

# FactoryRally-Receiver

version 1.0

**Julian Kolesik**

April 04, 2021



# Contents

<b>Welcome to FactoryRally-Sender's documentation!</b>	<b>1</b>
Hardware_Main module	1
GUI package	1
Submodules	1
GUI.GameGUI module	1
GUI.NetworkUI module	2
Module contents	2
MQTT package	2
Submodules	2
Module contents	2
MQTT.MQTTPublisher module	2
NetworkUtilities package	3
Submodules	3
NetworkUtilities.NetworkUtility module	3
Module contents	3
REST package	3
Submodules	3
REST.ConnectionHandler module	3
REST.RESTClient module	4
REST.ResourceHandler module	5
REST.Resources module	6
Module contents	7
<b>Indices and tables</b>	<b>7</b>
<b>Index</b>	<b>9</b>
<b>Python Module Index</b>	<b>13</b>



# Welcome to FactoryRally-Sender's documentation!

## Hardware\_Main module

`class Hardware_Main.HardwareMain`

Bases: `object`

This class initiates the REST Receiver and the MQTT Sender and starts the game process.

`setup_connection_handler ()`

This method initializes the connection handler.

`setup_resource_handler ()`

This method initializes the resource handler.

`Hardware_Main.reset ()`

This method performs a reset upon game end.

## GUI package

### Submodules

### GUI.GameGUI module

`class GUI.GameGUI.GameGUI`

Bases: `tkinter.Tk`

This class provides a simple interface for the user to choose a game.

`show_frame (cont)`

This method is used to switch between frames.

**Parameters:** `cont` – the name of the frame which should be switched to

`class GUI.GameGUI.GameSelector (parent, controller)`

Bases: `tkinter.Frame`

This class provides a simple interface for the user to choose a game.

`return_game ()`

This method retrieves the selected game when the user presses the button.

`set_games (games)`

This method sets a given list.

**Parameters:** `games` – the currently active games

`class GUI.GameGUI.GameStartPage (parent, controller)`

Bases: `tkinter.Frame`

This class is used to tell it should generate a new Publisher as a new game started.

`ACTIVE = False`

`button_click ()`

This method sets the parameter active to true if a game is running according to user input.

`get_state ()`

This method returns the current state of the button. If it is pressed, it resets it.

## MQTT package

**Returns:** whether or not the button is pressed

`class GUI.GameGUI.InformationDisplay (parent, controller)`

Bases: `tkinter.Frame`

This class is used to display the current game event on the Raspberry Pi.

`update_information (msg)`

This method displays the current game event.

**Param:** msg: the current game event

## GUI.NetworkUI module

### Module contents

## MQTT package

### Submodules

### Module contents

## MQTT.MQTTPublisher module

`class MQTT.MQTTPublisher.MQTTPublisher (gui, connection_handler, resource_handler)`

Bases: `object`

This class is the MQTT Sender which pushes the current game event to the broker with the according topic.

`ACTIVE = False`

`GAME_STOP = False`

`GEN_TOPIC = 'general'`

`SETUP = False`

`broker = 'broker.emqx.io'`

`client_id = 'python-mqtt-5949'`

`close_game ()`

This method unsubscribes from all current clients/topics when a game ended.

`connect_mqtt ()`

This method creates a connection to the MQTT Broker.

**Returns:** a client instance

`discover_and_notify ()`

This method handles the discovery of all clients in the current game and performs a mapping so each robots has its virtual id.

`generate_game ()`

This function generates REST Receivers for the given game. :return: a REST Receiver Instance

`ids = []`

## NetworkUtilities package

### `perform_game_start ()`

This method performs a fresh start for a user selected game. It is also called when a game is finished and a new game was chosen.

`port = 1883`

### `publish ()`

This method publishes the current message to the broker with the according topic if a game is currently active.

### `start ()`

This method starts the discover process, after that starts the client loop and publishing.

`topics = []`

`MQTT.MQTTPublisher.evaluate_relevance (msg)`

This function evaluates whether the message gets transported or not aka if a real robot can perform it.

**Parameters:** `msg` – the received message to check

**Returns:** relevant or not

## NetworkUtilities package

### Submodules

### *NetworkUtilities.NetworkUtility module*

`NetworkUtilities.NetworkUtility.connect_to_wlan (ssid, password)`

This function connects to a network.

**Parameters:**

- `ssid` – the SSID of the network

- `password` – the WPA passphrase of the given network

**Returns:** result of the command, e.g. if connection was successful

`NetworkUtilities.NetworkUtility.evaluate_result (result)`

This function evaluates whether the connection to a network was a success or failure.

