Rest API

This is a reference for the mHubX rest API. It describes the most important import/send functions so that you can implement interfaces to e.g. ERP, CRM/Webshop systems and to devices that record data:

You can import

- Production Orders
- Items

You can send

- Events
- Measurements

The rest endpoints each get its own heading. The description contains the url with the type of request (POST|GET). If a post is used, the post body is described: It's the message send. The message format is always json.

Url & Message Format

The URL format for endpoints corresponds to the standard REST-API format for calling actions in Juwi. It follows the pattern

```
{{url}}/juwi/action?page={page}&name={action}
```

The url itself is the baselUrl of the mhubx-cc installation plus the postfix module, e.g.

```
http://localhost:8080/mhubx-cc/module
```

Parameters will be attached afterwards, page and name are reserved id's

```
{{url}}/juwi/action?page={page}&name={action}&id_1=...&id_n=...
```

The URL format might change since a beautifier is planned in the future, but the messages stay the same: If you want to be compatible with future versions, you might want to make the url configurable in a way that let you use templates for your urls: If your client program displays billboards of kpis per machine you could indicate your urls with placeholders that correspond to the ids

Static link example, hardcoded ids display page with certain search content inline, e.g. in an iFrame. Note that this is actually not an API call, but a direct call to display a page

```
[
  {
    "ManagerName": "Workplace",
    "ManagerDeeplinks": [
      {
        "MachineIdentifier": "cps1",
        "Url": "http://localhost:8080/mhubx-
cc/wiki/OperationProgress?search=cps1",
        "Label": "OperationProgress"
```

```
      }
  }
]
```

Now lets define a real API call where the client program parses the returned json message to display an entry in a billboard over the machine visualization

```
[
  {
    "ManagerName": "Workplace",
    "ManagerKPi": [
      {
        "MachineIdentifier": "cps1",
        "KpiIdentifier":"oee",
        "Url": "http://localhost:8080/mhubx-
cc/module/juwi/action?page=Logic.Interface&name=getMeasurement&msm_id=oee&s
ource=system&system_id=cps1",
      }
  }
]
```

Note that by default if no start_dt and end_dt is indicated as parameter, the oee of today will be returned. For kpis that are range based, the return value daterange indicates this.

The following returned message would then be parsed by the client program to create the billboard.

```
"systems": {
    "cps1": {
        "errors": [],
        "oee": {
                    "model_id": "oee",
                    "instance_id": "cps1",
                    "current": 54.39263571328892,
                    "avail": 0.99,
                    "performance": 0.7325607503473254,
                    "yield": 0.75,
                    "oee": 0.5439263571328892,
                    "config": {
                        "label": "OEE",
                        "unit": "%",
                        "fradi": 1,
                        "minmax_range": [
                            0,
                            100
                        ],
                        "target_range": [
                            80,
                            100
                        ]
                    },
                    "daterange": [
                        "2020-10-13 00:00:00.000",
                        "2020-10-13 20:29:43.987"
                    ]
                }
        }
}
```

The billboard could now build its entry with the following values

```
var label = msg.config.label;
var unit  = msg.config.unit;
var value = msg.current;
```

Authentication

The mHubX v1 API uses basic http authentication. Username and password are transferred in the http authorization header. Every http API will let you set this header. For juwi you can for example set it with the following code:

```
var client = jrt.restClient();
client.setBasicAuth("system", "changeit");
client.setUrl("http://localhost:8080/mhubx-cc");
var call =
client.prepare("/module/juwi/action?page=Rest.ProdOrder&name=doImport",
"POST");
call.setBody(data);
var result = client.exec(call);
```

In a production environment you should always use https encryption, since the password itself is transfered unencrypted. Therefore you need the tls/ssl layer of https to protect it.

Postman Testing

The REST-API comes with a postman api doc. It can be downloaded here

http://download.nuveon.de/doc/mHubX/book/html5/docs/mHubX/api/mHubX%20API%20v1.postman_collection.json

Download the file to your local hard drive and then import it with the import menu of Postman. An imported file will look like this:

Don’t forget to create an environment that will set the following placeholders

- {{url}} - baseUrl of your mHubx installation + "/module" postfix
- {{username}} - username for basic auth
- {{password}} - password for basic auth
- {{system}} - the system_id you want to test
- {{main_module}} - the main module_id within the system you want to test

The current_timestamp is generated by pre_request scripts if needed, so you don’t have to set it. Here’s an example environment:

Import Prod Orders

With prod order import you can import production orders in bulk. This is used in the demo webshop to place orders into mHubX.

