Chapter 4

API Interactive Map Module and RISE Drone System

Chapter Contents

The Interactive Map Module (IMM) is a module for drone control within the RISE Drone System. The module can be connected directly to the Get Image Application (GIA) running on the drone or to the ground based Swarm Control Service (SCS). The module hosts a database and a web-server that offers the user a web-based GIU where the map/picture data is presented and the user generates new POIs (views) by navigating the map. From the IMM point of view the communication is exactly the same to the SCS and the GIA, so from here on the GIA and SCS are abbreviated as RISE Drone System (RDS).

Note:	Future development: Multisessions can possibly be handled by publishing pois with a session ID as topic.

4.1 Communication

The communication is carried as JSON-messages via ZeroMQ. Socket description TBD.

- The IMM Publishes pois (views) via a Publish socket, RDS Subscribes. This link is named poi-link.
- The IMM can both pull and push information to/from the via an Request-Reply socket information from to the RDS. This link is named info-link where IMM sends requests (client) and RDS replies(server).
- The RDS Publishes georeferenced pictures via a Publish socket. IMM Subscribes. This link is named pic-link.

The IP number setup is as follows TBD.

Listing 4.1: IP numbers

```
{
  "IMM": "xxx.xxx.xxx.xxx",
  "RDS": "yyy.yyy.yyy.yyy"
}
```

The ports used are described in Settings.json. The programs should read their settings from this file.

Listing 4.2: Settings.json-file

```
{
  "IMMPubSocket": "tcp://*:5570",
  "IMMSubSocket": "tcp://localhost:5571",
  "IMMReqSocket": "tcp://localhost:5572",
  "RDSPubSocket": "tcp://*:5571",
  "RDSSubSocket": "tcp://localhost:5570",
  "RDSRepSocket": "tcp://*:5572"
}
```

General function call and ack/nack functions, function argument can be a JSON object, string or number:

Listing 4.3: JSON object function call

```
{
"fcn": "the_name_of_the_function"
"arg": {
        "arg1": 0,
        "arg2": "string_argument_example"
        }
}
```

General response, arg2 is optional and depends on the specific function call.

Listing 4.4: JSON object function call response

```
{
```

```
  "fcn": "ack",
  "arg": "the_name_of_the_function",
  ["arg2": JSON-object with information]
}

{
  "fcn": "nack",
  "arg": "the_name_of_the_function",
  "arg2": "Some text describing the issue"
}
```

4.2 POI-link

The poi-link is set up as Publish Subscribe type. As soon as a GUI client has connected to the IMM, been granted a GUI client ID, session ID and the area of interest is set, the IMM starts publishing pois based on the current view in the GUI(s).

Note:	API between GUI client and IMM is not covered in this document

4.2.1 Fcn: add_poi

The function *add_poi* publishes the corner coordinates and the center coordinates of the current view displayed in the GUI. Views that shall be queued are assigned a unique force que id higher than 0, views that shall not be forced are assigned force_que_id 0. Since several GUI clients can be supported the unique *client_id* is included too.

Listing 4.5: Function call: *add_poi*

```
{
"fcn": "add_poi",
"arg":
  {
    "client_id": 1,
    "force_que_id": 0,
    "coordinates":
    {
      "up_left":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "up_right"
      {
        "lat": 58.123456,
        "long": 16.123456
      },
```

```
      "down_left":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "down_right":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "center":
      {
        "lat": 58.123456,
        "long": 16.123456
      }
    }
  }
}
```

4.3 PIC-link

The pic-link is set up as Publish Subscribe type. As soon as a picture is taken by a drone the georeferenced picture will be published by RDS.

4.3.1 Fcn: new_pic

The function *new_pic* publishes georeferenced pictures. Argument holds drone_id as a string, type as a string [”rgb”, ”IR”], force_que_id as an integer and the coordinates in decimal degrees in a separate JSON-object. TBD how is the picture attached to the message?

Listing 4.6: Function call: *new_pic*

```

{
"fcn": "new_pic",
"arg":
  {
    "drone_id": "one",
    "type": "rgb",
    "force_que_id": 0,
    "coordinates":
    {
      "up_left":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "up_right"
```

```
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "down_left":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "down_right":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "center":
      {
        "lat": 58.123456,
        "long": 16.123456
      }
    }
  }
}
```

4.4 INFO-link

The info-link is set up as a Request Reply type. Through this link the IMM can both push and pull information form the RDS. The available commands are listed in this section.

4.4.1 Fcn: set_area

The function *set_area* pushes the defined area of interest (boundaries). The area must be set before RDS will follow any instructionsSince several GUI clients can be supported the unique *client_id* is included too. Polygon is defined by a number of waypoints wp_0, wp_1, ..wp_n and the interpretation is the area created by the lines wp_0-wp_1, wp_1-wp_2, ..wp_n-wp_0.

Note:	Wp lines defined by the consecutively numbered waypoints may never cross

Note:	Python dictionaries does not have a given sort order

Listing 4.7: Function call: *set_area*

```
{
"fcn": "set_area",
"arg":
  {
```

```
    "client_id": 1,
    "coordinates":
    {
      "wp0":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "wp1"
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "wp2":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "wp3":
      {
        "lat": 58.123456,
        "long": 16.123456
      }
    }
  }
}
```

4.4.2 Fcn: get_info

The function *get_info* requests information from the RDS. The requested info type is tagged as the function argument. Available arguments are [drone-info].

Listing 4.8: Function call: *get_info*

```
{
  "fcn": "get_info",
  "arg": "drone-info"
}
```

The reply holds a list with the connected drones and their time to bingo (remaining time to aborting mission due to fuel), drone-id as a string and time2bingo as an integer [minutes]:

Listing 4.9: Function call response:

```
{
"fcn": "ack",
"arg": "get_info",
"arg2":
  {
  "drone-id": "one",
```

```
  "time2bingo": 15
  }
}
```

4.4.3 Fcn: set_mode

The function set_mode sets the mode to AUTO or MAN. If mode is set to AUTO the coordinates of the current view are sent as zoom reference. If mode is set to MAN the zoom can be omitted. In AUTO the map navigation does not affect the drones and allows the user to navigate the map while the drones collects pictures from the area of interest at the current zoom level.

Listing 4.10: Function call: **quit**

```
{
  "fcn": "set_mode",
  "arg":
    {
    "mode": "AUTO",
    "zoom":
    {
      "up_left":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "up_right"
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "down_left":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "down_right":
      {
        "lat": 58.123456,
        "long": 16.123456
      },
      "center":
      {
        "lat": 58.123456,
        "long": 16.123456
      }
  }
}
```

4.4.4 Fcn: clear_que

The function $clear_que$ clears all view in the current que.

Listing 4.11: Function call: **clear_que**

```
{
  "fcn": "clear_que",
  "arg": ""
}
```

4.4.5 Fcn: que_ETA

The function que_ETA requests the ETA for next que item.

Listing 4.12: Function call: **que_ETA**

```
{
  "fcn": "que_ETA",
  "arg": ""
}
```

The response holds the number of seconds estimated to the next queued item to be handled.

Listing 4.13: Function call response:

```
{
"fcn": "ack",
"arg": "que_ETA",
"arg2": 30
}
```

4.4.6 Fcn: quit

The function quit informs the RDS that the last GUI client disconnected and that the mission can be aborted. Drone will fly home and land.

Listing 4.14: Function call: **quit**

```
{
  "fcn": "quit",
  "arg": ""
}
```