**Parameters:** `result` – the output of the

**Returns:** a list of success and msg

`NetworkUtilities.NetworkUtility.return_all_wifi_connections ()`

This function returns all currently available wifi access points.

**Returns:** list with all wifi SSIDs

### Module contents

## REST package

### Submodules

### *REST.ConnectionHandler module*

`class REST.ConnectionHandler.ConnectionHandler (api)`

Bases: `object`

This class handles the connection to the API and manages different possible errors as well as it performs checks.

```
api_root_url = 'http://localhost:5050/'
```

```
game_started = 'PLAYING'
```

```
wait_for_api_availability ()
```

This function waits until the API is reachable.

```
wait_for_initialized_game ()
```

This function waits until at least one game is initialized.

```
wait_for_running_game (game_id, resource_handler)
```

This function waits until a game is started.

**Parameters:**

- **game\_id** – the given game
- **resource\_handler** – the resource handler

## ***REST.RESTClient module***

```
class REST.RESTClient.RestReceiver (res, conn, game_id)
```

Bases: **object**

This class provides the interface to access the REST API and process the retrieved information.

```
EXECUTION_PHASE = False
```

```
PLAYING_STATE = 'PLAYING'
```

```
check_if_all_player_have_entity ()
```

This method checks if all given players have their entity assigned.

**Returns:** whether or not all players have an entity

```
evaluate_correct_topic (msg)
```

This method returns the corresponding topic of the given message.

**Returns:** the controlling player of the given entity

```
event_types = ['movement', 'upgrade purchase', 'fill register', 'activate upgrade', 'lazer shot', 'game start', 'clear shop', 'fill shop', 'register clear', 'programming timer start', 'programming timer stop', 'random card distribution', 'take card event', 'activate checkpoint', 'game_phase_changed', 'game_round_phase_changed', 'pause', 'unpause', 'damage', 'lazer hit', 'push', 'join', 'lock in', 'robot_start_executing', 'heal', 'energy gain']
```

```
generate_entity_mapping ()
```

This method generates a dictionary containing the player id and the corresponding controlled entity.

**Returns:** dict of entity mapping

```
get_controlled_entities ()
```

This method returns the controlled entity dict.

**Returns:** dict which contains the mapping of entity to player

```
get_current_message ()
```

This method returns the latest (pop) game event message (JSON).

**Returns:** current message

```
topic = 'general'
```

```
REST.RESTClient.check_if_event_is_action (msg)
```

This function checks if the msg contains an entityID.



**Parameters:** **msg** – current message

**Returns:** returns whether or not the event is an action with entity

## ***REST.ResourceHandler module***

`class REST.ResourceHandler.ResourceHandler (api)`

Bases: **object**

This class provides functions for interaction with the API resources.

**add\_resources ()**

This function connects the API with the resources.

**check\_for\_lobby\_game (games)**

This method waits until there are games in LOBBY state.

**Parameters:** **games** – all game ids that are currently active

**create\_consumer (game\_id)**

This method registers an consumer for hardware interaction.

**Parameters:** **game\_id** – the given game identifier

**Returns:** Response from the server e.g. the pat and the id

**get\_all\_robots (game\_id, user\_token)**

This method returns all robot ids.

**Parameters:**

- **game\_id** – the given game identifier
- **user\_token** – the given consumer access token

**Returns:** all robot ids

**get\_controlled\_entities (game\_id, player\_id, user\_token)**

This method returns the controlled entities of the given player.

**Parameters:**

- **user\_token** – the given consumer access token
- **game\_id** – the given game identifier
- **player\_id** – the given player id

**Returns:** the id of the controlled robot from the given player

**get\_event\_head (game\_id, user\_token)**

This method returns the event head message from the API endpoint.

**Parameters:**

- **game\_id** – the given game identifier
- **user\_token** – the given consumer access token

**Returns:** the latest event

**get\_game\_state (game\_id)**

This method returns the current state of the given game.

**Parameters:** **game\_id** – the given game identifier

**Returns:** state of the game

**get\_games ()**

This function returns all currently active games.

**Returns:** the game ids of all active games

**get\_player** (game\_id, player\_id)

This function returns information about the given player in the given game.

**Parameters:**

- **player\_id** – the given player id
- **game\_id** – the given game identifier

**Returns:** information about the given player

**get\_players** (game\_id, user\_token)

This function returns all active players in the given (game\_id) game.

