# RGPIO terminology

- server : the central system where the application runs that uses RGPIO. Network traffic runs between server and devices (not between devices).

- device : the actual physical remote devices. Each device has a unique identifier (HWid)

- device model : describes devices that have the same behaviour and interface (same pins). Device models are read from a configuration file when RGPIO starts. Devices report their model name.

- device group : represents  devices that are treated collectively by the application. E.g. "garden lights" that are all switched on and off together, based on time and/or events from sensors.

# Example of an application that uses RGPIO

The environment has a motion sensor , one indoor siren and several outdoor sirens, and a number of garden lights. We want the indoor and outdoor sirens to go off and the garden lights to switch on during 1 minute when the backdoor sensor detects movement.

The structure of the application is:

1. Initialize RGPIO : this reads the description of the device models
2. Declare the device groups
3. Start the RGPIO device monitoring thread. Devices broadcast their HWid and model name when powering up. The monitor periodically broadcasts to solicit device reports.
4. Response to events is implemented in event listener(s)

**class ExampleClass implements RGPIODigitalPinEventListener {**

**RGPIODeviceGroup binnenSirene;**

**RGPIODeviceGroup buitenSirene;**

**RGPIODeviceGroup achterdeurSensor;**

**RGPIODeviceGroup tuinlampen;**

**public void onDigitalPinEvent(RGPIODigitalPinEvent event) {**

**if ((event.device.deviceGroupName.equals("achterdeur"))**

**&& (event.pin.label.equals("movement"))**

**&& (event.state)) {**

**buitenSirene.getPin("active").setState(true);**

**binnenSirene.getPin("active").setState(true);**

**tuinlampen.getPin("on/off").setState(true);**

**Thread.sleep(1000);**

**buitenSirene.getPin("active").setState(false);**

**binnenSirene.getPin("active").setState(false);**

**tuinlampen.getPin("on/off").setState(false);**

**}**

**}**

**public void start(){**

**RGPIO.initialize(System.getProperty("user.home") + "\\Documents\\");**

**buitenSirene = new RGPIODeviceGroup("SIRENE", "buitensirene", 1, 1);**

**binnenSirene = new RGPIODeviceGroup("SIRENE", "binnensirene", 1, 1);**

**achterdeurSensor = new RGPIODeviceGroup("PIR", "achterdeur", 1, 1);**

**tuinlampen = new RGPIODeviceGroup("RELAY", "tuinlamp", 0, 999);**

**RGPIO.startDeviceMonitor();**

**achterdeurSensor.getPin("movement").addDigitalPinListener(this);**

**RGPIO.wait(); // prevents that start() exits**

**}**

**}**

# API description

Devices do not appear individually in an application but are addressed as device groups. The device groups that will be used in the application are declared in the beginning, for example

       RGPIODeviceGroup tuinlampen = new RGPIODeviceGroup("RELAY", "tuinlamp");

* The string "tuinlamp" is the name of the device group
* The string "RELAY" is the device model.
* The device group can have any number of devices (including 0)

If the application wants to address a specific device it must be declared in its own device group.

       RGPIODeviceGroup frontDoorPIR = new RGPIODeviceGroup("PIR", "frontDoor”);

You can ask to be warned when the number of devices in a group exceeds certain limits:

frontDoorPIR.setMinimum(1);

frontDoorPIR.setMaximum(1);

A device group has pins. Names and type of pins are described in the device model.

* Setting a pin on a device group to HIGH or LOW causes the corresponding command to be sent to all its member devices.
* A pin event from one device in the group is handled as an event for the device group.
* Reading a pin of a device group returns a list of the values sent by the devices.

An example of a device group could be motion sensors where it does not matter to the application which one detects movement.  
  
Working with device groups allows an application to run with all, none, or a part of the remote devices active, and their numbers may change at any time The application may lose part or all of its functionality but will not crash, does not have to probe for new devices nor do error checking.

Use case:

There is a motion sensor at the back door of the house. You declare it as a device group with setMinimum(1). For you this is absolutely critical and hence you decide to put a second one for redundancy. This requires no change to the application. The second device will report to RGPIO and start working in parallel with the first one. When either of them trigger, the application is notified. When both are broken, a warning is generated. If you want to be sure that both are working, set the minimum to 2.

# Device discovery and assignment to a device group

RGPIO knows about the existence and address of devices by means of "REPORT" messages sent by the devices. A REPORT message contains the HWid of the device, its device model, and its uptime in seconds.  
Devices broadcast REPORT messages when they power up.  
RGPIO broadcasts REPORT requests periodically.  
From an incoming REPORT message, RGPIO knows the IP address of the device.  
From an incoming REPORT broadcast request, the device knows the IP address of the server.

When a device does not reply to several REPORT requests from the server, it is set to NOTRESPONDING. A warning is generated if the minimum number of the devices in the group is no longer met.  
  
