Pushing Performance

People | Power | Partnership

HARTING HAIIC MICA GPIO Container

2. Edition 2016, 07/16
Doc No 20 95 100 0003 / 99.01

This application note explains how to use the GPIO container and demonstrates the json-rpc calls to configure the GPIOs.

Contents

1 GPIO Container Basics

1.1 Overview

The GPIO container offers interfaces to access the GPIOS described in the Hardware Development Guide. A webserver runs on the container with the following features:

-A webGUI to configure the GPIOs

-A websocket interface to access the GPIOs over JSON-RPC calls

-SSO-Integration to allow access only after authentication

MQTT can be used as a transport protocol to other applications. Events can be defined for each GPIO pin in order to subscribe outputs for being set by published messages or on the other hand pins configured as input publish MQTT state messages every 200 ms. The state information is sent by the MQTT payload as int number (0 for off and 1 for on).

1.2 Installation

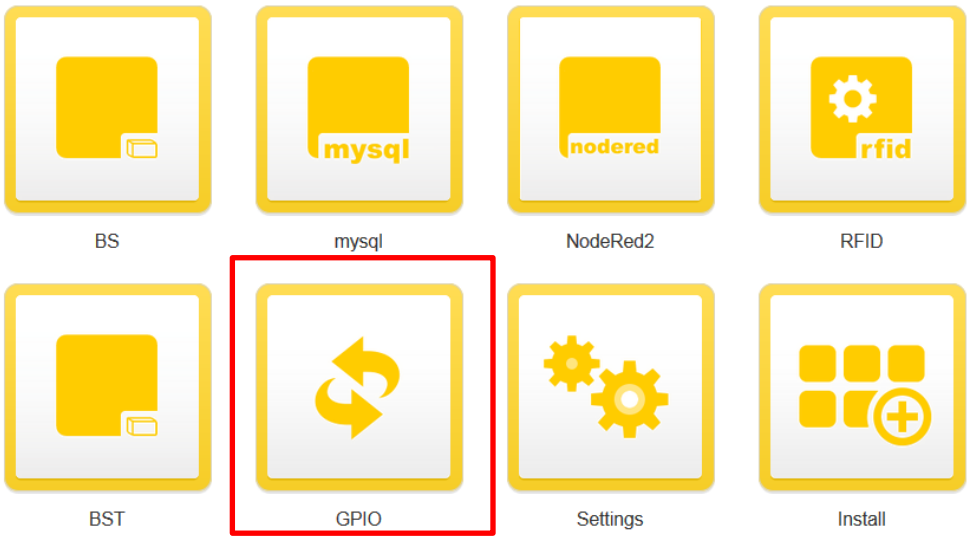

Figure 1: MICA Homescreen Including a Highlighted GPIO Container

The installation and configuration routine of the GPIO Container follows the standard routine as provided by the IIC MICA and can be found in the “MICA Programming Guide”.

