OS^{3}P

Implementing JSON over websockets

Authors	P. Rademakers
	F. Bodewes
Version	0.91
Status	Initial release
Release date	12-07-2016

Version History

Version	Date	Author	Description
0.9	25.06.2016	Patrick Rademakers	Initial draft based upon the OCPP Implementation guide for JSON
0.91	12.07.2016	Frank Bodewes	Editorial changes related to protocol name and prepared document for initial release

Contents

List of Tables

1 Introduction

1.1 Purpose of this document

The purpose of this document is to give the reader the information required to create a correct interoperable OS^{3}P (Open Smart Secondary Substation Protocol) JSON implementation. We will try to explain what is mandatory, what is considered good practice and what one should not do, based on our own experience. Undoubtedly misunderstandings or ambiguities will remain but by means of this document we aim to prevent them as much as possible.

1.2 Intended audience

This document is intended for developers looking to understand and/or implement OS^{3}P in a correct and interoperable way. Rudimentary knowledge of implementing web services on a server or embedded device is assumed.

1.3 Why JSON?

JSON is the most commonly used lightweight notation format for data exchange between systems, and within systems internally. It stands for JavaScript Object Notation and finds it origins, as the name suggest in Javascript, a scripting language for web pages. Since its conception it has however been adopted by the development community as a whole as an easier to write, read and parse language than XML based formats. You'll find JSON support for nearly every programming language and environment nowadays, making development quite easy, even if both sides of the connection are implemented using totally different languages and tools.

1.4 Why websockets?

Standard web technology such as HTTP(S) is client server based, meaning that the only way the server, a central system, has of giving information to a client is in the form of a response to a client (PC/device) request. Only by switching roles, the client becoming server and the server becoming client, can the initiative for communication be reversed. This however us usually made very difficult by firewalls, NAT, etc. and also makes client side (PC/device) implementations much more difficult. A workaround for this has been to very frequently send "check for update" requests from the client, resulting in a lot of pointless communication.

Websockets have been devised to solve this problem. The initiative for the connection is still taken by the client to avoid firewall and NAT related issues, but in the case of websockets the connection remains open, thus creating a bidirectional connection where either side can initiate requests. For browsers this technology is used to show live updates of information on web pages. For devices this same technology offers a very effective way to have bidirectional data communication without the overhead of traditional polling strategies or complexities of connecting through firewalls and NAT's..

1.5 Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

1.6 Definitions & Abbreviations

IANA	Internet Assigned Numbers Authority (www.iana.org).
OS^{3}P	Open Smart Secondary Substation Protocol
RPC	Remote procedure call
WAMP	WAMP is an open websocket subprotocol that provides messaging patterns to handle asynchronous data.

1.7 References

[JSON]	http://www.json.org/
[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997. http://www.ietf.org/rfc/rfc2119.txt
[RFC2616]	http: http://tools.ietf.org/html/rfc2616
[RFC3629]	UTF-8: http://tools.ietf.org/html/rfc3629
[RFC3986]	URL: http://tools.ietf.org/html/rfc3986
[RFC6455]	Websockets: http://tools.ietf.org/html/rfc6455
[WAMP]	http://wamp.ws/
[WIKIWS]	http://en.wikipedia.org/wiki/websocket
[WS]	http://www.websocket.org/

2 Benefits & Issues

The websocket protocol is defined in [RFC6455]. Working implementations of earlier draft websocket specifications exist, but OS^{3}P implementations SHOULD use the protocol described in [RFC6455].

Be aware that websocket defines its own message structure on top of TCP. Data sent over a websocket, on a TCP level, is wrapped in a websocket frame with a header. When using a framework this is completely transparent. When working for an embedded system however, websocket libraries may not be available and then one has to frame messages correctly according to [RFC6455] him/herself.

3 Connection

For the connection between a Substation and a Central System using JSON over websockets, the Central System acts as a websocket server and the Substation acts as a websocket client.

3.1 Client request

To set up a connection, the Substation initiates a websocket connection as described in [RFC6455] section 4, "Opening Handshake".

OS^{3}P imposes extra constraints on the URL and the websocket subprotocol, detailed in the following two sections 4.1.1 and 4.1.2.

3.1.1 The connection URL

To initiate a websocket connection, the Substation needs a URL ([RFC3986]) to connect to. This URL is henceforth called the "connection URL". This connection URL is specific to a Substation. The Substation's connection URL contains the Substation identity so that the Central System knows which Substation a websocket connection belongs to.

A Central System supporting OS^{3}P MUST provide at least one OS^{3}P endpoint URL, from which the Substation SHOULD derive its connection URL. This OS3P endpoint URL can be any URL with a "ws" or "wss" scheme. How the Substation obtains an OS^{3}P endpoint URL is outside of the scope of this document.

To derive its connection URL, the Substation modifies the OS^{3}P endpoint URL by appending to the path first a '/' (U+002F SOLIDUS) and then a string uniquely identifying the Substation. This uniquely identifying string has to be percent-encoded as necessary as described in [RFC3986].

Example 1: for a Substation with identity “SUBS001” connecting to a Central System with OS^{3}P endpoint URL "ws://centralsystem.example.com/OS3P" this would give the following connection URL:

ws://centralsystem.example.com/OS3P/SUBS001

Example 2: for a Substation with identity “SUBS 123” connecting to a Central System with OS3P endpoint URL "wss://centralsystem.example.com/OS3P" this would give the following URL:

wss://centralsystem.example.com/OS3P/SUBS%20123

3.1.2 OS^{3}P version

The exact OCPP version MUST be specified in the Sec-Websocket-Protocol field. This SHOULD be one of the following values:

Table 1: OS^{3}P Versions

Protocol version	websocket subprotocol name
1.0	OS3P1.0
1.x	OS3P1.x
2.0	OS3P2.0

These are official websocket subprotocol name values. They are registered as such with IANA.

Note that OS3P 1.x and 2.0 are in the list. These are strictly examples. No future versions or work on them have been define at the writing of this document. Since the JSON over websocket solution is independent of the actual message content the solution can be used for all OS^{3}P versions.

It is considered good practice to include the OS^{3}P version as part of the OS^{3}P endpoint URL string. If you run a web service that can handle multiple protocol versions on the same OS^{3}P endpoint URL this is not necessary of course.

3.1.3 Example of an opening a websocket

A websocket is initiated by sending a HTTP request to a server. The following is an example of an opening HTTP request of an OS3P connection handshake:

```
GET /webServices/OS3P/SUBS123 HTTP/1.1

Host: some.server.com:44044

Upgrade: websocket

Connection: Upgrade

Sec-websocket-Key: x3JJHMbDL1EzLkh9GBhXDw==

Sec-websocket-Protocol: OS3P1.0, OS3P1.x

Sec-websocket-Version: 13
```

The bold parts are found as such in every websocket handshake request, the other parts are specific to this example.

In this example, the Central System's OS^{3}P endpoint URL is "ws://some.server.com:44044/webServices/OS3P". The Substation's unique identifier is "SUBS123", so the path to request becomes "webServices/OS3P/SUBS123".

With the Sec-websocket-Protocol header, the Substation indicates here that it can use OS3P1.0 and OS3P1.x, with the order of preference from left to right. If you only want or support one specific version you only indicate that version. Specifying multiple options gives the central system the deciding choice.

The other headers in this example are part of the HTTP and websocket protocols and are not relevant to those implementing OS^{3}P on top of third-party websocket libraries. The roles of these headers are explained in [RFC2616] and [RFC6455].

3.2 Server response

Upon receiving the Substation's request, the Central System has to finish the handshake with a response as described in [RFC6455].

The following OS^{3}P -specific conditions apply:

- If the Central System does not recognize the Substation identifier in the URL path, it SHOULD send an HTTP response with status 404 and abort the websocket connection as described in [RFC6455].

- If the Central System does not agree to using any of the subprotocols offered by the client, it MUST complete the websocket handshake with a response without a Sec-websocket-Protocol header and then immediately close the websocket connection.

So if the Central System accepts the above example request and agrees to using OS^{3}P 1.0 with the Substation, the Central System's response will look as follows:

```
HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-websocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=

Sec-websocket-Protocol: OS3P1.0
```

The bold parts are found as such in every websocket handshake response, the other parts are specific to this example.

