

Inxpect MSK-101-POE Rest API

Version: 1.0.0

Inxpect SPA

Abstract

This document describes the Rest API provided by MSK-101-POE sensor

Contents

1	Overview	3
2	Update process	3
3	APIs	5
3.1	/detectionStatus (GET)	5
3.1.1	Returned value	6
3.2	/alarmStatus (GET)	6
3.2.1	Returned value	6
3.3	/sensorConfiguration (GET and PUT)	8
3.3.1	Configuration object encoding	8
3.4	/sensorPositionStatus (GET)	11
3.4.1	Returned value	11
3.5	/sensorInfo (GET)	11
3.5.1	Returned value	11
3.6	/calibrationStatus (PUT)	12
3.6.1	Input argument	12
3.6.2	Return value	12
3.7	/calibrationData (GET)	13
3.8	/installationConfigurations (GET)	13

3.8.1	Returned value	13
3.9	/netSettings (GET and PUT)	14
3.9.1	Configuration encoding	14
3.10	/changePassword (PUT)	15
3.10.1	Password encoding	15
3.10.2	Return value	15
3.11	/comBoardName (GET and PUT)	15
3.11.1	Name encoding	16
3.11.2	Return value	16
3.12	/login (PUT)	16
3.12.1	Password encoding	16
3.12.2	Return value	16
3.13	/logout (PUT)	16
3.13.1	Return value	16
3.14	/sensorReset (PUT)	17
3.14.1	Return value	17
3.15	/comboardReset (PUT)	17
3.15.1	Return value	17
3.16	/updateToken (GET)	17
3.16.1	Return value	17
3.17	/eventLog (GET)	18
3.17.1	Return value	18
3.18	/startProcessUpdate (PUT)	19
3.18.1	Argument encoding	19
3.19	/startManifestUpdate (PUT)	19
3.19.1	Return value	19
3.20	/chunkManifestUpdate (PUT)	19
3.21	/stopManifestUpdate (PUT)	20
3.21.1	Return value	20
3.22	/startFileUpdate (PUT)	20
3.22.1	Return value	20
3.23	/chunkFileUpdate	20
3.24	/stopFileUpdate	21
3.24.1	Return value	21
3.25	/stopProcessUpdate	21
3.25.1	Argument	21
3.25.2	Return value	21

1 Overview

This document describes the MSK-101-POE Rest API interface. This device has an ethernet outlet that can be used to power the device (POE) and to communicate with it. Communication is implemented by an HTTPS Rest API that allows reading/writing the device configuration and retrieve the device status.

The device communicates over a secure HTTP connection (HTTPS) and the client must install the proper certificate which is available at (<https://www.inxpect.com/security/tools>)

Most of the operation on the device are restricted to authenticated clients. A client needs to login to get authenticated. In order to login a client shall provide the device password and will get a session token that shall be added to any http call as an header in the form: **Authorization: Bearer token** where token is the verbatim token as received on login success.

Since tokens have an expiration date which is unknown the client must be ready to handle the unauthorized error (HTTP 403). If any call fails with such an error the client shall login and repeat the original call.

2 Update process

The MSK-101-POE sensor allows upgrading all its software components OTC (over the cable). The upgrade process is not trivial as there are several components in the system with their own software on board. On top of this the firmware is signed to guarantee system security and this introduces some further complexity. Update packages are available as IFW files which are distributed by Inxpect and cannot be created or manipulated by anyone else. An IFW file is just a zipped file containing the files and metainformation to upgrade one system component. An IFW file *MUST* contain a special file named package.json which describes the whole package. This JSON object has the following structure:

```
{"f": "CBPOE_v1209.bin", "v": "1.209", "df": 2,
  "dst": 1, "vc": 1209, "dm": 1}
```

the “dst” field specifies the target subcomponent for which this was is meant. The system has three different components which can be upgraded:

- MSK-101 core system
- Communication board
- Configuration web interface (web server)

If the destination is either MSK-101 core system or the communication board then the “f” field specifies the file with the device firmware. If the destination is the web server then “f” contains the name of another json file describing the multi file web server update (this file is named contents.json). Since updating the whole server may take some time there is a mechanism to verify what really needs to be updated. The client performing the update shall GET the following resource on the device:

msk101poe-xxxxx.local/contents.json

contents.json looks like this:

```
{"dm": 0, "fA": [{"f": "static/favicon.ico.gz",
                  "d": "B75U5szUcv12LI0xB2qY81m84aH+M4/FD3odmQQ4eLo="},
                ...],
  "df": 0, "v": "1207", "vc": 1207}
```

The file contains an array (“fA”) with the list of files for the web server. Each file is described by its name and by its fingerprint. The contents.json retrieved from the device can be compared to the one stored in the IFW package and only the files whose fingerprint is different shall be sent to the device. The ones whose fingerprint is unchanged can be skipped.

