v.1.1

Clou RFID reader connector web API description

By	Date	Comments
Pavel Fomin	12.Dec.2017	Created v.1a., description of basic commands
Pavel Fomin	21.Dec.2017	- Fixed some logging issues - Fixed bugs in stop command processing - Fixed reliability issue with reader device connection management - UDP beacon info content updated and enriched - Method HEAD support added - Added choice of antenna with start scan command - Added query command to request parameters and settings from reader device
Pavel Fomin	17.Apr.2019	- Significantly updated the API logics to full v.1
Pavel Fomin	12.Dec.2019	- Updated documentation for public use

1. Request to server

1.1. HTTP methods allowed: GET, HEAD, POST.

1.2. API request URL format:

http(s)://<host:port>/api/v1/<rid>/<method>

where:
<host:port> - URL of your service
api - name of the API
v1 - version of the API, now version 1
<rid> - ID of the reader, this is the name of the reader that is configured in the /etc/clou.conf file, check the configuration manual
<method> - API metod name

1.3. API methods
1.3.1. getstatus: get the current status from the connector process
1.3.2. getdatacount: get the count of unique RFID tags in the memry of the process by the moment of request
1.3.3. getdata: get all tag data from the memory of the process
1.3.4. cleandata: clean all tag data in the process memory completely
1.3.5. shutdown: safely shut down the process
1.3.6. update: update the query commands templates reference, check the commands template reference manual
1.3.7. query: send the query command to the reader, check the commands template reference manual

1.4. HTTP requests payload format: always JSON, http://json.org/

2. Response from server

2.1. Preferred to use Nginx web server to work on server side.

2.2. HTTP replies payload format: always JSON, http://json.org/
2.2.1. Server SHOULD supply the response with JSON payload for statuses other than 200 providing information about error reasons.

3. API methods

GET is the default HTTP method, if not specified other.

All methods except **query** if worked successfully reply with **200 OK** and JSON of the structure:

```
{
    "is-ok": true,
    "result": {<result payload>}
}
```

where:

`true` or `false`	-	succesfully worked or not,
`<result payload>`	-	is the JSON with payload, see for each command below

In case of `"is-ok": false` the content of `<result payload>` is a string describing the problem that happened.

Method **query** behaves differently, it demands JSON payload in HTTP request with query information, and replies **200 OK** only if the reader replied to process with some data. If there were some problems during the sending query to reader, getting or processing the reply from reader, or other issues, the HTTP status code will be not 200 OK, but other, and the "Error" JSON will be returned in the HTTP response payload, like described in (4.1.).

3.1. getstatus

Get the current status from the connector process.
Request payload: None
Reply with `<result payload>` which is JSON:

```
{
    "reader-connected": true, # if reader now connected to the process
    "reader-connected-since": "17.04.2019 17:27:45+0300",
    "reader-disconnected-since": null,
    "reader-last-act-time": "17.04.2019 17:49:01+0300",
    "time-since-reader-last-act": 0.40787816047668457,     # seconds
    "process-up-since": "17.04.2019 17:27:34+0300",
    "time-since-clock-check": "17.04.2019 17:42:34+0300",  # NTP clock
check by the process
    "ntp-avg": 0.0014733076095581055,   # average difference between
process server clock and NTP clock
    "ntp-max": 0.0028798580169677734,   # max difference between process
server clock and NTP clock for last 100 checks
    "queue-to-send-len": 0,   # technical
    "queue-sent-len": 0,      # technical
    "decoded-frames-list-dicts-len": 0,# technical
    "fme-CLU-recv-list-len": 0,    # technical
    "fme-STS-recv-list-len": 0,    # technical
    "config": {<config JSON contents loaded by process>},
    "cmd-template-reference-list": [<list of commands in then refer-
ence>]
}
```

3.2. getdatacount

Get the count of unique RFID tags in the memry of the process by the moment of request.
Request payload: None
Reply `<result payload>` = int, number of scanned unique tags stored in the process memory at the moment of request

3.3. getdata

Get all tag data from the memory of the process.
Request payload: None
Reply `<result payload>`: list of JSONs, of the format below

```
{
    "EPC_code": "300ED89F3350005FE270E5B7", # EPC code of the tag
    "ant_id": 1,    # ID of the antenna that catched this tag
    "params": {     # set of optional parameters that reader can send
        "RSSI": 98, # RSSI value
        "TIME": 1167611022.604744,# UNIX timestamp by clock of reader
when tag wass catched
        "SERIES_NUM": "006067A6"  # serial number for confirming the
tag
        # there can be more optional parameters, check the Clou Proto-
col specification
    },
    "decode_error": false,    # if there were any errors decoding the
tag data from the frame received from reader
    "EPC_len": 96,  # bit length of EPC code of tag
    "UMI": 0,       # UMI, check UHF RFID standards
    "XPC_indicator": 0,  # XPC, check UHF RFID standards
    "num_sys_id_toggle": 0,   # check UHF RFID standards
    "RFU": "0x00"   # RFU, check UHF RFID standards
}
```

3.4. cleandata

Clean all tag data in the process memory completely.
Request payload: None
Reply `<result payload>`: string saying how many records in the process memory was erased:
`"Successfully erased __ RFID tag records in tag buffer"`

3.5. shutdown

Safely shut down the process.
Request payload: None
Reply `<result payload>`: string saying that process is going to safely shut down:
`"Successfully shutting down the process"`

3.6. update

Update the query commands templates reference, please check the commands template reference manual.
Request payload: None
Reply `<result payload>`: string saying that process is successfully updated the templates reference of commands from specified number of files, and lists file names from which comannds were loaded:
`"Successfully updated command reference __ files: [ . . . ]"`