The roleof the Sec-websocket-Accept header is explained in [RFC6455].

The Sec-websocket-Protocol header indicates that the server will be using OS^{3}P 1.0 on this connection.

3.3 More information

For those doing their own implementation of the websocket handshake, [WS] and [WIKIWS] give more information on the WebSocket protocol.

4 RPC framework

A websocket is a full-duplex connection, simply put a pipe where data goes in and data can come out and without a clear relation between in and out. The websocket protocol by itself provides no way to relate messages as requests and responses. To encode these request/response relations we need a small protocol on top of websocket. This problem occurs in more use cases of websocket so there are existing schemes to solve it. The most widely-used is WAMP (see [WAMP]) but with the current version of that framework handling RPCs symmetrically is not WAMP compliant. Since the required framework is very simple we have taken over the WAMP derivate made for OCPP over websockets.

Basically the need is very simple: we need to send a message (CALL) and receive a reply (CALLRESULT) or an explanation why the message could not be handled properly (CALLERROR). For possible future compatibility we will keep the numbering of these message in sync with WAMP. Our actual OS^{3}P message will be put into a wrapper that at least contains the type of message, a unique message Id and the payload, the message itself.

The whole message consisting of wrapper and payload MUST be valid JSON encoded with the UTF-8 (see [RFC3629]) character encoding.

Note that all valid US-ASCII text is also valid UTF-8, so if a system sends only US-ASCII text, all messages it sends comply with the UTF-8 requirement. A Substation or Central System SHOULD only use characters not in US-ASCII for sending natural-language text.

Note: You may find the Substation identity missing in the following paragraphs. The identity is exchanged during the websocket connection handshake and is a property of the connection. Every message is sent by or directed at this identity. There is therefore no need to repeat it in each message.

4.1.1 The message type

To identify the type of message one of the following Message Type Numbers MUST be used.

Table 2: Message types

MessageType	MessageTypeNumber	Remarks
CALL	2	Request
CALLRESULT	3	Response to specific request
CALLERROR	4	Error response to a specific request

When a server receives a message with a Message Type Number not in this list, it SHALL ignore the message payload. Each message type may have additional required fields.

4.1.2 The message ID

The message ID serves to identify a request. A message ID for a CALL message MUST be different from all message IDs previously used by the same sender for CALL messages on the same websocket connection. A message ID for a CALLRESULT or CALLERROR message MUST be equal to that of the CALL message that the CALLRESULT or CALLERROR message is a response to.

Table 3: Unique Message Id

Name	Datatype	Restrictions
messageId	string	Maximum of 36 characters, to allow for GUID’s

4.2 Call

A Call always consists of 4 elements: The standard elements MessageTypeId and UniqueId, a specific Action that is required on the other side and a payload, the arguments to the Action. The syntax of a call looks like this:

[<MessageTypeId>, “<UniqueId>”, “<Action>”, {<Payload>}]

Table 4: Call Fields

Field	Meaning
UniqueId	this is a unique identifier that will be used to match request and result.
Action	the name of the remote procedure or action. This will be a case-sensitive string containing.
Payload	Payload is a JSON object containing the arguments relevant to the *Action*. If there is no payload JSON allows for two different notations: *null* or an empty object *{}*. Although it seems trivial we consider it good practice to only use the empty object statement. Null usually represents something undefined, which is not the same as empty, and also *{}* is shorter.

As an example of a bootNotification could look like this:

[2, "19223201", "BootNotification", {"vendor": "VendorX", "model": "MDL1"}]

4.3 CallResult

If the call can be handled correctly the result will be a regular CallResult. Error situations that are covered by the definition of the OS^{3}P response definition are not considered errors in this context. They are regular results and as such will be treated as a normal CallResult, even if the result is undesirable for the recipient.

A CallResult always consists of 3 elements: The standard elements MessageTypeId and UniqueId and a payload, containing the response to the *Action* in the original Call. The syntax of a call looks like this:

[<MessageTypeId>, "<UniqueId>", {<Payload>}]

Table 5: CallResult Fields

Field	Meaning
UniqueId	This must be the exact same ID that is in the call request so that the recipient can match request and result.
Payload	Payload is a JSON object containing the results of the executed *Action*. If there is no payload JSON allows for two different

	notations: *null* or an empty object *{}*. Although it seems trivial we consider it good practice to only use the empty object statement. Null usually represents something undefined, which is not the same as empty, and also *{}* is shorter.

An example of a bootNotificationResponse could look like this:

[3, "19223201", {"status":"Accepted", "currentTime":"2013-02-01T20:53:32.486Z", "interval":300}]

4.4 CallError

We only use CallError in two situations:

1) An error occurred during the transport of the message. This can be a network issue, an availability of service issue, etc. Since CallError needs to refer to the original request, this only works if the original request ID can be discerned somehow. If this is not possible, which is very likely in this case, the call should simply be ignored.
2) The call is received but the content of the call does not meet the requirements for a proper message. This could be missing mandatory fields, an existing call with the same unique identifier is being handled already, unique identifier too long, etc.

A CallError always consists of 5 elements: The standard elements MessageTypeId and UniqueId, an errorCode string, an errorDescription string and an errorDetails object. The syntax of a call looks like this:

[<MessageTypeId>, "<UniqueId>", "<errorCode>", "<errorDescription>", {<errorDetails>}]

Table 6: CallError Fields

Field	Meaning
UniqueId	This must be the exact same id that is in the call request so that the recipient can match request and result.
ErrorCode	This field must contain a string from the ErrorCode table below.

ErrorDescription	Should be filled in if possible, otherwise a clear empty string “”.
ErrorDetails	This JSON object describes error details in an undefined way. If there are no error details you MUST fill in an empty object *{}*.

Table 7: Valid Error Codes

Error Code	Description
NotImplemented	Requested Action is not known by receiver
NotSupported	Requested Action is recognized but not supported by the receiver
InternalError	An internal error occurred and the receiver was not able to process the requested Action successfully
ProtocolError	Payload for Action is incomplete
SecurityError	During the processing of Action a security issue occurred preventing receiver from completing the Action successfully
FormationViolation	Payload for Action is syntactically incorrect or not conform the PDU structure for Action
PropertyConstraintViolation	Payload is syntactically correct but at least one field contains an invalid value
OccurenceConstraintViolation	Payload for Action is syntactically correct but at least one of the fields violates occurence constraints
TypeConstraintViolation	Payload for Action is syntactically correct but at least one of the fields violates data type constraints (e.g. *“somestring”: 12*)
GenericError	Any other error not covered by the previous ones

5 Connection

5.1 Security

For security we rely first and foremost on the security at a network level. Additionally the websocket standard specifies how websocket connections can be encrypted with Transport Layer Security (TLS). This combination is generally called secure websocket or wss. To use TLS, use "wss" as the URL scheme instead of "ws" (just like "https" and "http"). Issuing TLS certificates is the responsibility of the network operator.

A TLS-encrypted connection SHOULD always be used when the Central System and Substation communicate over a public network.

5.2 Compression

Since JSON is very compact we recommend not to use compression in any other form than allowed as part of the WebSocket [RFC6455] specification. Otherwise it may compromise interoperability.

5.3 Data integrity

For data integrity we rely on the underlying TCP/IP transport layer mechanisms.

5.4 websocket Ping in relation to OCPP Heartbeat

The WebsSocket specification defines Ping and Pong frames that are used to check if the remote endpoint is still responsive. In practice this mechanism is also used to prevent the network operator from quietly closing the underlying network connection after a certain period of inactivity. This websocket feature can be used as a substitute for most of the OCPP Heartbeat messages, but cannot replace all of its functionality.

An important aspect of the Heartbeat response is time synchronisation. The Ping and Pong frames cannot be used for this so at least one original Heartbeat message a day is recommended to ensure a correct clock setting on the Substation.

5.5 Reconnecting

When reconnecting, a Substation should not send a BootNotification unless one or more of the elements in the BootNotification have changed since the last connection. This means that after reconnecting, it may occur that no immediate message is sent by the connecting client.