The other fields of the package.json are the following ones:

- v is the version of the component (string value)
- vc is the version of the component (numeric value suitable for comparison)
- dm is the device mode this file is meant for
- df is the device family this file is meant for

For any file that is meant for the device there must be a corresponding file with the same file name and the extra extension “.mf” which is its meta information. For example:

```
CBPOE_v1209.bin
CBPOE_v1209.bin.mf
package.json
```

In this example we have the `package.json` descriptor and one file (`CBPOE_v1209.bin`) with its corresponding meta information. The package can contain any number of files for the device but they must belong to the same component.

Once a client has unpacked and read the IFW file it can start the actual update process which is composed by the following steps:

1. Start an update session for a given component (e.g. core sensor firmware)
2. Start update file meta information
3. Send meta information chunks (one by one)
4. Stop update meta information
5. Start sending file data
6. Send data chunks (one by one)
7. Stop sending data
8. Stop update session

Steps 2-7 must be repeated for each file to send.

3 APIs

The APIs provided by MSK-101-POE are listed in this section. Each API is described with its parameters and generated output. APIs arguments and results are encoded as JSON objects.

3.1 `/detectionStatus` (GET)

This endpoint allows retrieving the information about the latest targets detected by the sensor. When the sensor is not detecting any target it returns the latest detected targets so the information is never lost. The `elapsedTime` and `acqIdx` fields can be used to discard information that has already been processed.

3.1.1 Returned value

```
{ "fps":<I:value>, "acqIdx":<I:value>,
  "elapsedTime":<I:value>,
  "targets":[ {"distance":<I:value>,
               "rcs":<I:value>,
               "id":<I:value>
             }
            ]
}
```

Where:

- fps is the frame per seconds that the sensor is able to process. Under normal conditions this value is greater or equal to 100
- acqIdx is the frame counter (a frame corresponds to the acquisition of the radar signal)
- elapsedTime how long has passed since these targets have been detected
- targets is an array of detected moving targets. Each target is described by the following info:
 - distance is the distance in [mm] from the sensor
 - rcs is a measure of the “size” of the detected target. For calibrated sensor this value is on average greater than 1000 when the target is the object used for calibration or anything bigger than that
 - id is the target identifier. The sensor tracks targets and assign them an ID. This identifier may change if a target is lost

3.2 /alarmStatus (GET)

This endpoint allows retrieving the latest alarm status and the last status when an alarm occurred. When the sensor is not detecting any alarm it returns the latest detected targets so the information is never lost. The elapsedTime and acqIdx fields can be used to discard information that has already been processed.

3.2.1 Returned value

```
{ "fps":<I:value>, "acqIdx":<I:value>, "elapsedTime":<I:value>,
  "alarms": [ {"type":<I:value>,
```

```

        "cause":<I:value>,
        "target": {"distance": <I:value>,
                  "rcs": <I:value>
                  }
      }
    ]
  }

```

Where:

- fps is the frame per seconds that the sensor is able to process. Under normal conditions this value is greater or equal to 100
- acqIdx is the frame counter (a frame corresponds to the acquisition of the radar signal)
- elapsedTime how long has passed since these targets have been detected
- alarms is an array with the latest alarms. Each alarm is described by the following fields:
 - type specifies the type of the alarm. It can be:
 - * 1 for an alarm generated by a detected target
 - 2 for an alarm generated by a fault situation
 - 3 for an alarm generated by a tampering situation
 - cause specifies the root cause of the alarm.
 - If the alarm is triggered by a moving target then this value can be:
 - * 4 if the alarm has been generated in the alarm area
 - * 2 if the alarm has been generated in the pre-alarm area
 - If the alarm is generated by a tampering situation its value can be:
 - * 2 if the sensor has been moved
 - * 3 if the sensor has been masked
 - If the alarm is generated by a fault situation its value can be:
 - * 100 for input voltage out of range
 - * 101 for detection error
 - * 102 for security error
 - * 103 for HW head fault
 - * 104 for masking
 - * 105 for motion detector not responding
 - * 106 for masking not ready
 - * 107 for corrupted storage
 - * 1001 for communication fault
 - target is an array of detected moving targets. Each target is

described by the following info:

- distance is the distance in [mm] from the sensor
- rcs is a measure of the “size” of the detected target. For calibrated sensor this value is on average greater than 1000 when the target is the object used for calibration or anything bigger than that
- id is the target identifier. The sensor tracks targets and assign them an ID. This identifier may change if a target is lost

3.3 /sensorConfiguration (GET and PUT)

Handles the sensor configuration. Most of the values are expressed as integer numbers and shall not be restricted to a subrange. The JSON definition here below shows the typical range just for convenience but there is no guarantee these ranges won't change in the future.

