RGPIO – Remote GPIO java library

The RGPIO library extends the PI4J library that controls the GPIO pins of a Raspberry PI (further: RASPI), to include the GPIO pins of remote devices. These remote devices can be ESPxxx or other RASPIs. The remote devices connect to the central RASPI via WiFi.

Using RGPIO facilitates implementing a project with one central RASP that controls its own GPIO pins but also those from remote devices:

# Design principles

* All devices for use with RGPIO are programmed identically, making them interchangeable and easily replaceable.
  + ESP devices exist in different models. Some have analog pins and more or fewer digital pins than others. The ESP must be able to communicate its model to the RASPI
  + Raspberry Pis also exist in different models, the same applies.
* Remote devices receive an IP address via DHCP from the WiFi router on power-on. They then report to the RASPI and repeat this on regular intervals. The RASPI therefore knows which devices are alive and can take action if it does not receive updates in time.
* The remote devices do not know the IP address of the RASPI. They report to the RASPI by sending a UDP datagram to the broadcast address 255.255.255.255. From this datagram the RASPI discovers the IP address of the device.
* Remote devices have a unique hardware identifier from the factory. In a specific environment, they are wired to perform a certain function – e.g. they control a specific relay switch. We also assign an identifier to this function. The RGPIO program needs to be able to couple the hardware identifier to the function identifier. A way to do this is to manually power up one remote device at a time. RGPIO receives the broadcast containing the hardware identifier and checks if it is already known. If not, user interaction is required to couple the hardware identifier to a function identifier.
  + When an ESP is out for some time or is replaced by another one, the above mechanism assures continuity. In the latter case, the new hardware identifier must manually be to its function by the user.
  + When the RASPI is out for some time, the ESP’s are not aware of this. In any case they can not take a specific action since they are programmatically identical.
  + When the WiFi is out for some time: ? zal later blijken…
* Function identifiers have to be assigned by the user of the system to each individual GPIO pin. At power-up of a remote device, the user will therefore be asked to specify the function of each pin that serves a purpose, for instance, when a 4-pin ESP with GPIO 0 and 1 in use comes up, the user has to provide the function of these two pins.

|  |  |  |
| --- | --- | --- |
| 45:22:af:d3 | GPIO 0 | Boiler Relay |
| 45:22:af:d3 | GPIO 1 | Camera on/off |
| 45:22:af:d3 | GPIO 2 |  |
| 45:22:af:d3 | GPIO 3 |  |

* When all remote devices are identified and working, they can be controlled in a uniform way using the function identifier:
  + Pin=RGPIO.getPin(“Camera on/off”)
  + Pin.getState();
  + Pin.setState(Boolean: state);
  + Pin.addListener();
  + … analoge pinnen ?

# Implementation overview

* RGPIO provides a class RGPIODispatcher(). The application on RASPI has to create an instance of this class before any pins can be used. The constructor of RGPIODispatcher starts a thread that listens for broadcasts and builds a directory of available GPIO pins.
* Classes exist for different pin types. Different pin types have different methods and associated events. Listeners can be added to the pins for these events.
  + RGPIODigitalOutputPin
    - setState(Boolean state)
    - boolean getState()
  + RGPIODigitalInputPin
    - addListener(listener)

The class of listener must implement the DigitalPinStateChangeListener interface, with method(s):

void doOnDigitalPinStateChange(DigitalPinStateChangeEvent event)

The event contains the pin for which the event was fired and the new pin state.