POST

```
{{url}}/juwi/action?page=Logic.ProdOrder&name=doImport
```

BODY

```
{
    "prod_orders" : [
        {
            "prod_order_id" : "test4",
            "item_id" : "_void",
            "target_qty" : 3,
            "target_date" : "2018-10-05T10:30:51.336+02:00",
            "target_dur" : 3000,
            "json_data" : {},
            "comments" : "REST Import",
            "pos" : [
                {
                    "pos_id" : 0,
                    "seq" : 0,
                    "dsc" : "",
                    "operation_type_id" : null,
                    "target_date" : "2018-10-08T13:12:50.332+02:00",
                    "target_setup_dur" : 0,
                    "target_prod_dur" : 1000*60,
                    "target_qty_produced" : 3,
                    "target_qty_scrap" : 0,
                    "target_qty_giveaway" : 0,
                    "json_data" : {},
                    "comments" : ""
                }
            ]
        }
    ]
}
```

Note that target_prod_dur is dur per piece (te) while target setup dur is setup dur for the whole operation.

Create Device

POST

```
{{url}}/juwi/action?page=Logic.Controller&name=doImport
```

BODY

```
{
    "type_id": "acme123",
    "serial_number": "545016",

    "sensors": [

            { "tag": "ACCx", "type":"Acceleration", "unit":"g-force" },
            { "tag": "ACCy", "type":"Acceleration", "unit":"g-force" },
            { "tag": "ACCz", "type":"Acceleration", "unit":"g-force" },

            { "tag": "L1", "type":"Light", "unit":"lux" },

            { "tag": "T1", "type":"Temperature", "unit":"C" },
            { "tag": "T2", "type":"Temperature", "unit":"C" },
            { "tag": "T3", "type":"Temperature", "unit":"C" },
            { "tag": "T_RH1", "type":"Temperature", "unit":"C" },
```

```
            { "tag": "T_P1", "type":"Tempeature", "unit":"C" },

            { "tag" : "DMS1", "type":"Extensometer", "unit":"cnts" },
            { "tag" : "DMS2", "type":"Extensometer", "unit":"cnts" },

            { "tag" : "RH1", "type":"Humidity", "unit":"%rH" },

            { "tag" : "P1", "type":"Pressure", "unit":"mbar" },

            { "tag" : "BAT", "type":"Voltage", "unit":"V" }

    ],

    "switches": [],
    "detectors": []
}
```

Send Module Events

Sending events to mHubX is always tied to a source module, you indicate this module as id in the form "system_id;module_id". Then you indicated the type "mHub_ModuleEvent". In an object called "mHub_ModuleEvent" you then indicate name and value of the event.

POST

```
{{url}}/juwi/action?page=Logic.Interface&name=postEvent
```

BODY

```
{
        "source": {
        "id":"cps1;PR" },
    "type": "mHub_ModuleEvent",

        "mHub_ModuleEvent": {
               "name"  : ....
         "value" : ....
          }
}
```

The following event names exist

- PROD_COUNT - indicated a production count (good and bad)
- AUTO_START - indicateds that a module starts production in automatic mode
- AUTO_STOP - indicateds that a module stop production in automatic mode

AUTO_START and AUTO_STOP are important for the speed calculation. A normal production machine would send signals in the following order

- AUTO_START
- PROD_COUNT (1…n)
- AUTO_STOP

AUTO_START, AUTO_STOP

This will indicate the production start of a machine in automatic mode. This will start the speedometer, using the qty_min and qty_factor defined as standard for the module

```
"mHub_ModuleEvent": {
        "name": "AUTO_START",
}
```

The AUTO_STOP Event will stop the speedometer.

```
"mHub_ModuleEvent": {
        "name": "AUTO_START",
}
```

PROD_COUNT produced

The normal prod count indicates that a production has happened at the module. It does not tell you if it was good or bad. If no "scrap signal" comes afterward, it means its good production. Good production in mHub is only counted at the main module.

```
"mHub_ModuleEvent": {
        "name": "PROD_COUNT",
    "value" : {
            "produced" : { "value": 1}
    }
}
```

PROD_COUNT scrap

Scrap signals are send if a module detects a scrap. It needs to send a prod count first

```
"mHub_ModuleEvent": {
        "name": "PROD_COUNT",
    "value" : {
            "scrap" : { "value": 1}
    }
}
```

Alarms

You can send alarms to modules. In the default module interface an alarm is an error code > 0. Therefore to activate an alarm you send an error code > 0.