3.3.1 Configuration object encoding

The configuration object is returned when the endpoint is read via a GET operation and it is stored on the device when it is PUT.

```
{
  "configuration": {
    "alarmDistance": <I: [minDist, maxDist]>,
    "preAlarmDistance": <I: [minDist, maxDist]>,
    "alarmSensitivity": <I: [128:228]>,
    "mountHeight": <I: [1000-4000]>,
    "mountElevation": <I: [+90, -90]>,
    "mountRotation": <I: [-180, +180]>,
    "mountAzimuth": <I: [-180, +180]>,
    "calibrationCoefficient": <I: [300, 50000]>,
    "ledMode": <I: enumeration>,
    "channel": <I: [0-14]>,
    "configurationsCount": <I>,
    "country": <String: [3 character ISO code]>,
    "antiMaskingMode": <I: enumeration>,
    "antiMaskingExitMode": <I: enumeration>,
    "antiTamperMode": <I: enumeration>,
    "antiTamperExitMode": <I: enumeration>,
    "numConflictingSensors": <I: [0, 50]>,
    "ssMode": <I: enumeration>,
  }
}
```



```

        "ssMinDist":<I:[minDist,maxDist]>,
        "ssMaxSize":<I>,
        "ssNumMaxZones":<I>,
        "ssManualZones":[{"min":<I>,"max":<I>}]
    },
    "version":<I>
}

```

Where:

- alarmDistance: the length of the alarm area expressed in millimeters
- preAlarmDistance: the length of the pre-alarm area expressed in millimeters. This value must be greater or equal to the alarmDistance. In case there is no pre alarm then this value shall be equal to the alarmDistance
- alarmSensitivity: defines the level of pet tolerance. Higher values correspond to decreasing sensibility and increasing pet tolerance. The value of this parameter is a byte expressing a percentage [0-100] with the first bit set to 1. In other words the value is: percentage | 0x80 where percentage is a value in the [0-100] range. When the percentage is set to 75 (parameter is 75 | 0x80) the sensor triggers an alarm for a target whose size is comparable to the one used during calibration. Percentage values greater than 75 mean the sensor is less sensible and triggers alarms for targets bigger than the one that calibrated the sensor.
- mountHeight: defines the installation height in millimeters
- mountElevation: defines the sensor pitch. Usually this value is negative as negative values mean the sensor points downward. Values close to 0 mean the sensor is pointing parallel to the underlying ground.
- mountRotation: defines the sensor rotation. Values close to 0 degrees implies the sensor antenna is horizontal (volumetric mode) while values close to 90 degrees imply the sensor antenna is vertical (barrier mode)
- mountAzimuth: this value specifies the installation azimuth
- calibrationCoefficient: this value is used to calibrate the sensor in the environment where it operates. The correct value can be set using the automatic procedure (see the calibrationStatus api) or it can be manually set.
- ledMode: defines the led mode behavior.
 - 0 for normal led mode (red for alarm condition, blue for detected targets but no alarm yet, blinking red for pre alarm)
 - 1 for disabled mode