6 Configuration

The following items in OS^{3}P Get/ChangeConfiguration messages are added to control JSON/websockets behaviour:

Table 8: Additional OS^{3}P Keys

Key	Value
websocketPingInterval	[0..MAXINT] 0 disables client side websocket Ping/Pong. In this case there is either no ping/pong or the server initiates the ping and client responds with Pong. Positive values are interpreted as number of seconds between pings. Negative values are not allowed. ChangeConfiguration is expected to return a REJECTED result.

Annex: OS^{3}P 0.4 JSON over websockets

OS^{3}P 0.4 JSON over websockets

2016-07-12

Table of Contents

1. Introduction

The basis for this document is the first JSON over Websockets experimental implementations for OCPP 1.5. This strictly experimental OCPP version quickly became so popular that it was further developed and officially published as OCPP 1.6 by the Open Charge Alliance (OCA) Foundation.

This document lists all OS3P (Open Smart Secondary Substation Protocol) 0.4 messages, and gives examples how they would look like in JSON. The name OS3P is a working title and might change in the future.

Note: Parameters to functions can already be found in the JSON schema but further details per function will be added. Configuration keys defining system behavior and giving relevant information to the Central System still need to be defined. Since the message set will not change (much) the current document, in combination the implementation guide can already be used to get an impression of complexity and ease of implementation.

1.1. Change History

2016-07-12	Editorial changes and conversion to word document
2016-07-03	Added Tariff as contactorId and Geolocation to configuration keys.
2016-06-29	Several modifications and clarifications.
2016-06-28	Changed GetDiagnostics to GetFile to retrieve specific period of meter values and other files.
2016-06-25	Restructuring of document, added explanations and made small changes
2016-06-25	Initial version

1.2. Open Issues

This paragraph lists all Open issues

1.2.1. StatusNotifcation messages:

Document the allowed StatusNotification statusses (not complete)

1.2.2. MeterValues

- disable LS measurments (bitfield)
- check the required MeterValues details.

1.2.3. Configuration keys

Expand the list of required and optional configuration keys and explain their meaning.

1.2.4. Tariff Schedules

Very similar to the streetlight schedules, tariff schedules will be added. As indication for complexity look at the streetlight functions for setting, retrieving and clearing of schedules.

2. Context & Security

TO BE DEFINED

In short, security will be handled on Websocket level and below.

3. Commisioning and boot behaviour

3.1. First boot

The very first time a Substation connects to a Central System, it MUST first send a BootNotification. A Central System may respond with a BootNotificationResponse. There are now three possible scenario's based upon de value of the status field in the response:

Status	Behaviour
Accepted	The Central System recognises and accepts the Substation. **Only now** may the Substation initiate other messages. It MAY sync its internal clock on the *currentTime* field. The *interval* field MUST be used to set the frequency of Heartbeat messages.
Rejected	If rejected, the Central System will terminate the connection. If the *interval* field has a non-zero value it MAY be used as a timeout before a next connection attempt. The Central System MAY refuse any incoming connection attempts

Status	Behaviour
	from the Substation during this period.
Pending	Unlike the previous case, the connection SHOULD remain open so that the Central System has time to check and possibly change configuration settings. The Substation MUST respond to these requests but still MAY not initiate requests. Only exception is when the Central System sends a TriggerMessage request. In that case the Substation may initiate the requested message.

Note 1: First Boot behaviour also applies to a first boot after a firmware update.
Note 2: First Boot behaviour also applies when the URL to the Central System has changed.

3.2. Reboots

A Substation MUST send a new BootNotification everytime a one or more values of the fields in BootNotification have changed. This includes a firmware update if the firmware version value changes. If nothing has changed the BootNotification MAY still be sent but this is not required. If the reboot was the result of unexpected behaviour, the Substation must report this to the Central System via a StatusNotification.

3.3. Restoring network connection

If, for whatever reason, the network connection is lost the Substation is required to reconnect as soon as possible. Once reconnected, a BootNotification is not required unless one or more values in the BootNotification has changed.

4. Normal operations

This is a birectional protocol, meaning there are messages that a Substation can send to the Central System, and that there are other messages a Central System can send to a Substation. The Substation messages are aimed at sharing information such as measured data, configuration settings and system status. The Central System messages are aimed at managing configuration and behavior of a Substation.

4.1. Forcing Client-initated messages: TriggerMessage

Each side has its own responsibility for content and timing of the messages it is supposed to send. This is how it should be. Sometimes however, a Central System may want to receive Substation initiated messages immediately. E.g. when a Substation sends quarterly MeterValues. Suppose it sent the last message 5 minutes ago but somehow an alarm bell goes off and you want know what is happening right now. You would typically have to wait another 10 minutes.

Another example is during initial boot. Suppose you, as a Central System, get a BootNotification from a new Substation. Part of the commissioning process is that you return Pending and a delay when it may try again. In the meantime, this give the Central System time to check and change the configuration of the Substation. To make sure you have enough time to do what needs to be done you return with the Pending a retry interval of 15 minutes. If this Substation turns out to be perfectly configured already you are done after 10 seconds. You don't want to wait.

These are underlying reasons for the creation of the TriggerMessage message. It is Central System initiated but all is does is tell the Substation that it should initiate a specific message. The Substation can either Accept or Reject the message. If support for TriggerMessage is not implemented the Substation will return NotImplemented as the result status. It will Reject the request if it does support the operation but for whatever reason feels it cannot comply. Most commonly though, the result will be Accept, indicating the requested message SHALL be sent immediately.

When the requested message is sent, it will return the most recent information: either the current status of something or fresh readings/measurements. If a MeterValues message is requested you will get fresh readings. If a FirmwareStatusNotification is requested, you will get the actual status of the firmware update progress. If you do this while no firmware update was ongoing, you will get None.

4.2. Requests not supported or implemented

Many messages have in their response a status field, indicating a request has either been accepted or rejected. There are two special scenarios that are not covered by the status field but are instead handled in the WAMP layer:

1. **Not implemented:** Some request may be optional. In those cases the implementer can choose to skip them. In that case a WAMP level error MUST be returned with the value `NotImplemented`. the response SHOULD always be the same, unless perhaps after a firmware update.

2. **Not supported:** The WAMP level error `NotSupported` indicates that the requested function is implemented, but for some other reason cannot be handled. This may be due to a configuration setting, an unsupported value or combination of values, etc.

4.3. Retrieving log files

Under discussion

4.4. Updating Firmware

Under discussion

5. Configuration keys

Configuration keys that show settings and possibly influence behavior can be retrieved via the GetConfiguration request. If a settings is marked RW it is also writeable and can be changed via the ChangeConfiguration request. RO stands for Read Only. If marked RO a ChangeConfiguration attempt will be rejected.

key	RO|RW	type	description
Sunrise	RO	datetime	the sunrise time of today
Sunset	RO	datetime	the sunset time of today
Geolocation	RW	latlng	Lattitude and longitude, to be used as basis for astronomical clock.