POST

```
{{url}}/juwi/action?page=Logic.Interface&name=postAlarm
```

BODY

```
{
        "source": {
        "id":"{{system}};{{main_module}}" },
    "type":"module",
    "severity":"error",
    "code": "100",
    "text": "No Material"
}
```

To deactivate the alarm you send the error code 0

BODY

```
{
        "source": {
        "id":"{{system}};{{main_module}}" },
    "type":"module",
    "severity":"error",
    "code": "0",
    "text": ""
}
```

To get all current alarms for a system you can send the following request

GET

```
{{url}}/juwi/action?page=Logic.Interface&name=getAlarms&system_id={{system}
}
```

The result looks like this

```
{
    "system_id": "cps1",
    "alarms": [
        {
            "module_id": "PR",
            "errors": [
                {
                    "code": "100",
                    "msg": "No Material"
                }
            ]
        }
    ]
}
```

Send Measurements

POST

```
{{url}}/juwi/action?page=Logic.Interface&name=postMeasurement
```

BODY

```
{
        "source": { "id":"acme123;134554;T1" },
    "data": {
        "source_ts": "2019-10-23T07:07:32.939Z",
        "value": 83.32
    }
}
```

Get Measurements

You can get measurements by indicating their id. You can get measurements for the current datetime, at a certain datetime or for a daterange. If you need it for the current datetime, simply leave out the dt parameters. dt is indicated in the standard ISO format.

GET

```
{{url}}/juwi/action?page=Logic.Interface&name=getMeasurement
```

PARAMS

```
&msm_id=oee
```

Source: system,operation,controller

```
&source=system
```

Date

```
&dt=2018-10-08T13:12:50.332+02:00
```

Daterange (Interval half open)

```
&from_dt=2018-10-08T00:00:00.000+02:00
&to_dt=2018-10-09T00:00:00.000+02:00
```

RETVAL

```
{
   "value": 83.32
   "unit": "pct"
}
```

System Measurements

mHub module measurements ID's

- avail_state → current availability state (+ color code)
- performance → speed: qty/minute
- yield → good production in %
- scrap → scrap
- good_prod → good production
- oee → displays oee in daterange
- current_op → current operation + statistics

You can get all systems and all measurements or you can filter for specific ones. If you don't indicate system_id or set it to "*", you will get all systems. If you don't indicate msm_id or set it to "*", you will get all measurements. You can indicate more than one system or measurement by separating them with ;

Examples

- system_id=cps1;cps2;
- msm_id=good_prod;scrap;

All measurments for ALL systems

```
{{url}}/juwi/action?page=Logic.Interface
        &name=getMeasurement
        &source=system
        &system_id=*
        &msm_id=*
```

All measurements for system cps1;cps2;

```
{{url}}/juwi/action?page=Logic.Interface
        &name=getMeasurement
        &source=system
        &system_id=cps1;cps2;
```

RETVAL avail_state

```
{
    "systems": {
        "cps1": {
            "errors": [],
            "avail_state": {
                "model_id": "avail_state",
                "instance_id": "cps1",
                "event_id": "SINIT",
                "color_code": "#cdcdcd",
                "signal_color": "white",
                "label": "SINIT: no state yet"
            }
        }
    }
}
```

The signal_color indicates coloring for apps that only support the basic color range, like red,green,blue and yellow or that need some kind of severity indicator. Most severe avail states like idle_unreasoned and idle_fault are colored in red and therefore if read is prioritized in those systems can be displayed first. Yellow comes second: therefore signal_color uses the traffic light metaphor.

