's in future
- Advanced tools to manage peripherals

```
remoteBoard := remotePharo evaluate: [ RpiBoardBRev1 current].  
remoteBoard inspect
```

Inspector on an IotRemoteBoard (a RpiBoardBRev1)

an IotRemoteBoard (a RpiBoardBRev1)

P1 Devices Raw Meta

Id	Value	Name	Pin#	Pin#	Name	Value	Id
		3.3v	1	2	5v		
0		SDA (I2C)	3	4	5v		
1		SCL (I2C)	5	6	Ground (0v)		
4		GPIO7	7	8	SerialPortTXD		14
		Ground (0v)	9	10	SerialPortRXD		15
17		GPIO0	11	12	GPIO1	● out	18
21		GPIO2	13	14	Ground (0v)		
22		GPIO3	15	16	GPIO4		23
		3.3v	17	18	GPIO5		24
10		MOSI (SPI)	19	20	Ground (0v)		
9		MISO (SPI)	21	22	GPIO6	● in	25
11		SCLK (SPI)	23	24	CE (SPI)		8
		Ground (0v)	25	26	CE (SPI)		7

```

"an IotBoardConnector(P1): gpio0..gpio7 vars are bound to pins"
led := gpio1.
led beDigitalOutput.
led value: 1.
led value: 0.
led bePWMOutput.
led value: 100.

button := gpio6.
button beDigitalInput. "button"
button enablePullUpResister.
button value. "1"

```

Devices model to program physical things connected to board

Inspector on an IotRemoteBoard (a RpiBoardBRev1)

an IotRemoteBoard (a RpiBoardBRev1)

P1 Devices Raw Meta

Id	Value	Name	Pin#	Pin#	Name	Value	Id
		3.3v	1	2	5v		
0		SDA (I2C)	3	4	5v		
1		SCL (I2C)	5	6	Ground (0v)		
4		GPIO7	7	8	SerialPortTXD		14
		Ground (0v)	9	10	SerialPortRXD		15
17		GPIO0	11	12	GPIO1	● out	18
21		GPIO2	13	14	Ground (0v)		
22		GPIO3	15	16	GPIO4		23
		3.3v	17	18	GPIO5		24
10		MOSI (SPI)	19	20	Ground (0v)		
9		MISO (SPI)	21	22	GPIO6	● in	25
11		SCLK (SPI)	23	24	CE (SPI)		8
		Ground (0v)	25	26	CE (SPI)		7

```
"an IotBoardConnector(P1): gpio0..gpio7 vars are bound to pins"  
button := board installDevice: (IotButton fromGroundTo: gpio6).  
switch := board installDevice: (IotSwitch for: gpio1 using: button).
```

Inspector on an lotRemoteBoard (a RpiBoardBRev1)

an lotRemoteBoard (a RpiBoardBRev1)

P1 Devices Raw Meta

Name	Status	Peripherals
Button	on	GPIO6 ● in ; Ground (0v)
Switch	on	GPIO1 ● out ; Button

Remote debugger

The screenshot shows a remote debugger interface for an IoT-Button device. The window title is "IoT-Button>>checkState in #[193 51 236 167]:40423".

The interface is divided into several sections:

- Left Panel (Project Explorer):** Shows a tree view of the project structure, including folders like "IoT-Devices-Button", "IoT-Devices-Switch", "IoT-Hardware-Core", "IoT-Hardware-RaspberryPi", "IoT-RemoteToolsServer", "IssueTracking", "IssueTracking-Tests", "JQuery-Core", "JQuery-JSON", and "JQuery-Tests-Core".
- Middle Panel (Class Explorer):** Shows the class hierarchy for "IoT-Button". The class is expanded to show its methods: "lotButtonPressed", "lotButtonReleased", "lotGroundPin", and "lotPowerPin".
- Right Panel (Method Explorer):** Shows a list of methods for the selected class. The "checkState" method is highlighted. Other methods include "announceState", "accessing", "controlling", "initialization", "private", "subscription", "testing", "transfer", and "overrides".
- Bottom Panel (Code Editor):** Shows the source code for the "checkState" method. The code is as follows:

```
checkState
| currentState |
currentState := gpioPin value.
lastState ~= currentState ifTrue: [
  self halt.
  lastState := currentState.
  self announceState ]
```

The bottom of the window shows a status bar with "6/8 [13]" and a "Format as you read" checkbox.

Deploy

- Save image at the end

`remotePharo saveImage`

- On start up all board state is recovered
- Set up image as service with Linux tools

Future

- More RaspberryPI models
- Beaglebone models
- Deploying as service from image
- Zeroconf for armVM+IoT
- Improve code management
- General evolution of PharmIDE
 - Automatic detection of running images in network
 - Remote refactoring
 - Security
 - many other things

The end