6. Client-initiated Messages

6.1. BootNotification

6.1.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "BootNotificationRequest",
    "type": "object",
    "properties": {
        "vendor": {
            "type": "string",
            "maxLength": 20
        },
        "model": {
            "type": "string",
            "maxLength": 20
        },
        "stationNumber": {
            "type": "string",
            "maxLength": 10
        },
        "serialNumber": {
            "type": "string",
            "maxLength": 25
        },
        "firmwareVersion": {
            "type": "string",
            "maxLength": 50
        },
        "iccid": {
            "type": "string",
            "maxLength": 20
        },
        "imsi": {
            "type": "string",
            "maxLength": 20
        }
    },
    "additionalProperties": false,
        "required": [
        "vendor",
        "model",
        "stationNumber",
        "serialNumber",
        "firmwareVersion",
        "iccid",
        "imsi"
    ]

}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "BootNotificationResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Pending",
                "Rejected"
            ]
        },
        "currentTime": {
            "type": "string",
            "format": "date-time"
        },
        "interval": {
            "type": "integer"
        }
    },
    "additionalProperties": false,
    "required": [
        "status",
        "currentTime",
        "interval"
    ]
}
```

6.1.2. examples

request:

```
{
  "vendor": "BrandX",
  "model": "SUBS-A1",
  "stationNumber": "987654321",
  "serialNumber": "AQ1234VB5678",
  "firmwareVersion": "1.1.0",
  "iccid": "83310415107943019317",
  "imsi": "080921439201835338"
}
```

response:

```
{
  "status": "Accepted",
  "currentTime": "2013-02-01T15:09:18Z",
  "interval": 1200
}
```

6.2. GetFileStatusNotification

6.2.1. remarks

If the underlying GetFileRequest has given an explicit `requestId`, it is mandatory to use this identifier in all related GetFileStatusNotification's.

6.2.2. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetFileStatusNotificationRequest",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                                "None",
                "Uploaded",
                "Uploading",
                "UploadFailed"
            ]
        },
        "requestId": {
            "type": "string",
            "maxLength": 25
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetFileStatusNotificationResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

6.2.3. examples

request:

```
{
  "status": "Uploaded"
}
```

response:

```
{}
```

6.3. FirmwareStatusNotification

6.3.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "FirmwareStatusNotificationRequest",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                                    "None",
                "Downloading",
                "Downloaded",
                "DownloadFailed",
                "Installing",
                "InstallationFailed",
                "Installed"
            ]
        },
        "requestId": {
            "type": "string",
            "maxLength": 25
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "FirmwareStatusNotificationResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

6.3.2. examples

request:

```
{
  "status": "DownloadFailed"
}
```

response:

```
{}
```

6.4. Heartbeat

6.4.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "HeartbeatRequest",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "HeartbeatResponse",
    "type": "object",
    "properties": {
        "currentTime": {
            "type": "string",
            "format": "date-time"
        }
    },
    "additionalProperties": false,
    "required": [
        "currentTime"
    ]
}
```

6.4.2. examples

request:

```
{}
```

response:

```
{
  "currentTime": "2013-02-01T15:09:18Z"
}
```

6.5. MeterValues

Table 1. MeterValues details

Measurand	Unit

Table 1. MeterValues details

Measurand	**Unit**
Location	Description
AmbientTemperature	Celsius
-	Ambient temperature
OilTemperature	Celsius
-	temperature of transformer
BoardTemperature	Celsius
-	temperature of the board
Voltage.L1N	V
-	Actual voltage phase L1
Voltage.L2N	V
-	Actual voltage phase L2
Voltage.L3N	V
-	Actual voltage phase L3
Current.N	A

Table 1. MeterValues details

Measurand	Unit
-	Actual three phase neutral current
Current.L1	A
-	Actual station current phase L1
Current.L2	A
-	Actual station current phase L2
Current.L3	A
-	Actual station current phase L3
Current.Sum	A
-	Actual station current (L1 + L2 + L3)
PowerFactor.L1	-
-	Power factor (cos phi) L1
PowerFactor.L2	-
-	Power factor (cos phi) L2
PowerFactor.L3	-

Table 1. MeterValues details

Measurand	Unit
-	Power factor (cos phi) L3
PowerFactor.Sum	-
-	Power factor (cos phi) of total
ActivePower.L1	W
-	Active power usage phase L1
ActivePower.L2	W
-	Active power usage phase L2
ActivePower.L3	W
-	Active power usage phase L3
ActivePower.Sum	W
-	Active power usage total
ApparentPower.L1	VA
-	Apparent power usage phase L1
ApparentPower.L2	VA

Table 1. MeterValues details

Measurand	Unit
-	Apparent power usage phase L2
ApparentPower.L3	VA
-	Apparent power usage phase L3
ApparentPower.Sum	VA
-	Apparent power usage total
ReactivePower.L1	var
-	reactive power phase L1
ReactivePower.L2	var
-	reactive power phase L2
ReactivePower.L3	var
-	reactive power phase L3
ReactivePower.Sum	var
-	reactive power total
RealEnergyConsumed.L1	Wh

Table 1. MeterValues details

Measurand	Unit
-	Real energy consumed phase L1
RealEnergyConsumed.L2	Wh
-	Real energy consumed phase L2
RealEnergyConsumed.L3	Wh
-	Real energy consumed phase L3
RealEnergyConsumed.Sum	Wh
-	Real energy delivered total
RealEnergyDelivered.L1	Wh
-	Real energy delivered phase L1
RealEnergyDelivered.L2	Wh
-	Real energy delivered phase L2
RealEnergyDelivered.L3	Wh
-	Real energy delivered phase L3
RealEnergyDelivered.Sum	Wh

Table 1. MeterValues details

Measurand	Unit
-	Real energy delivered total
ApparentEnergy.L1	VAh
-	apparent energy phase L1
ApparentEnergy.L2	VAh
-	apparent energy phase L2
ApparentEnergy.L3	VAh
-	apparent energy phase L3
ApparentEnergy.Sum	VAh
-	apparent energy total
ReactiveEnergy.L1	varh
-	reactive energy phase L1
ReactiveEnergy.L2	varh
-	reactive energy phase L2
ReactiveEnergy.L3	varh

Table 1. MeterValues details

Measurand	Unit
-	reactive energy phase L3
ReactiveEnergy.Sum	varh
-	reactive energy total
THDv.L1N	%
-	Total Harmonic Distortion of voltage, phase L1
THDv.L2N	%
-	Total Harmonic Distortion of voltage, phase L2
THDv.L3N	%
-	Total Harmonic Distortion of voltage, phase L3
THDc.L1	%
-	Total Harmonic Distortion of current, phase L1
THDc.L2	%
-	Total Harmonic Distortion of current, phase L2
THDc.L3	%

Table 1. MeterValues details

Measurand	Unit
-	Total Harmonic Distortion of current, phase L3
Frequency	Hz
-	Line Frequence
Current.L1	Amp
P1Meter	actual current phase L1 streetlights
Current.L2	Amp
P1Meter	actual current phase L2 streetlights
Current.L3	Amp
P1Meter	actual current phase L3 streetlights
ActivePower.L1	kW
P1Meter	actual power usage phase L1 streetlights
ActivePower.L2	kW
P1Meter	actual power usage phase L2 streetlights
ActivePower.L3	kW

Table 1. MeterValues details

Measurand	Unit
P1Meter	actual power usage phase L3 streetlights
Energy.Active.Import.Register	kWh
P1Meter	Total energy delivered (total of tarif I + II) streetlights
Power.Active.Import	kW
P1Meter	Actual power (L1 + L2 + L3) streetlights
DayMax.Voltage.L1	V
-	Daily maximum voltage phase L1
DayMax.Voltage.L2	V
-	Daily maximum voltage phase L2
DayMax.Voltage.L3	V
-	Daily maximum voltage phase L3
DayMax.Current.L1	A
-	Daily maximum station current phase L1
DayMax.Current.L2	A

Table 1. MeterValues details

Measurand	Unit
-	Daily maximum station current phase L2
DayMax.Current.L3	A
-	Daily maximum station current phase L3
DayMax.ActivePower.L1	W
-	Daily maximum Active power usage phase L1
DayMax.ActivePower.L2	W
-	Daily maximum Active power usage phase L2
DayMax.ActivePower.L3	W
-	Daily maximum Active power usage phase L3
DayMax.ReactivePower.L1	var
-	Daily maximum reactive power phase L1
DayMax.ReactivePower.L2	var
-	Daily maximum reactive power phase L2
DayMax.ReactivePower.L3	var

Table 1. MeterValues details

Measurand	Unit
-	Daily maximum reactive power phase L3
DayMax.Energy.Active.Import.Register	kWh
P1Meter	Total energy delivered during day
DayMin.Voltage.L1	V
-	Daily minimum voltage phase L1
DayMin.Voltage.L2	V
-	Daily minimum voltage phase L2
DayMin.Voltage.L3	V
-	Daily minimum voltage phase L3
DayMin.ActivePower.L1	W
-	Daily minimum Active power usage phase L1
DayMin.ActivePower.L2	W
-	Daily minimum Active power usage phase L2
DayMin.ActivePower.L3	W

Table 1. MeterValues details

Measurand	Unit
-	Daily minimum Active power usage phase L3
DayMin.ReactivePower.L1	var
-	Daily minimum reactive power phase L1
DayMin.ReactivePower.L2	var
-	Daily minimum reactive power phase L2
DayMin.ReactivePower.L3	var
-	Daily minimum reactive power phase L3

The following are no longer used, but older version of the firmware might still send these.