RETVAL performance

```
{
    "systems": {
        "cps1": {
            "errors": [],
            "performance": {
                "model_id": "performance",
                "instance_id": "cps1",
                "current": 0.0,
                "theoretical": 81.0,
                "min": 0.0,
                "max": 0.0,
                "trend": 90,
                "config": {
                    "label": "Performance",
                    "unit": "pk/h",
                    "fradi": 0.0,
                    "minmax_range": [
                        0.0,
```

```
                        72.0
                    ],
                    "target_range": [
                        50.0,
                        60.0
                    ]
                }
            }
        }
    }
}
```


RETVAL yield


```
{
    "systems": {
        "cps1": {
            "errors": [],
            "yield": {
                "model_id": "yield_pct",
                "instance_id": "cps1",
                "current": 0.0,
                "config": {
                    "label": "Yield",
                    "unit": "%",
                    "fradi": 1.0,
                    "minmax_range": [
                        0.0,
                        100.0
                    ],
                    "target_range": [
                        80,
                        100
                    ]
                }
            }
        }
    }
}
```


RETVAL scrap


```
{
    "systems": {
        "cps1": {
            "errors": [],
            "scrap": {
                "model_id": "scrap_pk",
                "instance_id": "cps1",
                "current": 0.0,
                "config": {
                    "type": "progress",
                    "color": "red",
                    "label": "Scrap",
                    "unit": "pk",
                    "fradi": 0.0,
                    "target": 99.0
                },
                "progress_pct": 0.0
            }
        }
    }
```

```
}
```


RETVAL good_prod


```
{
    "systems": {
        "cps1": {
            "errors": [],
            "good_prod": {
                "model_id": "good_prod_pk",
                "instance_id": "cps1",
                "current": 0.0,
                "config": {
                    "type": "progress",
                    "label": "Good Prod",
                    "unit": "pk",
                    "fradi": 0.0,
                    "target": 99.0
                },
                "progress_pct": 0.0
            }
        }
    }
}
```


RETVAL oee


```
{
    "systems": {
        "cps1": {
            "errors": [],
            "oee": {
                "model_id": "oee",
                "instance_id": "cps1",
                "current": 0.0,
                "avail": 1.0,
                "performance": 1.0,
                "yield": 0.0,
                "oee": 0.0,
                "config": {
                    "label": "OEE",
                    "unit": "%",
                    "fradi": 1.0,
                    "minmax_range": [
                        0.0,
                        100.0
                    ],
                    "target_range": [
                        80.0,
                        100.0
                    ]
                },
                "daterange": [
                    "2020-11-23 00:00:00.000",
                    "2020-11-23 08:25:36.875"
                ]
            }
        }
    }
}
```

RETVAL current_op

Current Operation ON


```
{
    "systems": {
        "cps1": {
            "errors": [],
            "current_op": {
                            "success": true,
                            "operation": {
                                "system_id": "cps1",
                                "prod_order_id": "po_7001_1",
                                "pos_id": 1,
                                "target_prod_dur": 60000,
                                "target_setup_dur": 120000,
                                "target_prod_dur_total": 600000.0,
                                "target_dur": 720000.0,
                                "target_start_dt": "2020-09-16
07:07:00.000",
                                "target_end_dt": "2020-09-16 07:19:00.000",
                                "target_qty_min": 1.0,
                                "good_prod_qty": 5.0,
                                "scrap_qty": 4.0,
                                "open_qty": 5.0,
                                "start_date": "2020-09-16 07:02:43.864",
                                "predicted_dur": 0
                            }
                        }
        }
    }
}
```


No Current Operation: If there is no current operation, success will be true, but there will be no "operation" field in the current operation!


```
{
    "systems": {
        "cps1": {
            "errors": [],
            "current_op": {
                "success": true
            }
        }
    }
}
```


Operation Measurements

- exec_dur → exec dur in millis overall
- qty_produced → all qty produced
- qty_scrap → scrapp of all qty produced
- setup_dur → tr dur
- prod_dur → te dur (dur per qty including idle times & scrap)

Send Log Messages

You can send log messages that can be shown in combination with measurements. For details see the section on log messages in the mqtt api chapter

POST

```
{{url}}/juwi/action?page=Logic.Interface&name=postLogMsg&id=cps1;PR
```

BODY

```
10-21 17:47:26 |WikiSecurityEvent             |INFO |
WikiSecurityEvent.LOGIN_AUTHENTICATED
[source=com.ecyrd.jspwiki.auth.Authenti
cationManager@41bc501b, princpal=com.ecyrd.jspwiki.auth.WikiPrincipal
system, target=com.ecyrd.jspwiki.WikiSession@736dd47f]
```