2 User-Interface of the GPIO Container

2.1 Overview of the User Interface

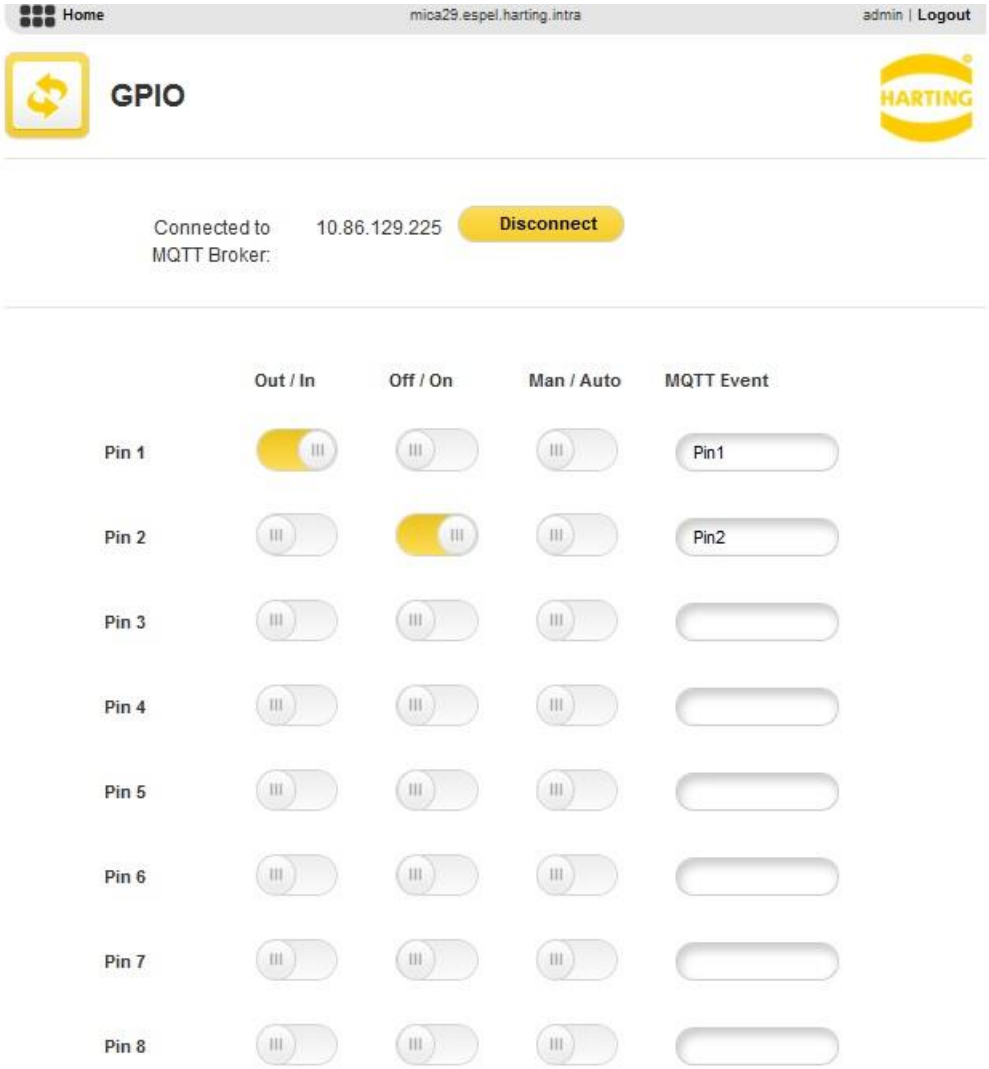

Figure 2: User Interface of the GPIO Container

The GPIO Container provides a web application to configure and set GPIO pins comfortably via MICA Web-GUI. Just click on the installed Container Icon. You can turn each pin on or off via the second button. GPIO channels can be reconfigured from Input to Output via the first button. Via On/Off Button you can set the state for output pins. The GPIO web-gui uses a JSON-RPC websocket-service to access GPIO hardware.

As an alternative approach MQTT is implemented communicate to GPIO, though configuration for MQTT

(subscribing/unsubscribing/registering publishers/ broker) is done by JSON-RPC calls via websockets. **Caution: Check that all channels are in default configuration when attaching them to a new hardware device!**

2.2 Accessing the GPIOs over JSON-RPC Calls

One way to configure the GPIOs is by using a websocket connection.

Example: Connecting to the Websocket Service of the GPIO container from the Python 3.4 Demo Container 1.1.0

Configure the Python Container as described in the Programming Guide and connect it to the internet as described in Chapter “GUI Programming”. Access the WebGUI of the Python 3 Demo Container. Start a new project and a new script by inserting a project and a script name in the Python editor on the webpage and clicking enter. In case you want to use another container make sure Python3, pip3, urllib3 and websockets are installed.

Enter the following Python Script:

```
import sys, json, websocket, ssl, urllib3
import base64
import hashlib
from base64 import b64encode


MICA = "mica-cp.local"       # MICA Base Name, works only if base
and container configured with ipv4
GPIO_CONTAINER = "GPIO3"     # GPIO Container Name
ROLE = "admin"
PW = "admin"

passwd_b64 = str( b64encode( bytes( PW, "utf8" ) ), "utf8" )
service_url = "https://"+MICA+"/base_service/"
rpc_obj = {
    "jsonrpc": "2.0",
    "id":1,
    "method":"get_auth_token",
    "params":{
        "user": "admin", "pwd": passwd_b64
        }
}
http = urllib3.PoolManager(cert_reqs= ssl.CERT_NONE,
assert_hostname=False, ca_certs="/META/harting_web.crt")
```

```
at = http.urlopen( "POST", service_url, body=json.dumps( rpc_obj ) )


rep_data_str = str(at.data, "utf-8")
ret = json.loads (rep_data_str)
ret1 = ret["result"][1]

ws_url = "wss://"+MICA+"/"+GPIO_CONTAINER+"/"

wesckt = websocket.create_connection( ws_url,sslopt = {"cert_reqs":
ssl.CERT_NONE, "ca_certs":"/META/harting.crt",
"check_hostname":False})

call = {"id": 1, "method": "login", "params": [ret1]}
wesckt.send( json.dumps(call) )
print( json.dumps(call) )
result_str = wesckt.recv()
print (result_str)
call = {"id": 1, "method": "get_pin_states", "params": [0]}
wesckt.send( json.dumps(call) )
result_str = wesckt.recv()
print (result_str)
```

The response should look like this:

```
{"id":1,"results":[{"direction":1,"event":"","state":0},{"direction"
:0,"event":"","state":0},{"direction":1,"event":"","state":0},{"dire
ction":0,"event":"","state":0},{"direction":0,"event":"","state":0},
{"direction":0,"event":"","state":0},{"direction":0,"event":"",
"state":0},{"direction":0,"event":"","state":0}]}
```

Further rpc-calls to configure the pins are described in the following. For convenience purposes, id and json-rpc properties are left out, Symbols: >>>: defines request from client to service, <<<: defines response from service.