- **channel**: this parameter can be used to avoid interferences among sensors working in the same environment. 0 means this sensor is alone, 1 corresponds to channel A, 2 to channel B and so on
- **numConflictingSensors**: number of conflicting sensors in this environment. Each of them shall have a different channel to avoid interferences
- **antiMaskingMode**: specifies the anti masking mode. The parameter can take these values:
 - 0 for normal mode
 - 1 for disabled mode
 - 2 for aggressive mode (in this case anti masking is more sensible but it can generate false masking conditions if the sensor is exposed to direct rain)
- **antiTamperMode**: specifies the anti tamper mode. The parameter can take these values:
 - 0 for disabled mode
 - 2 for tamper on sensor moved mode
- **country**: the ISO3 code of the country where the sensor operates
- **configurationsCount**: this value is read only and counts the number of times the sensor has been configured. The value is returned when the configuration is fetched (GET) and it shall not be present when the configuration is stored (PUT). This value is reset on factory restore.
- **version**: the configuration format version. The value is returned when the configuration is fetched (GET) and it shall not be present when the configuration is stored (PUT).
- **ssMode**: semi statics mode. It can take one of the following values:
 - 0 for disabling semi statics
 - 1 for automatic mode 1 (i.e. max 2 semi static objects whose position is automatically found by the sensor and with a max size of 1.5 meters)
 - 2 for automatic mode 2 (i.e. max 3 semi static objects whose position is automatically found by the sensor and with a max size of 2.5 meters)
 - 3 for semi automatic mode (i.e. the position of semi statics is automatically found by the sensor but the max number and size is set by the user)
 - 4 for manual mode (everything is controlled by the user)

When the configuration is saved the caller can specify only the fields that

must change. All the unspecified fields will hold their current value. On configuration save the operation returns a JSON object like the following:

```
{"errorCode":code}
```

An error code 0 means no error (in that case the errorDesc is not present)

3.4 */sensorPositionStatus (GET)*

This endpoint returns current values of elevation and rotation

3.4.1 Returned value

```
{"elevation":<I>, "rotation": <I>}
```

Where:

- elevation is the sensor current pitch. Negative angles mean the sensor is pointing downward
- rotation is the sensor current rotation. Values around 0 mean the sensor is horizontal (volumetric) while values around 90 mean the sensor is vertical (barrier)

3.5 */sensorInfo (GET)*

Returns the sensors info (e.g. model, family, fw version)

3.5.1 Returned value

```
{"sensorInfo":{"model":<I>,"family":<I>,"fwVersion":<I>,"deviceId":<I>,"minDistance":<I>,"maxDistance":<I>,"minAngle":<I>,"maxAngle":<I>},"comBoardInfo":{"model":<I>,"family":<I>,"fwVersion":<I>,"deviceId":<I>,"pinResetRequired":<I>}}
```

```
}

```

The sensor info is split between the actual radar sensor and the communication board which is used to communicate over ethernet. For the sensor the following info is available: - model: the device model (must be 1 for current version of MSK-101-POE) - family: the device family (must be 5 for MSK-101-POE devices) - fwVersion: the firmware version - deviceId: the device unique id - minDistance: the minimum distance from the sensor at which a target can be detected - maxDistance: the maximum distance from the sensor at which a target can be detected - minAngle: the minimum angle from the sensor point of view at which a target can be detected - maxAngle: the maximum angle from the sensor point of view at which a target can be detected

The span between minAngle and maxAngle defines the sensor FOV

For the communication board the following info is available: - model: the com board model (must be 1 for the ethernet board embedded in the MSK-101-POE) - family: the com board family (must be 5 for the ethernet board embedded in the MSK-101-POE) - fwVersion: fw version of the communication board - deviceId: unique com board device id - pinResetRequired: specifies if the com board has a pin/password which the user needs to change

3.6 /calibrationStatus (PUT)

This endpoint allows starting/stopping/saving a sensor calibration.

3.6.1 Input argument

```
{"operation":<I:[0=stop,1=start standard,2=start zone based,3=save]>}
```

3.6.2 Return value

The API returns the status of the operation performed with an object with the following format:

```
{"errorCode":<I:[error code, where 0 means no error]>}
```

When the operation is “stop” the error code signals both blocking errors and warnings. In particular: - 0 means everything is OK - 223 means calibration data is poor, it is usable but the info is not very good - any other value means

the calibration procedure failed and the calibration data cannot be saved on sensor

3.7 /calibrationData (GET)

This endpoint allows retrieving information on the latest calibration performed (the information is not persisted on sensor so it does not survive reboot cycles). The returned JSON object contains data collected during calibration.

Return value

```
{"version":<I:calib data version>,"value":<I>,"mse":<I>,"peaks":[<I>]}
```

Where: - version is the data format version - value is the calibration coefficient as it has been computed - mse (Mean Square Error) gives an indication of how good the calibration was. Lower values indicate a good calibration - peaks is raw data collected during the calibration process. The meaning of these values is intentionally not explained here because the purpose is only to collect data that can be sent to the Inxpect engineering team for debugging purposes

3.8 /installationConfigurations (GET)

This endpoint allows retrieving the supported and optimal sensor configurations.