3.7. query

Send the query command to the reader, check the commands template reference manual.
Request payload: `<JSON with command to send>`

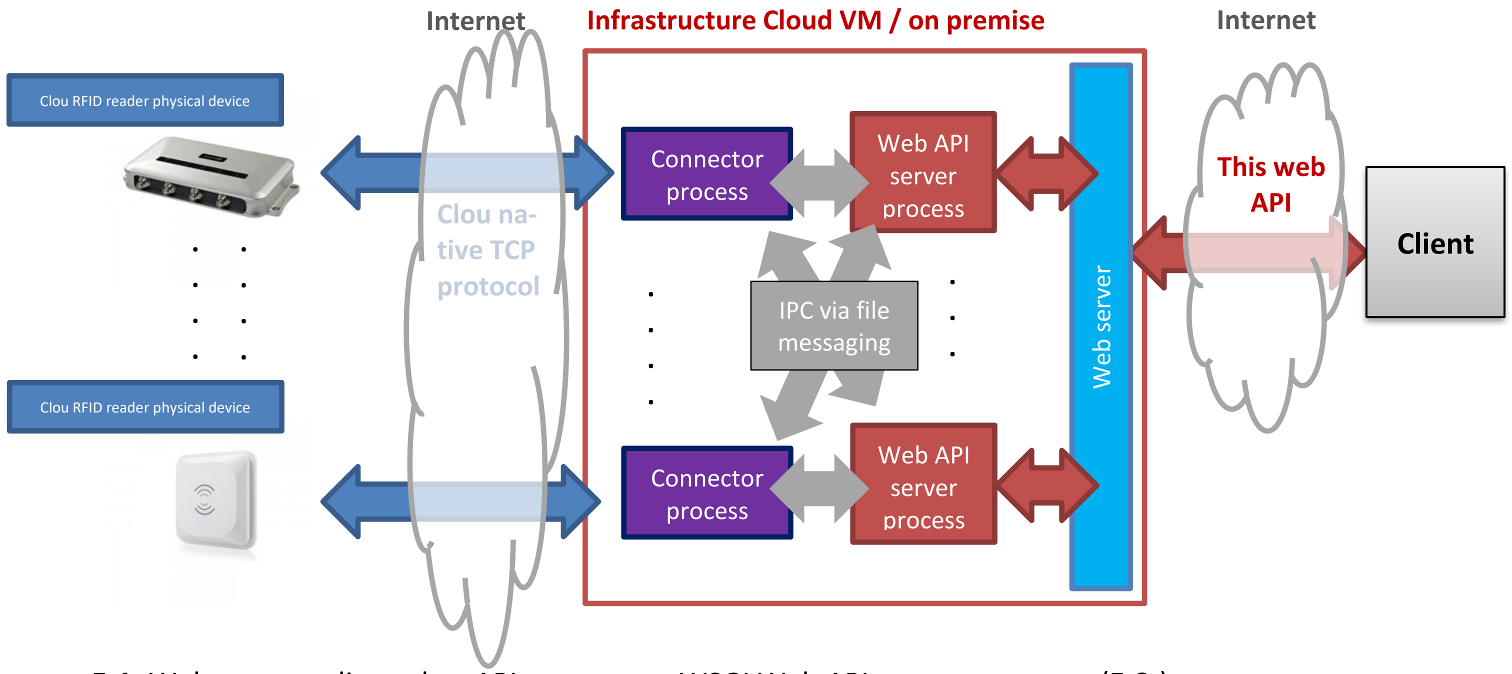
Reply `<result received JSON>`
Command references are stored in JSON files, one file per command, named **<command ID>.json**, in each file is described the request format and the reply format.
When running (**3.6.**) **update** command, all files from the configured command reference folder are read and connector process is trying to decode them; after that new commands are ready to be used. With this process new commands can be added be configuration, without updating the application itself.
In case of new commands configuration requested please create the issue at the GitHub repository.
Please check command templates references ***.json.txt** for understanding the format of JSON used to send commends and JSON to receive when command completed.

4. General requirements

4.1. If the HTTP status in a server reply not equals 200 OK, the HTTP reply can be supplied with JSON payload informing about error reasons.
JSON
Always a JSON object with one or more fields.
The only field that MUST be in JSON supplied (if any) with non-200 HTTP replies is:
"Error":"error text"
Where:
error text – text describing the error
Other fields' values received in non-200 reply payload JSON are to be investigated by the support.
NOTE: anyway, the values of JSON "Error" field or other possible error fields are readable and understandable with common human sense, almost always.

4.2. The server is now in beta-testing operation mode. Please, log & track all the communication with server in order to reproduce and fix possible problems faster by our support.

5. Model



5.1. Web server – dispatches API requests to WSGI Web API server processes (**5.2.**)

5.2. Web API – WSGI applications: **clouweb.py** = WSGI app:

```
def application(environ, start_response):
        ...
        ...
```

- tested with Nginx Unit, but no limitations only complying WSGI

5.3. Connector process: **cloucon.py**, Clou protocol managed by this process;
important: *for each RFID reader there must be 1 and only 1 runnig process*, this is set up with configuration in **clou.conf.**
Example of command to launch the process in detached mode:

```
python37 /usr/share/dev/clouweb/cloucon.py msk_cl7206b2
/usr/share/dev/clouweb/clou.conf +0300 &
```

where:
`msk_cl7206b2` – first command line parameter is the name of RFID reader device, this can be any string ID, corresponding to ID in the **clou.conf**
`/usr/share/dev/clouweb/clou.conf` – second parameter is the path to config
`+0300` – time zone for logging relative to UTC, +0300 is Moscow time; no DST supported

5.4. File messaging model is used for fully async communication between web API porcesses and connector processes: **fme.py**, this is the module developed for file messaging; not documented.