Set state of pin

Method: set_state

Parameters: gpio_pin_number (integer value between 0 and 7 (pin 1 to pin 8)),

state (integer value between 0 (off) and 1 (on))

Return Value: Configuration of pin in case of success, error message including error code in case of error

Set pin 1 on

```
>>> { "method" : "set_state", "params" : [ 0, 1 ] }
```

```
<<< { "result" : 1 }
```

Set pin 1 on (but configured as input)

```
>>> { "method" : "set_state", "params" : [ 0, 1 ] }
<<<{ "error":{"code":-32094,"data":"Pin configured as Input!","mes-
sage":"Server Error"} }
```

Set direction of pin

Method: set_configuration

Parameters: gpio_pin_number (integer value between 0 and 7 (pin 1 to pin 8)),

direction (integer value between 0 (output) and 1 (input))

Return Value: Configuration of pin in case of success, error message including error code in case of error

Example:

Set pin 1 to input

```
>>>{ "method" : "set_configuration", "params" : [ 0, 1 ] }
<<<{ "result" : [0,1,0,false] }
```

Get pin information

Method: get_pin_states

Parameters: integer between 0 and 10

Return Value: json array of all pins including state, direction and MQTT event topic information

Example:

```
>>>{ "method" : "get_pin_states", "params" : [0] }
<<<{"result": [{"direction": 1,"event":"","state":0},{"direc-
tion":0,"event":"pin2","state":0},...,{"direc-
tion":0,"event":"pin4","state":1}]}
```

When using MQTT connect to a broker by typing in an address or mdns-name in the text field on top of the container website. Click the connect button. The Man/Auto switch can then be used to register topics for each pin. Man means you can set input/output configuration via RPC and set state while configured as output. After turning the Button to Auto means the pin becomes a subscriber if set to output or a publisher if set to input. While turned to Auto you will not be able to use In/Out and On/Off Button of the according pin anymore.

Input information is published only if state changes of the according pin occured.

MQTT state information is published / expected by subscribed pin in JSON Format:

{

"state":<0|1>,

"gpio_pin":< 0 - 7 >(optional value)

}

Get MQTT connection state

{

"state":<0|1>,

"gpio_pin":< 0 - 7 >(optional value)

}

Method: get_mqtt_state

Parameters: none

Return Value: In case a connection is established already the broker ip or broker.<mica-name>.local is returned, if not "off" is returned

Example:

get MQTT state

```
>>> { "method" : "get_mqtt_state", "params" : [] }
<<< { "result":" mqtt-mica-n7v6o.local" }
```

Enable MQTT Communication with Broker

Method: enable_mqtt

Parameters: broker_ip (ip or mdns name of MQTT broker to connect to as string value)

Return Value:

"Success" in case of success, error message including error code in case of error.

Example: enable MQTT connection to broker with name "mqtt-mica-n7v6o.local”

```
>>> { "method":"enable_mqtt", "params":[" mqtt-mica-n7v6o.local"] }
<<< { "result":"Success" }
```

Disable MQTT Communication with Broker

Method: disable_mqtt

Parameters: none

Return Value: "Success" in case of success, error message including error code in case of error

Example: disable MQTT connection

```
>>> { "method" : "disable_mqtt", "params" : [] }
<<< { "result":"Success" }
```

Register MQTT event, if pin is configured as output, an event is subscribed for this output pin. If configured as input, registering this pin leads to MQTT event publishing of input state information every 200 ms.

Method: subscribe

Parameters: gpio_pin_number (integer value between 0 and 7 (pin 1 to pin 8)), topic (string value of registered MQTT event topic)

Return Value: "Success" in case of success, error message including error code in case of error

Example: register event with topic "pin1-topic" for pin 1

```
>>> { "method" : "subscribe", "params" : [ 0, "pin1-topic" ] }
<<< { "result":"Success" }
```

Delete MQTT event from pin

Method: delete_event

Parameters: gpio_pin_number (integer value between 0 and 7 (pin 1 to pin 8))

Return Value:

"Success" in case of success, error message including error code in case of error

Example: delete MQTT event from pin 1

```
>>> { "method" : "delete_event", "params" : [ 0 ] }
<<< { "result":"Success" }
```

The following error codes can turn up:

SSO_CLIENT_NOT_CONNECTED = -32099,
NOT_AUTHORIZED = -32098,
SPI_CONNECTION_ERROR = -32097,
TOKEN_NOT_VALID = -32096,
LOGGED_IN_ALREADY = -32095
SET_GPIO_INPUT_STATE = -32094,
NON_EXISTING_PIN = -32093,
NON_EXISTING_DIRECTION = -32092,
NON_EXISTING_STATE = -32091,
COULD_NOT_CONNECT_TO_BROKER = -32090,
MQTT_RUNNING_ALREADY = -32089,
MQTT_NOT_RUNNING = -32088,
MQTT_PIN_WITH_EVENT_ALREADY = -32087,
MQTT_PIN_WITHOUT_EVENT = -32086