Table 2. Depricated MeterValue details

ReactiveEnergyInductive.L1	varh
-	reactive energy inductive phase L1
ReactiveEnergyInductive.L2	varh
-	reactive energy inductive phase L2
ReactiveEnergyInductive.L3	varh

Table 2. Depricated MeterValue details

ReactiveEnergyInductive.L1	**varh**
-	reactive energy inductive phase L3
ReactiveEnergyInductive.Sum	varh
-	reactive energy inductive total
ReactiveEnergyCapacitive.L1	varh
-	reactive energy capacitive phase L1
ReactiveEnergyCapacitive.L2	varh
-	reactive energy capacitive phase L2
ReactiveEnergyCapacitive.L3	varh
-	reactive energy capacitive phase L3
ReactiveEnergyCapacitive.Sum	varh
-	reactive energy capacitive total

Table 3. default values of optional fields

field	**default value**
context	`Sample.Periodic`

Table 3. default values of optional fields

field	default value
format	`Raw`
unit	-
location	-

6.5.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "MeterValuesRequest",
    "type": "object",
    "properties": {
        "meterValue": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "timestamp": {
                        "type": "string",
                        "format": "date-time"
                    },
                    "sampledValue": {
                        "type": "array",
                        "items": {
                            "type": "object",
                            "properties": {
                                "value": {
                                    "type": "string"
                                },
                                "context": {
                                    "type": "string",
                                    "enum": [
                                        "Interruption.Begin",
                                        "Interruption.End",
                                        "Sample.Clock",
                                        "Sample.Periodic",
                                        "Trigger",
                                        "Other"
                                    ]
                                },
                                "format": {
                                    "type": "string",
                                    "enum": [
                                        "Raw",
                                        "SignedData"
                                    ]
                                },
                                "measurand": {
                                    "type": "string",
                                    "enum": [
                                        "AmbientTemperature",
                                        "OilTemperature",
                                        "Voltage.L1N",
                                        "Voltage.L2N",
                                        "Voltage.L3N",
                                        "Current.L1",
                                        "Current.L2",
                                        "Current.L3",
                                        "Current.N",
                                        "Current.Sum",
                                        "PowerFactor.L1",
                                        "PowerFactor.L2",
                                        "PowerFactor.L3",
                                        "PowerFactor.Sum",
                                        "ActivePower.L1",
                                        "ActivePower.L2",
                                        "ActivePower.L3",
                                        "ActivePower.Sum",
                                        "ApparentPower.L1",
                                        "ApparentPower.L2",
                                        "ApparentPower.L3",
                                        "ApparentPower.Sum",
                                        "ReactivePower.L1",
                                        "ReactivePower.L2",
                                        "ReactivePower.L3",
                                        "ReactivePower.Sum",
                                        "RealEnergyConsumed.L1",
                                        "RealEnergyConsumed.L2",
                                        "RealEnergyConsumed.L3",
                                        "RealEnergyConsumed.Sum",
                                        "RealEnergyDelivered.L1",
                                        "RealEnergyDelivered.L2",
                                        "RealEnergyDelivered.L3",
                                        "RealEnergyDelivered.Sum",
                                        "ApparentEnergy.L1",
                                        "ApparentEnergy.L2",
                                        "ApparentEnergy.L3",
                                        "ApparentEnergy.Sum",
                                        "ReactiveEnergy.L1",
                                        "ReactiveEnergy.L2",
                                        "ReactiveEnergy.L3",
                                        "ReactiveEnergy.Sum",
                                        "THDv.L1N",
                                        "THDv.L2N",
                                        "THDv.L3N",
                                        "THDc.L1",
                                        "THDc.L2",
                                        "THDc.L3",
                                        "Frequency",

"Energy.Active.Import.Register",
                                        "Power.Active.Import",

"ReactiveEnergyInductive.L1",

"ReactiveEnergyInductive.L2",

"ReactiveEnergyInductive.L3",

"ReactiveEnergyInductive.Sum",

"ReactiveEnergyCapacitive.L1",

"ReactiveEnergyCapacitive.L2",

"ReactiveEnergyCapacitive.L3",

"ReactiveEnergyCapacitive.Sum",

"DayMax.Voltage.L1",

"DayMax.Voltage.L2",

"DayMax.Voltage.L3",

"DayMax.Current.L1",

"DayMax.Current.L2",

"DayMax.Current.L3",

"DayMax.ActivePower.L1",

"DayMax.ActivePower.L2",
```

```json
"DayMax.ActivePower.L3",

"DayMax.ReactivePower.L1",

"DayMax.ReactivePower.L2",

"DayMax.ReactivePower.L3",

"DayMax.Energy.Active.Import.Register",

"DayMin.Voltage.L1",

"DayMin.Voltage.L2",

"DayMin.Voltage.L3",

"DayMin.ActivePower.L1",

"DayMin.ActivePower.L2",

"DayMin.ActivePower.L3",

"DayMin.ReactivePower.L1",

"DayMin.ReactivePower.L2",

"DayMin.ReactivePower.L3"
                                    ]
                                },
                                "location": {
                                    "type": "string",
                                    "enum": [
                                        "In",
                                        "Out",
                                        "Body",
                                        "P1Meter"
                                    ]
                                },
                                "unit": {
                                    "type": "string",
                                    "enum": [
                                        "Wh",
                                        "kWh",
                                        "varh",
                                        "kvarh",
                                        "W",
                                        "kW",
                                        "VA",
                                        "VAh",
                                        "kVA",
                                        "var",
                                        "kvar",
                                        "A",
                                        "V",
                                        "K",
                                        "Hz",
                                        "C",
                                        "%"
                                    ]
                                }
                            },
                            "required": [
                                "value",
                                "measurand"
                            ]
                        }
                    }
                },
                "required": [
                    "timestamp",
                    "sampledValue"
                ]
            }
        }
    },
    "additionalProperties": false,
    "required": [
        "meterValue"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "MeterValuesResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

6.5.2. examples

request:

```
{
  "meterValue": [
    {
      "timestamp": "2013-03-07T16:52:16Z",
      "sampledValue": [
        {
          "value": "12",
          "unit": "A",
          "measurand": "Current.L1"
        },
        {
          "value": "300",
          "unit": "kWh",
          "measurand": "Energy.Active.Import.Register"
        }
      ]
    },
    {
      "timestamp": "2013-03-07T19:52:16Z",
      "sampledValue": [
        {
          "value": "20",
          "unit": "kWh",
          "measurand": "Energy.Active.Import.Register"
        },
        {
          "value": "40",
          "unit": "kW",
          "location": "P1Meter",
          "measurand": "ActivePower.L1"
        }
      ]
    }
  ]
}
```

response:

```
{}
```

6.6. StatusNotification

Note: Streetlight statusses are not send via this message, but rather via StreetlightStatusNotification and StreetlightModeStatusNotification

Table 4. StatusNotifcation fields

field	description

Table 4. StatusNotifcation fields

field	description
`test`	(optional) if test is true, this error is a result from a hardware test, not from a *real* error situation
`status`	indicates the status of the transformer station as a whole. If this is Faulted, the station is in an error state. This doesn't necessarily mean that the current status notification is describing an error as it could be a message indicating that an error is resolved (but not necessarily all errors).
`errorCode`	indicates whether this status notification describes an error and which error this is. If this is NoError, this notification does not describe an error and the info field could contain a string indicating whether an error has been resolved. If this is OtherError, the vendorId and vendorErrorCode should be given to indicate which error this describes.
`info`	(optional) . Additional free format information related to the error. NB: This is (ab)used to mark specific errors as solved. See table below.
`timestamp`	(optional) describes the moment at which the given state or error was discovered.
`vendorId`	(optional) the unique identification for the vendor who specified this custom error.
`vendorErrorCode`	(optional) the identification of the custom error provided by the vendor.