**Parameters:**

- **user\_token** – the given consumer access token
- **game\_id** – the given game identifier

**Returns:** the player ids of all active players

## ***REST.Resources module***

```
class REST.Resources.ConsumersResource (*args, **kwargs)
```

Bases: `simple_rest_client.resource.Resource`

This class represents all resources which represent the consumer endpoint.

```
actions = {'create_consumer': {'method': 'POST', 'url': 'games/{}/consumers'}}
```

```
class REST.Resources.EventsResource (*args, **kwargs)
```

Bases: `simple_rest_client.resource.Resource`

This class represents all resources which represent the event handling endpoint.

```
actions = {'get_event_head': {'method': 'GET', 'url': 'games/{}/events/head?pat={}'}}
```

```
class REST.Resources.GamesResource (*args, **kwargs)
```

Bases: `simple_rest_client.resource.Resource`

This class represents all resources which represent the games endpoint.

```
actions = {'get_game_actions': {'method': 'GET', 'url': 'games/{}/actions'}, 'get_game_status': {'method': 'GET', 'url': 'games/{}/status'}, 'get_games': {'method': 'GET', 'url': 'games'}}
```

```
class REST.Resources.MapResource (*args, **kwargs)
```

Bases: `simple_rest_client.resource.Resource`

This class represents all resources which represent the maps endpoint.

```
actions = {'get_map': {'method': 'GET', 'url': 'games/{}/map'}}
```

```
class REST.Resources.PlayersResource (*args, **kwargs)
```

Bases: `simple_rest_client.resource.Resource`

This class represents all resources which represent the players endpoint.

```
actions = {'get_player': {'method': 'GET', 'url': 'games/{}/players/{}/pat={}'}, 'get_players': {'method': 'GET', 'url': 'games/{}/players?pat={}'}}
```

```
class REST.Resources.RobotsResource (*args, **kwargs)
```

Bases: `simple_rest_client.resource.Resource`

This class represents all resources which represent the robots endpoint.

```
actions = {'get_all_robots': {'method': 'GET', 'url': 'games/{}/entities/robots?pat={}'}, 'get_robot_info': {'method': 'GET', 'url': 'games/{}/entities/robots/{}/info'}, 'get_upgrades': {'method': 'GET', 'url': 'games/{}/entities/robots/{}/upgrades'}}
```

## *Module contents*

# Indices and tables

- `genindex`
- `modindex`
- `search`



# Index

## A

actions (REST.Resources.ConsumersResource attribute)  
(REST.Resources.EventsResource attribute)  
(REST.Resources.GamesResource attribute)  
(REST.Resources.MapResource attribute)  
(REST.Resources.PlayersResource attribute)  
(REST.Resources.RobotsResource attribute)  
ACTIVE (GUI.GameGUI.GameStartPage attribute)  
(MQTT.MQTTPublisher.MQTTPublisher attribute)  
add\_resources()  
(REST.ResourceHandler.ResourceHandler method)  
api\_root\_url  
(REST.ConnectionHandler.ConnectionHandler attribute)

## B

broker (MQTT.MQTTPublisher.MQTTPublisher attribute)  
button\_click() (GUI.GameGUI.GameStartPage method)

## C

check\_for\_lobby\_game()  
(REST.ResourceHandler.ResourceHandler method)  
check\_if\_all\_player\_have\_entity()  
(REST.RESTClient.RestReceiver method)  
check\_if\_event\_is\_action() (in module REST.RESTClient)  
client\_id (MQTT.MQTTPublisher.MQTTPublisher attribute)  
close\_game() (MQTT.MQTTPublisher.MQTTPublisher method)  
connect\_mqtt()  
(MQTT.MQTTPublisher.MQTTPublisher method)  
connect\_to\_wlan() (in module NetworkUtilities.NetworkUtility)  
ConnectionHandler (class in REST.ConnectionHandler)  
ConsumersResource (class in REST.Resources)  
create\_consumer()  
(REST.ResourceHandler.ResourceHandler method)

## D

discover\_and\_notify()  
(MQTT.MQTTPublisher.MQTTPublisher method)

## E

evaluate\_correct\_topic()  
(REST.RESTClient.RestReceiver method)  
evaluate\_relevance() (in module MQTT.MQTTPublisher)  
evaluate\_result() (in module NetworkUtilities.NetworkUtility)  
event\_types (REST.RESTClient.RestReceiver attribute)  
EventsResource (class in REST.Resources)  
EXECUTION\_PHASE  
(REST.RESTClient.RestReceiver attribute)