When RGPIO receives a REPORT messages from a device, it tries to determine to which group it belongs as follows:

1. From RGPIODevices.txt. This file contains entries of the format:

af3c45e6:PIR:front

When the device with HWid=af3c45e6 reports, it is assigned to the group "front"

1. When there is only one device group with this model as the device, the device is put in that group
2. When a reporting device can not be assigned to a group, RGPIO passes the HWid, model and power-on time of the device to the application via the message listener described further, and the application should pass this information somehow to the user. By switching the devices on in a certain sequence, the user can determine the HWid's, enter them in  RGPIODevices.txt and restart the application. This procedure does not require the owner to know the burned-in HWid of the device.

# Device interfaces

An application can interface with devices in two ways: via pins and via commands.

**Device pins**

A digital pin is a way to exchange (read/write) a Boolean value with the device. Writing a value is done via the setState(Boolean) method of the pin, reading a value is done via the getState() method.

An analog pin is a way to exchange (read/write) an Integer value with the device. Writing a value is done via the setValue(Integer) method of the pin, reading a value is done via the getValue() method.

Examples

RGPIODeviceGroup relay = new RGPIODeviceGroup("GenericESP", "relay"); relay.getPin("GPIO01").setState(true);

RGPIODeviceGroup indoorSiren = new RGPIODeviceGroup("SIREN", "indoorSiren"); indoorSiren.getPin("Volume").setValue(50);

A device can report a change of state/value of an input pin asynchronously (e.g. when someone pushes a button). This is called a pin event. For a digital pin (analog pin uses the same principle):  
The device sends an EVENT message to RGPIO. RGPIO creates a DigitalPinEvent object and calls all the objects that have been added as listeners to this pin. From the information in the DigitalPinEvent object, the listener can decide what to do. Listeners must implement the RGPIODigitalPinEventListener interface.  
  
  
example:      frontDoorPIR.getPin("movement").addDigitalPinListener(this);  
  
**Device commands**  
  
Certain information can not elegantly be sent to a device using the pin interface model. For instance a command that requires more than one argument or that can have more than two discrete values. Examples:  
  
LOWPOWER:howlong:howdeep  
TRIGGER:gpio2:delta:interval

The last command could tell the device that it should send an EVENT command for the analog input pin gpio2 every 'interval' seconds or when the change is more than 'delta' volt.  
  
The application can send commands to the devices in a group. The list of commands that are understood by a device are described in the device model.

# Messages

RGPIO generates messages when encountering anomalies, but has no channel to communicate these messages to the user.It does not print or log these messages. Instead it calls a listener that is implemented by the application. The listener gets a RGPIOMessageEvent object with all information and can decide on an appropriate action (e.g. light a red LED, send an e-mail, log the message, or discard it).

Before calling any other RGPIO code the application must register the listener:

RGPIO.addListener(messageListener);  
  
The object messageListener must implement the interface RGPIOMessageListener, example :  
  
    public void onError(String message) throws Exception {  
        System.out.println("ERROR " + message);  
        System.exit(-1);  
    }  
  
    public void onWarning(String message) throws Exception {  
        System.out.println("Warning " + message);  
    }  
  
    public void onDebugInfo(String message) {   // ignore debug messages  
    }

# Protocol

UDP messages are exchanged between server and device.

* The server listens on port 2222 to requests from devices
* All devices listen on port 3333 to requests from the server
* Both server and devices expect a reply packet and retransmit RGPIO.retries times if the reply does not arrive within the timeout of RGPIO.timeout

**Description of the messages**

%s : alphanumerical string

%d : integer

%d : boolean HIGH or LOW

%p = %b for digital pins, %d for analog pins

|  |  |
| --- | --- |
| **Request from server** | **Reply from device** |
|  |  |
| Report | ACK (and sends Report request to port 2222) |
| Set/Pin:%s/Value:%p | ACK |
| Get/Pin:%s | %p |
| System/Command:%s/arg1:%s/arg2:%s… | ACK |
|  |  |
| **Request from device** | **Reply from server** |
|  |  |
| Report/HWid:%s/Model:%s/Uptime:%d | ACK |
| Event/HWid:%s/Model:%s/Pin:%s/%p | ACK |

Besides the requests above, the device may understand commands that can be sent by the application. The commands, arguments and type are specified in the device model. The application can send a command string to the device and is responsible for its correctness.

**Invalid requests**

RGPIO checks the reply from a device. If the syntax does not correspond to what is listed in the table above, the reply is supposed to be an error message and is forwarded as such to the RGPIO listener.

**Retransmits**

RGPIO retransmits a server request a few times and waits a few seconds for a reply. If the last timeout expires, an error message is sent to the RGPIO listener (except for ‘Report’ broadcasts).