The `info` field is not only free format. We use some fixed strings to communicate 3 types of errors to be resolved. These are documented in the table below.

Table 5. StatusNotification info field

info text	meaning
`Indicator fault (MV) resolved`	the `MVFaultIndicatorError` is resolved
`Over current error (MV) resolved`	the `OverCurrentFailure` of the Medium Voltage Fault indicator is resolved
`Ground failure resolved`	the `GroundFailure` of the Medium Voltage Fault indicator is resolved

Table 6. StatusNotification statusses (not complete)

status	description
`GroundFailure`	Ground failure detected (or in test mode), `info` field indicates which group
`MVFaultIndicatorError`	Error dected in functionality of MVFaultIndicator, `info` field indicates which group
`OverCurrentFailure`	Overcurrent detected (or in test mode), `info` field indicates which group

6.6.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "StatusNotificationRequest",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Ok",
                "Faulted"
            ]
        },
        "errorCode": {
            "type": "string",
            "enum": [
                "GroundFailure",
                "HighTemperature",
                "MVFaultIndicatorError",
                "NoError",
                "OtherError",
                "OverCurrentFailure",
                "PowerMeterFailure",
                "PowerSwitchFailure",
                "ResetFailure",
                "UnderVoltage",
                "OverVoltage",
                "WeakSignal"
            ]
        },
        "info": {
            "type": "string",
            "maxLength": 128
        },
        "timestamp": {
            "type": "string",
            "format": "date-time"
        },
        "vendorId": {
            "type": "string",
            "maxLength": 128
        },
        "vendorErrorCode": {
            "type": "string",
            "maxLength": 64
        },
        "test": {
            "type": "boolean",
            "description": "true indicates that this is not a real
error, but rather a notification resulting from a hardware test"
        }
    },
    "additionalProperties": false,
    "required": [
        "status",
        "errorCode"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "StatusNotificationResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

6.6.2. examples

request:

```
{
  "status": "Ok",
  "errorCode": "NoError",
  "info": "",
  "timestamp": "2013-02-01T15:09:18Z",
  "vendorId": "",
  "vendorErrorCode": ""
}
```

response:

```
{}
```

6.7. StreetlightModeStatusNotification

This messages is used to inform the backend that the scheduling mode of the streetlight changed. This message might be triggered by a OverrideStreetlight or ResumeStreetlight request, but it might also come spontanious. (when a user enabled the local override mode via the hardware switch).

Note: The station garantees delivery of these events. In case communication is down communication the station will store these events, and send the messages later. The station will retry untill each event is delivered. Only when the backend did send a response, the station considers the event delivered.

Table 7. Streetlight mode is notified using these fields

field	description
`contactorId`	indicates from which contactorId we're sending the mode, if left out the mode is valid for **all** contactors.

Table 7. Streetlight mode is notified using these fields

field	description
`mode`	`Schedule`, `LocalOverride` or `RemoteOverride`. LocalOverride means locally in the station someone pulled the manual override switch. RemoteOverride is triggered by a OverrideStreetlight message, Schedule means the station switches the lights automatically according to its schedule.
`timestamp`	actual timestamp of the change moment.

Note: When a localoverride is implemented as a station wide switch, the enabling/disabling of local override will always communicate the mode of **all** contactorIds. Either by leaving out the `contactorId` field, or by sending multiple `StreetlightModeStatusNotification` messages.

6.7.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "StreetlightModeStatusNotificationRequest",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "mode": {
            "type": "string",
            "enum": [
                "Schedule",
                "LocalOverride",
                "RemoteOverride"
            ]
        },
        "timestamp": {
            "type": "string",
            "format": "date-time"
        }
    },
    "additionalProperties": false,
    "required": [
        "mode",
        "timestamp"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "StreetlightModeStatusNotificationResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

6.7.2. examples

example for all contactorIds

request:

```
{
  "mode": "LocalOverride",
  "timestamp": "2013-02-01T15:09:18Z"
}
```

example for specific contactorId

request:

```
{
  "contactorId": 1,
  "mode": "Schedule",
  "timestamp": "2013-02-01T15:09:18Z"
}
```

response:

```
{}
```

6.8. StreetlightStatusNotification

This message is used to inform the backend of actual light switches. This is send on ON→OFF, OFF→ON transitions.

Note: The station garantees delivery of these events. In case communication is down communication the station will store these events, and send the messages later. The station will retry untill each event is delivered. Only when the backend did send a response, the station considers the event delivered.

Table 8. Streetlight status is notified using these fields

field	description
`contactorId`	indicates from which contactorId we’re sending the status
`status`	`On` or `Off`, indicates if lights are on or off
`timestamp`	actual timestamp of the switch moment. (This may differ slightly from the scheduled time)
`unexpected`	(optional) true indicates this light was switch unexpectedly, not controlled by the RTU. This might indicate someone manually pulls a switch, of that a fault occurs.

6.8.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "StreetlightStatusNotificationRequest",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "status": {
            "type": "string",
            "enum": [
                "On",
                "Off"
            ]
        },
        "timestamp": {
            "type": "string",
            "format": "date-time"
        },
        "unexpected": {
            "type": "boolean"
        }
    },
    "additionalProperties": false,
    "required": [
        "contactorId",
        "status",
        "timestamp"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "StreetlightStatusNotificationResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

6.8.2. examples

request:

```
{
  "contactorId": 2,
  "status": "On",
  "timestamp": "2013-02-01T15:09:18Z"
}
```

response:

```
{}
```

7. Server-initiated Messages

7.1. ChangeConfiguration

Table 9. ChangeConfiguration parameters

key	description

Table 9. ChangeConfiguration parameters

key	description
example value	StreetlightProfileMaxStackLevel
highest stackLevel that may be used in SetStreetlightProfile	"16"

7.1.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ChangeConfigurationRequest",
    "type": "object",
    "properties": {
        "key": {
            "type": "string"
        },
        "value": {
            "type": "string"
        }
    },
    "additionalProperties": false,
    "required": [
        "key",
        "value"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ChangeConfigurationResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.1.2. examples

request:

```
{
  "key": "KVCBX_LANG",
  "value": "FR"
}
```

response:

```
{
  "status": "Accepted"
}
```

7.2. ClearSingleStreetlightProfile

ClearSingleStreetlightProfile can be used to clear a single schedule. This can be done by giving a specific and unique profileId, or by specifying a unique combination of `contactorId` and `stackLevel`.

To make sure no schedules are left behind when installing completely new schedules, the SetStreetlightProfile offers the flag `clearAllSchedules`. In that case using ClearSingleStreetlightProfile to first clear old schedules would be redundant.

`contactorId`	1-4 indicates which contactorId to erase, leaving this field out implies that this clear request is requested for **all** contactorIds.
`profileId`	indicates which profile must be deleted. If left out, and `stackLevel` is left out too, all profiles for this specific contactorId are deleted. `stacklevel` and `profileId` are mutual exclusive
`stackLevel`	indicates which profile must be deleted. If left out, and `profileId` is left out too, all profiles for this specific contactorId are deleted. `stackLevel` and `profileId` are mutual exclusive.