## G

game\_started  
(REST.ConnectionHandler.ConnectionHandler attribute)  
GAME\_STOP (MQTT.MQTTPublisher.MQTTPublisher attribute)  
GameGUI (class in GUI.GameGUI)  
GameSelector (class in GUI.GameGUI)  
GamesResource (class in REST.Resources)  
GameStartPage (class in GUI.GameGUI)  
GEN\_TOPIC (MQTT.MQTTPublisher.MQTTPublisher attribute)  
generate\_entity\_mapping()  
(REST.RESTClient.RestReceiver method)  
generate\_game()  
(MQTT.MQTTPublisher.MQTTPublisher method)  
get\_all\_robots()  
(REST.ResourceHandler.ResourceHandler method)  
get\_controlled\_entities()  
(REST.ResourceHandler.ResourceHandler method)  
(REST.RESTClient.RestReceiver method)  
get\_current\_message()  
(REST.RESTClient.RestReceiver method)  
get\_event\_head()  
(REST.ResourceHandler.ResourceHandler method)  
get\_game\_state()  
(REST.ResourceHandler.ResourceHandler method)  
get\_games()  
(REST.ResourceHandler.ResourceHandler method)  
get\_player()  
(REST.ResourceHandler.ResourceHandler method)  
get\_players()  
(REST.ResourceHandler.ResourceHandler method)  
get\_state() (GUI.GameGUI.GameStartPage method)  
**GUI**  
module

**GUI.GameGUI**  
module

## H

**Hardware\_Main**  
module

HardwareMain (class in Hardware\_Main)

## I

ids (MQTT.MQTTPublisher.MQTTPublisher attribute)

InformationDisplay (class in GUI.GameGUI)

## M

MapResource (class in REST.Resources)

**module**

GUI

GUI.GameGUI

Hardware\_Main

MQTT

MQTT.MQTTPublisher

NetworkUtilities

NetworkUtilities.NetworkUtility

REST

REST.ConnectionHandler

REST.ResourceHandler

REST.Resources

REST.RESTClient

**MQTT**

module

**MQTT.MQTTPublisher**

module

MQTTPublisher (class in MQTT.MQTTPublisher)

## N

**NetworkUtilities**

module

**NetworkUtilities.NetworkUtility**

module

## P

perform\_game\_start()  
(MQTT.MQTTPublisher.MQTTPublisher method)

PlayersResource (class in REST.Resources)

PLAYING\_STATE (REST.RESTClient.RestReceiver attribute)

port (MQTT.MQTTPublisher.MQTTPublisher attribute)

publish() (MQTT.MQTTPublisher.MQTTPublisher method)

## R

reset() (in module Hardware\_Main)

ResourceHandler (class in REST.ResourceHandler)

**REST**

module

**REST.ConnectionHandler**

module

**REST.ResourceHandler**

module

**REST.Resources**

module

**REST.RESTClient**

module

RestReceiver (class in REST.RESTClient)

return\_all\_wifi\_connections() (in module NetworkUtilities.NetworkUtility)

return\_game() (GUI.GameGUI.GameSelector method)

RobotsResource (class in REST.Resources)

## S

set\_games() (GUI.GameGUI.GameSelector method)

SETUP (MQTT.MQTTPublisher.MQTTPublisher attribute)

setup\_connection\_handler()  
(Hardware\_Main.HardwareMain method)

setup\_resource\_handler()  
(Hardware\_Main.HardwareMain method)

show\_frame() (GUI.GameGUI.GameGUI method)

start() (MQTT.MQTTPublisher.MQTTPublisher method)

## T

topic (REST.RESTClient.RestReceiver attribute)

topics (MQTT.MQTTPublisher.MQTTPublisher attribute)

## U

update\_information()  
(GUI.GameGUI.InformationDisplay method)

## W

wait\_for\_api\_availability()  
(REST.ConnectionHandler.ConnectionHandler method)

wait\_for\_initialized\_game()  
(REST.ConnectionHandler.ConnectionHandler method)

`wait_for_running_game()`  
(`REST.ConnectionHandler.ConnectionHandler` method)





# Python Module Index

## ***g***

GUI

GUI.GameGUI

## ***h***

Hardware\_Main

## ***m***

MQTT

MQTT.MQTTPublisher

## ***n***

NetworkUtilities

NetworkUtilities.NetworkUtility

## ***r***

REST

REST.ConnectionHandler

REST.ResourceHandler

REST.Resources

REST.RESTClient