3.8.1 Returned value

```
{"installations": [{"minDistance": <I>,"maxDistance": <I>,"minElevation": <I>,"maxElevation": <I>,"minHeight": <I>,"maxHeight": <I>,"orientation": <I>}, {...}]}
```

Where installations is an array with the supported and optimal configurations - minDistance is the minimum distance at which this configuration can detect targets (**value in centimeters**) - maxDistance is the maximum distance up to which this configuration can detect targets (**value in centimeters**) - minElevation is the minimum elevation for this configuration to apply (**value**

is positive when sensor is pointing downward) - minElevation is the maximum elevation for this configuration to apply (**value is positive when sensor is pointing downward**) - minHeight is the minimum height for this configuration to apply (**value in centimeters**) - maxHeight is the minimum height for this configuration to apply (**value in centimeters**) - orientation is the orientation for this configuration to apply (0 for horizontal and 1 for vertical)

For example we could have the following configuration:

```
{"minDistance": 250,"maxDistance": 1600,
  "minElevation": 12,"maxElevation": 18,
  "minHeight": 175,"maxHeight": 224,
  "orientation": 0
}
```

This configuration specifies that the sensor can be installed with an height in the [175,224] cm range, with an inclination in the [-12,-18] degrees range, and horizontal orientation (volumetric). In this configuration the detection range is in the [250,1600] cm range.

3.9 /netSettings (GET and PUT)

This endpoint allows reading/writing the network configuration for the device.

3.9.1 Configuration encoding

The configuration is encoded into a JSON object like this:

```
{"netConfiguration":<I>,
  "staticAddress":<String:address>, "subnet":<String:address>,
  "gateway":<String:address>, "curNetConfiguration":<I:[bitMask]>,
  "curHostAddress":<String:address>, "curSubnet":<String:address>,
  "curGateway":<String:address>
}
```

Where netConfiguration is a bitmask where:

- bit 0 toggles dhcp
- bit 1 toggles name resolution
- bit 2 toggles auto IP

For example a value of “2” for the netConfiguration means that the sensor will use a static IP address (both dhcp and auto ip are off) and name resolution via mDNS and NetBIOS is active. When the sensor is configured for a static IP address the user can specify the address using the “staticAddress” field. If this field is not specified the sensor fallbacks to 192.168.0.50. If the netConfiguration field has the value “7” the sensor is configured to use DHCP first and if this fails it will revert to Auto IP which is guranteed to find a valid IP address.

The fields prefixes witht “cur” (e.g. curNetConfiguration) specifies the actual values and are not necessarily identical to the configuration saved by the user. Suppose for example the sensor was configured with DHCP but the sensor is unable to get an IP. In this case after a timeout the sensor reverts to Auto IP which guarantees the sensor will get an IP assigned.

When the configuration is stored (PUT) the fields “curNetConfiguration”, “curHostAddress”, “curSubnet” and “curGateway” MUST be omitted. The following object is returned as response:

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successfull.

3.10 /changePassword (PUT)

This endpoint allows setting the device password.

3.10.1 Password encoding

The new password is encoded into a JSON object like this:

```
{"oldPassword":<String>, "newPassword":<String>}
```

3.10.2 Return value

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successfull.

3.11 /comBoardName (GET and PUT)

This endpoint allows reading/writing the device network name.

3.11.1 Name encoding

The new network name is encoded into a JSON object like this:

```
{"name":<String>}
```

3.11.2 Return value

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successful.

3.12 /login (PUT)

This endpoint allows logging into the device. Logging in allows creating a session (token based) so that subsequent operations authenticate via the given token.

3.12.1 Password encoding

```
{"password":<String>}
```

3.12.2 Return value

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successful. A special error code with value 100 is returned when the operation was successful but the password is the default one. All the next requests will fail until the password is not changed.

3.13 /logout (PUT)

This endpoint allows logging out from the device. Once a client has logged out its session is no longer valid and a new login is required.

3.13.1 Return value

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successful.

3.14 */sensorReset (PUT)*

This endpoint allows resetting the sensor configuration to its factory defaults. The configuration that gets changed is the sensor configuration while the network settings, the password and the com board name are not changed.

3.14.1 Return value

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successful.

3.15 */comboardReset (PUT)*

This endpoint allows resetting the communication board configuration to its factory defaults. The configurations that gets changed are the network settings, the password and the com board name while the sensor configuration is not changed.

This command is also available without authentication for 20 seconds after a startup when the board has an active connection

3.15.1 Return value

```
{"errorCode":<I:code>}
```

An error code 0 means the operation was successful.