7.2.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ClearSingleStreetlightProfileRequest",
    "type": "object",
    "properties": {
        "profileId": {
            "type": "integer",
            "minimum": 0
        },
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "stackLevel": {
            "type": "integer",
            "minimum": 0
        }
    },
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ClearSingleStreetlightProfileResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected",
                "Unknown"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.2.2. examples

request:

```
{
    "profileId": 2,
    "contactorId": 1,
    "stackLevel": 3
}
```

response:

```
{
    "status": "Accepted"
}
```

7.3. GetConfiguration

7.3.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetConfigurationRequest",
    "type": "object",
    "properties": {
        "key": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    },
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetConfigurationResponse",
    "type": "object",
    "properties": {
        "configurationKey": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "key": {
                        "type": "string"
                    },
                    "readonly": {
                        "type": "boolean"
                    },
                    "value": {
                        "type": "string"
                    }
                },
                "required": [
                    "key",
                    "readonly"
                ]
            }
        },
        "unknownKey": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    },
    "additionalProperties": false
}
```

7.3.2. examples

request:

```
{
  "key": [
    "KVCBX_PROFILE"
  ]
}
```

response:

```
{
  "configurationKey": [
    {
      "key": "KVCBX_PROFILE",
      "readonly": true,
      "value": "NQC-ACDC"
    }
  ],
  "unknownKey": []
}
```

7.4. GetEffectiveStreetlightDaySchedule

This message can be used to check back office and station implementations. This returns the effective schedule for a specific day.

Table 10. GetEffectiveStreetlightDaySchedule parameters

`contactorId`	indicates for which contactorId the request is intended
`at`	date on which to get the schedule. This must be in **local** time (not utc) and only the date part is used.

7.4.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetEffectiveStreetlightDayScheduleRequest",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "at": {
            "type": "string",
            "format": "date-time"
        }
    },
    "additionalProperties": false,
    "required": [
        "contactorId",
        "at"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetEffectiveStreetlightDayScheduleResponse",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "daySchedule": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "start": {
                        "type": "string"
                    },
                    "end": {
                        "type": "string"
                    }
                },
                "additionalProperties": false,
                "required": [
                    "start",
                    "end"
                    ]
            }
        }
    },
    "additionalProperties": false,
    "required": [
        "contactorId",
        "daySchedule"
    ]
}
```

7.4.2. examples

request:

```
{
    "contactorId": 1,
    "at": "2015-07-07T00:00:00"
}
```

response:

```
{
    "status": "Accepted",
    "contactorId": 1,
    "daySchedule": [
        {
            "start": "20:11:00",
            "end": "23:00:00"
        },
        {
            "start": "05:30:00",
            "end": "07:00:00"
        }
    ]
}
```

7.5. GetFile

7.5.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetFileRequest",
    "type": "object",
    "properties": {
        "requestId": {
            "type": "string",
            "maxLength": 36
        },
        "fileType": {
            "type": "string",
            "enum": [
                "Syslog",
                "MeterValueHistory",
                "VendorSpecific"
            ]
        },
        "location": {
            "type": "string",
            "format": "uri"
        },
        "startTime": {
            "type": "string",
            "format": "date-time"
        },
        "stopTime": {
            "type": "string",
            "format": "date-time"
        }
    },
    "additionalProperties": false,
    "required": [
        "fileType",
        "location"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "GetFileResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected"
            ]
        },
        "fileName": {
            "type": "string"
        }
    },
    "additionalProperties": false,
        "required": [
        "status",
        "fileName"
    ]

}
```

7.5.2. examples

request:

```
{
  "requestId": "08fb86a4-3e9b-11e6-ac61-9e71128cae77",
  "fileType": "MeterValueHistory",
  "location": "ftp://root:root@sftp.oursystem.foo/tmp",
  "startTime": "2013-02-01T15:09:18Z",
  "stopTime": "2013-02-01T15:09:18Z"
}
```

response:

```
{
  "status": "Accepted",
  "fileName": "MeterValuesHistory=20160515-20160615.txt"
}
```

7.6. OverrideStreetlight

This method is used to remotely override a streetlight schedule. You can use this to force streetlights on or off. The streetlights are kept on (or off) until you resume the schedule with a ResumeStreetlight message.

> Use action `Suspend` to safely install a new schedule without effecting the current state.

Table 11. OverrideStreetlight request parameters

parameter	description
`contactorId`	1-4 indicates which contactorId to override, leaving this field out implies that this actions is desired for **all** contactorIds.
`action`	`"On"` force lights on. `"Off"` force lights off. `"Suspend"` suspend schedule, keep lights in current state.

Table 12. OverrideStreetlight response `status` parameter

status value	description
`Accepted`	command is accepted
`RejectedLocalOverride`	command is rejected, because a local (by hardware switch) override is active now
`Rejected`	command is rejected for unknown reason

When the override mode is successfully enabled a StreetlightModeStatusNotification will be send with mode `RemoteOverride`.

7.6.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "OverrideStreetlightRequest",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "action": {
            "type": "string",
            "enum": [
                    "On",
                    "Off",
                    "Suspend"
                    ]
        }
    },
    "additionalProperties": false,
    "required": [
        "action"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "OverrideStreetlightResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected",
                "RejectedLocalOverride"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.6.2. examples

request:

```
{
    "contactorId": 1,
    "action": "On"
}
```

response:

```
{
    "status": "Accepted"
}
```

7.7. ResumeStreetlight

This message stops a remote override state, that was initiated before with OverrideStreetlight. When this call succeeds the station follows the schedule again.

`contactorId`	1-4 indicates which contactorId to override, leaving this field out implies that this actions is desired for **all** contactorIds.

Table 13. ResumeStreetlight response `status` parameter

status value	description
`Accepted`	command is accepted
`RejectedLocalOverride`	command is rejected, because a local (by hardware switch) override is active now
`Rejected`	command is rejected for unknown reason

When the resume action is completed StreetlightModeStatusNotification will be send with the current mode.

7.7.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ResumeStreetlightRequest",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        }
    },
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "OverrideStreetlightResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected",
                "RejectedLocalOverride"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.7.2. examples

request:

```
{
    "contactorId": 1,
    "type": "Immediate"
}
```

response:

```
{
    "status": "Accepted"
}
```

7.8. Reset

Trigger a hardware reset.

7.8.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ResetRequest",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ResetResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.8.2. examples

request:

```
{
  "type": "Soft"
}
```

response:

```
{
  "status": "Accepted"
}
```

7.9. ResetMVFaultIndicator

Reset all the Medium Voltage Fault indicators.

A reset of an indicater after a fault has been fixed. Or if a test was issued. If a ResetMVFaultIndicator is issued while the cause of an alarm was not fixed, the Indicator will re-enter the fault state, and send again a StatusNotification indicating the error.

Note: an Opto is an example of a Medium Voltage Fault indicator.

See related message: TestMVFaultIndicator

Table 14. ResetMVFaultIndicator response `status` parameter

status value	description
`Accepted`	command is accepted
`Rejected`	No Medium Voltage indicator device is available

7.9.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ResetMVFaultIndicatorRequest",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "ResetMVFaultIndicatorResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.9.2. examples

request:

```
{}
```

response:

```
{
  "status": "Accepted"
}
```

7.10. SetStreetlightProfile

Write a new profile to the station. This can be added additional to the current schedules, or it can install a new set (existing schedules are erased), depending on the `clearAllSchedules` parameters.

We specify a schedule per day. To ensure we can properly schedule periods during a night, a day, in SetStreetlightProfile context, starts at noon 12:00 and ends the next day at 11:59:59.
use the `clearAllSchedules` parameter to safely install a new schedule and prevent that a station is left without a schedule.
when using the `clearAllSchedules` and you have to issue `SetStreetlightProfile` multiple times, (when the messages are becoming too big for the station), ensure to send the basic profile first, a catch all sane default. This means you probably want to send the lowest stacklevels with the first `SetStreetlightProfile` call (the one that also has `clearAllSchedules`).
when you have to issue `SetStreetlightProfile` multiple times to set a complete profile, you can use OverrideStreetlight with action `Suspend`, **before** you start, to prevent switching the lights unintendedly when the sequence is not complete yet. To resume scheduling issue a ResumeStreetlight when the complete profile is set.

7.10.1. Details

Table 15. SetStreetlightProfile top level parameters

`contactorId`	indicates for which contactorId these profile is intended.
`clearAllSchedules`	(optional) when this is `true`, all profiles for this contactorId are erased, before the new profiles are installed. The erase and setting of the new profiles is atomic, either both succeed, or both fail. defaults to `false`.

A combined streetlight profile consists of a set of streetlight profiles, each with a unique `profileId` and `stackLevel`.

Table 16. SetStreetlightProfile parameters

`profileId`	unique id, can be used to delete an individual profile
`stackLevel`	a higher number has higher precedence, must be unique, two profiles with the same stack level are not allowed. must be lower or equal than `StreetlightProfileMaxStackLevel` configuration setting.
`recurrencyKind`	indicates if daySchedule should be daily, weekly or not recurring
`validFrom`	schedule is not valid before this date. This date is also used to determine on which day-of-the-week a weekly recurring schedule should be active. We only take into account the date part, the time part in validFrom is ignored. NB: the schedule will be active on the specified date, but only from noon 12:00 and later. (as before 12:00 is seen as the previous day).
`validTo`	(optional) schedule is not valid/used after this time. We only take into account the date part, the time part in validFrom is ignored. We use this as inclusive end date, ending the next day at 11:59:59

Table 17. daySchedule parameters

`start`	Indicates the time the `On` period starts
`end`	Indicates the time the `On` period ends

Table 18. daySchedule start/end parameters

`kind`	Indicates how the time is specified. "Absolute" means, time is passed as local time. "Sunset" means time is given relative to the sunset. "Sunrise" means time is given relative to the sunrise time. The station will calculate the sunrise/sunset times daily based on the location and date.
`time`	mandatory when `kind` is "Absolute". Indicates local time when action should occur. must be in format HH:mm:ss in 24 hour format: eg "21:03:00" or "06:30:59", start of day is "12:00:00" end of day is "11:59:59". You can optinally add ms, eg: "21:03:00.500".
`offset`	mandatory when `kind` is "Sunset" or "Sunrise", indicates time offset in **seconds** when action should occur. Negative values are allowed and mean *before*.

If a daySchedule period `end` time is before the `start` time, the light will not be switched on. So it it safe to use the `Sunset` or `Sunrise` offsets, if they cross an absolute time, the lights will be kept off.

7.10.2. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "SetStreetlightProfileRequest",
    "type": "object",
    "properties": {
        "contactorId": {
            "type": "integer",
            "minimum": 1,
            "maximum": 4
        },
        "clearAllSchedules": {
            "type": "boolean"
        },

        "streetlightProfiles": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "profileId": {
                        "type": "integer",
                        "minimum": 0
                    },
                    "stackLevel": {
                        "type": "integer",
                        "minimum": 0
                    },
                    "recurrencyKind": {
                        "type": "string",
                        "enum": [
                            "None",
                            "Daily",
                            "Weekly"
                        ]
                    },
                    "validFrom": {
                        "type": "string",
                        "format": "date-time"
                    },
                    "validTo": {
                        "type": "string",
                        "format": "date-time"
                    },
                    "daySchedule":  {
                        "type": "array",
                        "items":  {
                            "type": "object",
                            "properties" : {
                                "start": {
                                    "type": "object",
                                    "properties": {
                                        "kind": {
                                            "type": "string",
                                            "enum": ["Absolute",
"Sunset", "Sunrise"]
                                        },
                                        "offset": {
                                            "type": "integer",
                                            "minimum": -86400,
                                            "maximum": 86400
                                        },
                                        "time": {
                                            "type": "string"
                                        }
                                    },
                                    "required": [ "kind"],
                                    "additionalProperties": false
                                },
                                "end": {
                                    "type": "object",
                                    "properties": {
                                        "kind": {
                                            "type": "string",
                                            "enum": ["Absolute",
"Sunset", "Sunrise"]

                                        },
                                        "offset": {
                                            "type": "integer",
                                            "minimum": -86400,
                                            "maximum": 86400
                                        },
                                        "time": {
                                            "type": "string"
                                        }
                                    },
                                    "required": [ "kind"],
                                    "additionalProperties": false
                                }
                            },
                            "required": [ "start", "end" ],
                            "additionalProperties": false
                        },
                        "minItems": 1,
                        "maxItems": 4
                    }
                },
                "required": [
                    "profileId",
                    "stackLevel",
                    "recurrencyKind",
                    "daySchedule"
                ]
            },
            "minItems": 1
        }
    },
    "additionalProperties": false,
    "required": [
        "contactorId",
        "streetlightProfiles"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "SetStreetlightProfileResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.10.3. examples

request:

```
{
    "contactorId": 1,
    "clearSchedule": true,
    "streetlightProfiles": [
      {
        "profileId" : 1,
        "stackLevel": 2,
        "recurrencyKind": "Daily",
        "validFrom": "2015-06-06T00:00:00",
        "validTo": "2019-09-06T00:00:00",
        "daySchedule": [
            {
               "start": {
                 "kind": "Sunset",
                 "offset": 3600
                },
                "end": {
                  "kind": "Absolute",
                  "time": "01:00:00"
                }
            }, {
               "start": {
                 "kind": "Absolute",
                 "time": "06:00:00"
                },
                "end": {
                  "kind": "Sunrise",
                  "offset": 0
                }
             }
        ]
      }
    ]
}
```

response:

```
{
    "status": "Accepted"
}
```

7.11. TestMVFaultIndicator

Put all Medium Voltage Fault indicators in test mode. This will trigger an "OverCurrentFailure" and (optionally depending on the hardware) a "GroundFailure", and one (or two) StatusNotification will be send with this status This will only be restored after a ResetMVFaultIndicator message.

If the station detects during testing that an opto is not working correctly a StatusNotification with status `MVFaultIndicatorError` is send.

Note: an Opto is an example of a Medium Voltage Fault indicator.

Note: The number of StatusNotification messages depends on how many, and which indicators are installed. See checkout `MVFaultIndicatorError` and friend at <Configuration_TODO>> .

Any resulting error StatusNotification from a TESTMVFaultIndicator will have the `"test"` field set to `true`.

See related message: ResetMVFaultIndicator

Table 19. TestMVFaultIndicator response `status` parameter

status value	description
`Accepted`	command is accepted
`RejectedAlreadyFaulted`	One (or more) Indicators are already in alarm state (due to fault or due to test), Cannot start test.
`Rejected`	No Medium Voltage indicator device is available

7.11.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "TestMVFaultIndicatorRequest",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "TestMVFaultIndicatorResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "RejectedAlreadyFaulted",
                "Rejected"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.11.2. examples

request:

```
{}
```

response:

```
{
  "status": "Accepted"
}
```

7.12. TriggerMessage

7.12.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "TriggerMessageRequest",
    "type": "object",
    "properties": {
        "requestedMessage": {
                "type": "string",
                "enum": [
                    "BootNotification",
                    "GetFileStatusNotification",
                    "FirmwareStatusNotification",
                    "Heartbeat",
                    "MeterValues",
                    "StatusNotification",
                    "StreetlightModeStatusNotification",
                    "StreetlightStatusNotification"
                ]
            },
        "connectorId": {
            "type": "integer"
        }
    },
    "additionalProperties": false,
    "required": [
        "requestedMessage"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "TriggerMessageResponse",
    "type": "object",
    "properties": {
        "status": {
            "type": "string",
            "enum": [
                "Accepted",
                "Rejected"
            ]
        }
    },
    "additionalProperties": false,
    "required": [
        "status"
    ]
}
```

7.12.2. examples

request:

```
{
    "requestedMessage": "StatusNotification",
    "connectorId": 2
}
```

response:

```
{
    "status": "Accepted"
}
```

7.13. UpdateFirmware

7.13.1. schemas

request:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "UpdateFirmwareRequest",
    "type": "object",
    "properties": {
        "retrieveDate": {
            "type": "string",
            "format": "date-time"
        },
        "location": {
            "type": "string",
            "format": "uri"
        },
        "retries": {
            "type": "integer"
        },
        "retryInterval": {
            "type": "integer"
        },
        "requestId": {
            "type": "string",
            "maxLength": 36
        }
    },
    "additionalProperties": false,
    "required": [
        "retrieveDate",
        "location"
    ]
}
```

response:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "UpdateFirmwareResponse",
    "type": "object",
    "properties": {},
    "additionalProperties": false
}
```

7.13.2. examples

request:

```
{
  "retrieveDate": "2013-02-01T15:09:18Z",
  "location": "ftp://root:root@fork.gir.foo/tmp/kvcbx-updt.amx",
  "retries": 4,
  "retryInterval": 20
}
```

response:

```
{}
```

8. Data types

8.1. contactorId

Some messages have a contactorId field. The table below shows the possible values and their meaning.

Table 20. contactorId

nr	description
1	Evening/Morning
2	Night
3	Tariff
4	<unspecified>

8.2. latlng

A geo-location string, with format "**[** <real number> **,** <real number> **]**"