3.16 */updateToken (GET)*

This endpoint allows retrieving a new authentication token

3.16.1 Return value

```
{"errorCode":<I:code>, "token":<String>}
```

Where errorCode is 0 on success and token is the new token to be used in subsequent calls.

3.17 /eventLog (GET)

This endpoint returns the list of the latest alarms. For each alarm the sensor tracks detailed information at the beginning of the event and summarized information for the remaining time. An alarm event gets opened as soon as the sensor triggers an alarm condition and it is closed after 3 minutes where the alarm is OFF. If an alarm starts with a given type (e.g. movement detected) and then switches to another type (e.g. masking) then two different events are stored and the first one terminates as soon as the second one starts.

3.17.1 Return value

```
{ "events": [ { "type": <I:[1-3]>,"elapsedTime": <I:value>,
               "duration": <I:value>,"cause": <I:value>
             },
             { "type": <I:[1]>,
               "elapsedTime": <I:value>,
               "duration": <I:value>,
               "minDistance": <I:value>,
               "minRcs": <I:value>,
               "maxDistance": <I:value>,
               "maxRcs": <I:value>,
               "targets": [{"distance": <I:value>,
                           "rcs": <I:value>}
                         ]
             }
          ]
}
```

Where:

- type and cause values are the same described in alarm status
- elapsedTime is the time elapsed since the event start (in seconds)
- duration specifies how long the event lasted
- minDistance, minRcs, maxDistance, maxRcs are present only if the evtType is a presenceEvent. Distances are in mm and their meaning is the following:
 - minDistance is the minimum distance at which a target was detected in this event

- minRcs is the minimum rcs a target had in this event
- maxDistance is the maximum distance at which a target was detected in this event
- maxRcs is the maximum rcs a target had in this event
- targets are the values of distance and rcs between the alarm event. They are present only if the evtType is a presenceEvent. Distances are in mm

3.18 /startProcessUpdate (PUT)

This endpoint is used to notify the sensor about the beginning of an update session. An update session is specific for a device sub component (core sensore, communication board or web server).

3.18.1 Argument encoding

```
{“dst”:<I:[0: sensor, 1: comBoard, 2: server]>}
```

3.19 /startManifestUpdate (PUT)

This endpoint is used to notify the sensor about the beginning of a manifest update. The answer contains the chunk size that must be used to transfer data chunks

3.19.1 Return value

```
{ “errorCode”:, “maxChunkSize”:}
```

Where errorCode is 0 on success and in this case maxChunkSize is populated with the max dimension a chunk shall have.

3.20 /chunkManifestUpdate (PUT)

This endpoint shall be used to transmit a chunk of meta information. The max size of a chunk is returned by startManifestUpdate.

This request needs 2 HTTP special headers + X-Update-Filename: + Content-Range: bytes -/

Binary data (loaded from IFW meta information) shall be placed in the request body.

3.21 /stopManifestUpdate (PUT)

This endpoint informs the sensor that the metainformation has been completely transmitted.

3.21.1 Return value

```
{ "errorCode": }
```

Where errorCode is 0 on success

3.22 /startFileUpdate (PUT)

This endpoint is used to notify the sensor about the beginning of an item update. When the component being updated is either the core sensor or the communication board the item is actually the firmware image for the sub device. If the component being updated is the web server then the item can be any of the file in the web server tree (the web server has several files)

3.22.1 Return value

```
{ "errorCode":, "maxChunkSize": }
```

Where errorCode is 0 on success and in this case maxChunkSize is populated with the max dimension a chunk shall have.

3.23 /chunkFileUpdate

This endpoint shall be used to transmit a chunk of binary data. The max size of a chunk is returned by startFileUpdate.

This request needs 2 HTTP special headers + X-Update-Filename: + Content-Range: bytes -/

In the body of the request must be placed binary data (the chunk must be of the size specified in /startFileUpdate)

3.24 /stopFileUpdate

This endpoint informs the sensor that a file content has been completely transmitted.

3.24.1 Return value

```
{ "errorCode": }
```

Where errorCode is 0 on success

3.25 /stopProcessUpdate

This endpoint informs the sensor that an update process has completed. The client can specify if the update shall be terminated by force (in this case the update is aborted)

3.25.1 Argument

```
{ "force": }
```

A non zero value for the "force" parameter means the process shall be aborted

3.25.2 Return value

```
{ "errorCode": }
```

Where errorCode is 0 on success